TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA TOÁN - TIN HỌC

TIỂU LUẬN MÔN HỌC

Lập trình Hướng đối tượng (TN2301D)

ĐỀ TÀI:

Lập trình trò chơi Flappy Bird trên Unity

SINH VIÊN THỰC HIỆN:

TRẦN DUY THANH

MSSV 2015830
LỚP TNK44

Đà Lạt, 19/12/2022

TÓM TẮT

Flappy Bird là một trò chơi điện tử trên điện thoại do anh Nguyễn Hà Đông - lập trình viên Việt Nam phát triển,phát hành vào năm 2013. Nguyễn Hà Đông tạo ra Flappy Bird trong vòng một vài ngày. Vào cuối tháng 01/2014, Flappy Bird là ứng dụng được tải về nhiều nhất trên App Store cũng như trên Google Play.

Flappy Bird đã bị gỡ xuống trên cửa hàng App Store và Google Play bởi chính tác giả vào ngày 10/02/2014, do những vấn đề xã hội của nó - mà như tác giả cảm thấy - là gây nghiện và bị lạm dụng quá mức.

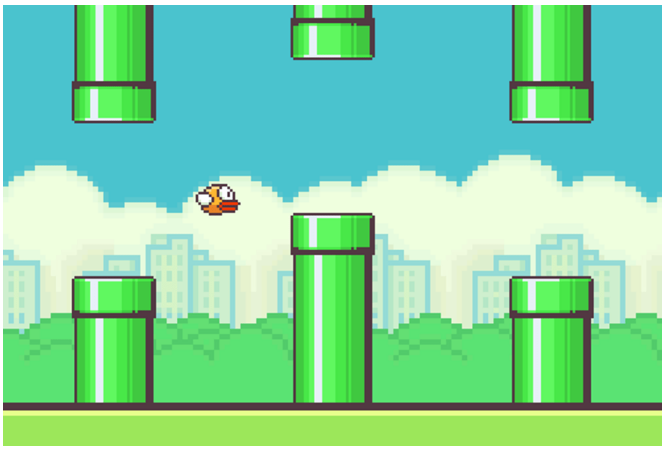
Nhìn chung, Flappy Bird thú vị và khá dễ để bắt đầu học và thực hành lập trình game. Trong phạm vi tiểu luận này, tôi sẽ viết lại Flappy Bird trên Unity.

1. Tổng quan về trò chơi FlappyBird.
2. Lập trình Flappy Bird trên Unity.
3. Cấu trúc mã nguồn C#.

Mục lục

Phần 1

Tổng quan về trò chơi FlappyBird



Hình 1.1: Trò chơi Flappy Bird trên máy tính bảng

1.1 Luật chơi

Có 2 loại đối tượng: "chú chim" và các "ống cống".

Nếu không tác động gì, chú chim sẽ rơi tự do. Người chời cần chạm vào màn hình (trên điện thoại thông minh hoặc máy tính bảng) hoặc gõ phím "khoảng trắng" (space) (trên web hoặc máy tính để bàn) để kích chú chim bay vọt lên. Mỗi lần chạm hoặc gõ phím "khoảng trắng" thì chú chim sẽ nhảy 1 lần.

Trò chơi kết thúc khi chú chim chạm đất hoặc chạm phải bất kì ống cống nào.

Mỗi khi vượt qua mỗi ống cống (hoặc cặp ống cống trên dưới) thì tích lũy thêm 1 điểm.

Phần 2

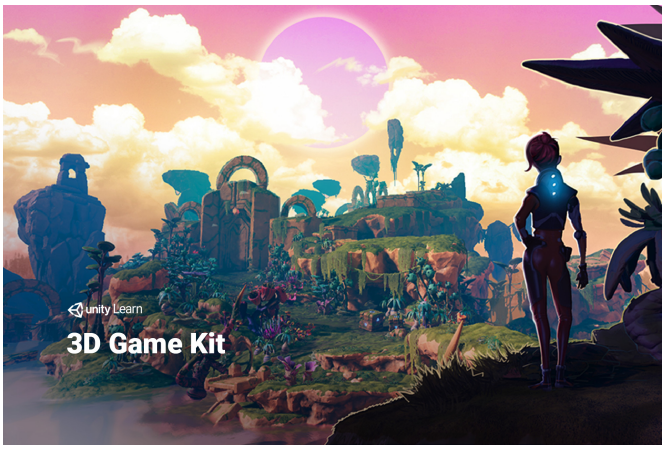
Lập trình Flappy Bird trên Unity

2.1 Tổng quan về Unity

Unity là một trình-viết-trò-chơi (Game Engine) đa xuyên nền tảng (cross-platform) do Unity Technologies phát triển, mà chủ yếu để phát triển video game cho máy tính, consoles và điện thoại. Lần đầu tiên nó được công bố chạy trên hệ điều hành OS X, tại Apple's Worldwide Developers Conference vào năm 2005, đến nay đã mở rộng 27 nền tảng.

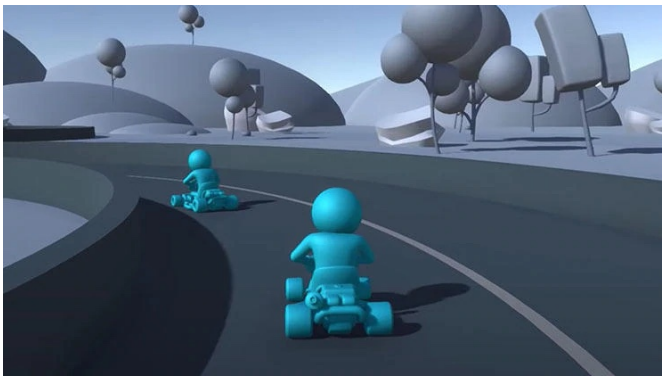
Hình 2.1: Trò chơi đồ họa 2D viết bằng Unity

Unity hỗ trợ đồ họa 2D và 3D, các chức năng được viết chủ yếu qua ngôn ngữ C#. Unity nổi bật với khả năng xây dựng trò chơi chạy trên nhiều nền tảng. Các nền tảng được hỗ trợ hiện nay gồm: Android, Android TV, Facebook Gameroom, Fire OS, Gear VR, Google Cardboard, Google Daydream, HTC Vive, iOS, Linux, macOS, Microsoft HoloLens, Nintendo 3DS family Nintendo Switch, Oculus Rift, PlayStation 4, PlayStation Vita, PlayStation VR, Samsung Smart TV, Tizen, tvOS, WebGL, Wii U, Windows, Windows Phone, Windows Store, và Xbox One.



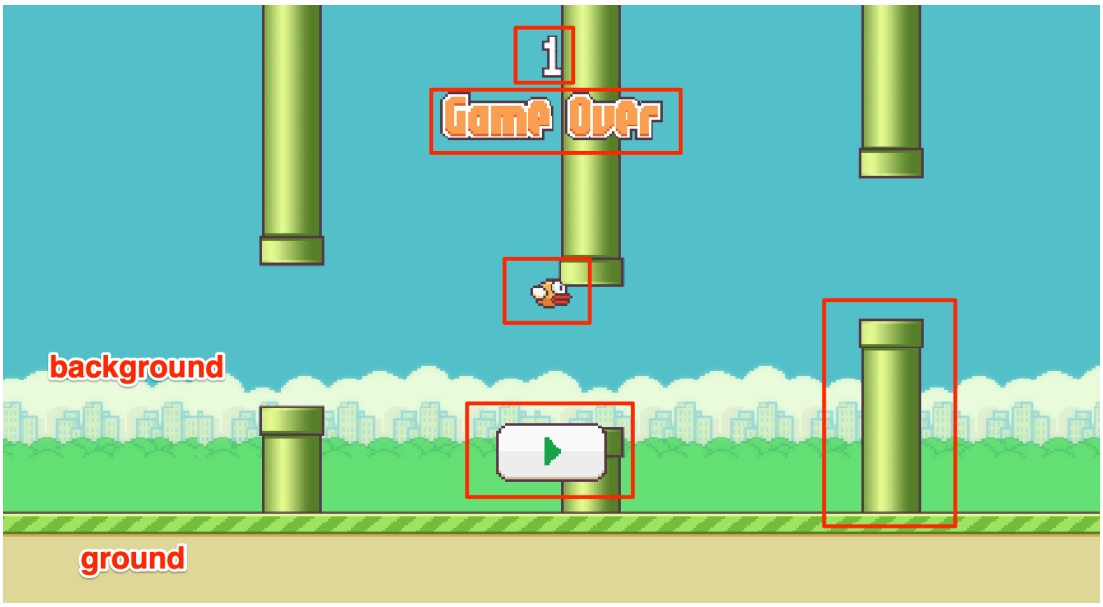
Hình 2.2: Trò chơi đồ họa 3D viết bằng Unity

2.2 Unity và Học sâu



Hình 2.3: Một trò chơi trong ML-Agents

Sử dụng môi trường giả lập để huyến luyện các mạng neuron học sâu đang là xu hướng, bởi nó có thể giúp tiết kiệm chhi phí huấn luyện, thu thập mẫu dữ liệu. Ngoài ra, môi trường giả lập còn tạo ra môi trường kiểm thử, đánh giá thống nhất để tiện bề so sánh hiệu quả, hiệu suất giữa các giải thuật với nhau. Unity cho ra đời ML-Agents nhằm mục đích này. ML-Agents hỗ trợ các lập trình viên có thể nhanh chóng tích hợp môi trường trò chơi giả lập này để huấn luyện các mạng neuron trong các giải thuật học tăng cường [Uni22b]



Hình 2.4: Các đối tượng trong Flappy Bird

2.3 Cấu trúc trò chơi Flappy Bird và các bài toán lập trình chủ yếu

Theo hình 2.4, Ta sẽ phân chia Flappy Bird thành các đối tượng sau:

1. *Ground*: Nền đất.
2. *Background*: Hình nền, là cảnh quan trời mây bãi cỏ nhà cửa phía xa.
3. *Bird*: Chú chim bay nhảy của chúng ta.
4. *Pipe*: Ống cống.
5. *PlayButton*: Nút ấn cho phép bắt đầu trò chơi.
6. *GameOver*: Dòng chữ thông báo kết thúc trò chơi.
7. *Score*: Phần hiển thị điểm số.

Khi đó, trò chơi Flappy Bird chính là sự chuyển động, tương tác giữa các đối tượng trên.

Các bài toán lập trình chủ yếu bao gồm:

1. *Bắt và xử lí sự kiện "gõ phím" hoặc "nhấp chuột"* để người chơi có thể can thiệp vào chuyển động của đối tượng `Bird` - chú chim.

2. *Rơi tự do.* Khi không có tác động nào, chú chim - `Bird` sẽ rơi tự do. Vậy ta sẽ cần một phương thức C# tương ứng với nhiệm vụ đó. Về phương ngang, thực ra chú chim không thay đổi tọa độ. Chú chim chỉ chuyển động theo phương đứng mà thôi.

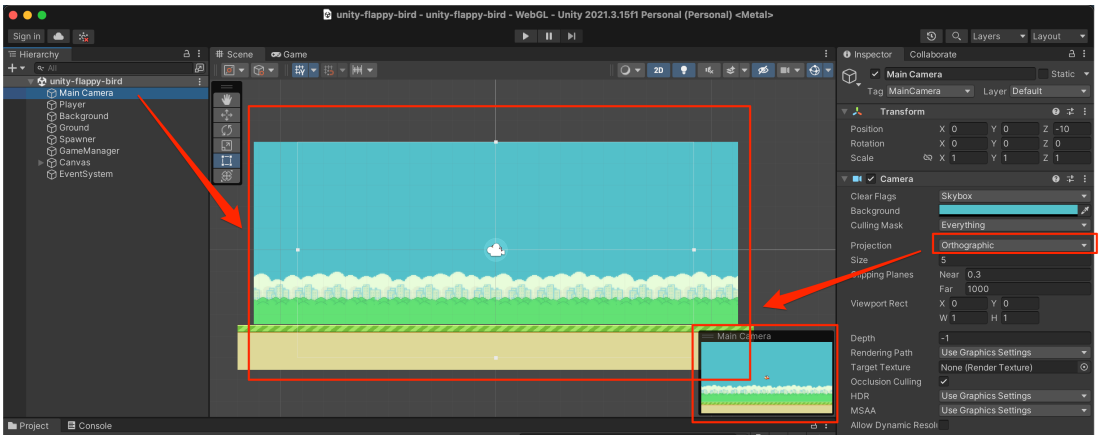
3. *Sinh ngẫu nhiên các ống cống* - ***`Pipe`***. Đây là phần thử thách của trò chơi, độ khó của trò chơi cũng phụ thuộc vào các thông số (khoảng cách giữa các cặp ống cống, chênh lệch giữa các khe hở trước và sau...) sinh ngẫu nhiên các ống cống - *Pipe* như thế này.

4. *Dịch chuyển đối tượng theo thời gian.* Cả nền đất - *Ground*, hình nền - *Background*, các ống cống - *Pipe* đều cần chuyển động sang trái theo thời gian với tốc độ khác nhau.

2.4 Biểu diễn các đối tượng trong Unity

2.4.1 Các cấu phần giao diện trong Unity

2.4.1.1 Scene



Hình 2.5: Flappy Bird chỉ sử dụng `Orthographic` và có duy nhất 1 `Scene`.

Không gian trò chơi có thể rất lớn, nhưng ngay khi chạy chương trình thì người chơi chỉ nhìn thấy một phần của nó. Unity gọi đó là `Scene`. Một trò chơi có thể có nhiều `Scene`.

Mỗi `Scene` sẽ này ứng với một góc nhìn nhất định. `Scene` mặc định là lấy theo góc nhìn trực diện `Orthographic`. Unity coi như góc nhìn đó là hệ quả thu

hình của 1 `Camera`. Bạn có thể tùy chỉnh giữa `Orthographic` và `Perspective`. Trong đó `Perspective` cho ra `Scene` trông như $2.5D$ hoặc $3D$.

Flappy Bird chỉ sử dụng `Orthographic` và có duy nhất 1 `Scene`. Xem hình 2.5. Và tất cả các đối tượng ta đã đề cập ở 2.3 đều sẽ xuất hiện trên `Scene` này.

2.4.1.2 Sprite

Hình 2.6: Các hình png dùng làm sprite cho các đối tượng trong FlappyBird

Một đối tượng đồ họa 2D sẽ tương ứng với 1 `Sprite`. Unity hỗ trợ ta thiết kế các đối tượng đồ họa đó, hoặc ta có thể thiết kế trên phần mềm khác và kết xuất (export) thành các tập tin (file) hình ảnh và bỏ vào đúng chỗ trong `Project`. Trong tiểu luận này, tôi sử dụng các hình thiết kế sẵn. Như vậy ta sẽ có các `Sprite` tương ứng với các danh sách đối tượng 2.3 như hình 2.6. Lưu ý:

1. *Score* không có đồ họa tương ứng, do đó sẽ không có `Sprite` tương ứng với nó. Ta sẽ biểu diễn *Score* bằng `Text` trên `Canvas` của Unity, sẽ đề cập ở phần sau.

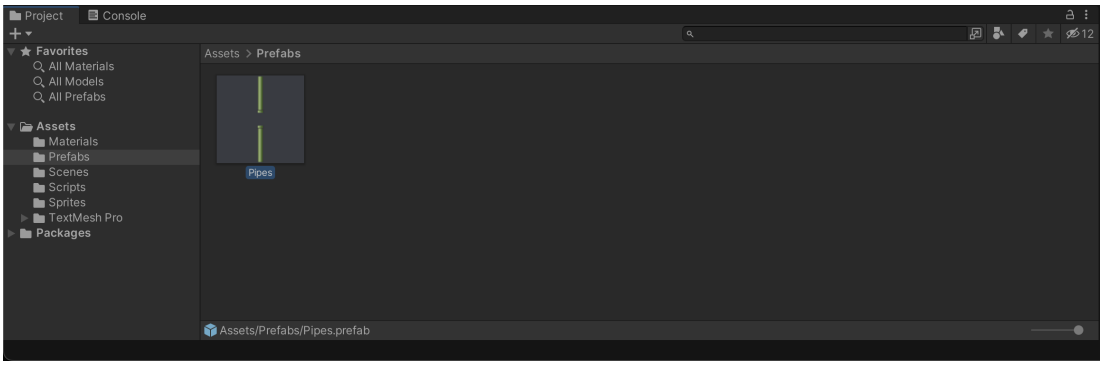
2. Chú chim - *Bird* của chúng ta sẽ có 3 `Sprite` tương ứng. Khi liên tiếp hiển thị chuỗi 3 hình này, ta sẽ sẽ tạo ra chuyển động vỗ cánh, làm trò chơi thêm sinh động. Ta sẽ đề cập nó ở phần sau.

2.4.1.3 Prefab

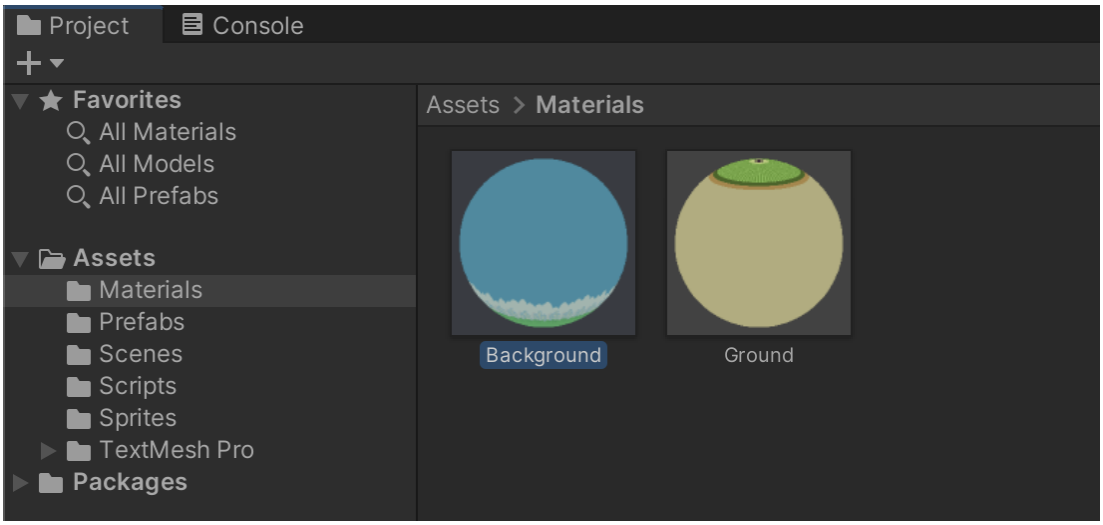
Một số đồ họa phức tạp gồm nhiều đồ họa đơn giản hợp thành thường đi chung với nhau thì nên gom vào một nhóm, gọi là `Prefab`. Trong Flappy Bird, ta thấy các ống cống - `Pipe` thường đi thành cặp, 1 ống ở dưới, 1 ống ở trên, ở giữa có khe hở. Chính vì vậy, ta sẽ gom cụm và tạo thành 1 `Prefab tương ứng cho nó`, xem hình 2.7.

2.4.1.4 Material

Để ý thấy rằng nền đất - *Ground* và hình nền *Background* chỉ có kích thước hữu hạn, nhưng lại dịch trái liên tục trong suốt trò chơi.



Hình 2.7: Prefab



Hình 2.8: Material sử dụng trong Flappy Bird

Unity hỗ trợ các chuyển động đồ họa kiểu này bằng `Texture` trong `Material`. Có thể hiểu nôm na là các đồ họa này sẽ được nhân bản, nối tiếp nhau, cuộn tròn lại. Quá trình dịch trái sẽ biến thành quá trình xoay sang trái và chiếu lên `Scene`.

Vì thế, thực chất, đối tượng nền đất - *Ground* và hình nền - *Background* của Flappy Bird sẽ tương ứng với `Material Ground` và `Material Background` trong Unity như trong hình 2.8.

2.4.1.5 Canvas và Button, Image, Text

`Canvas` là là một cấu phần giao diện (UI component) mà ta có thể đặt các cấu phần giao diện khác có chung nhiều thông số cài đặt (setting) lên. Ta cũng có thể hiểu `Canvas` là một cách gom nhóm nhiều cấu phần giao diện nhỏ lẻ.

Trong Flappy Bird, ta cần `Canvas` để đặt:

- Cấu phần giao diện `Button` ứng với *PlayButton*
- Cấu phần giao diện `Image` ứng với *GameOver*
- Cấu phần giao diện `Text` ứng với *Score*

Đến đây ta có bảng tổng hợp sau:

Đối tượng trong Flappy Bird	Đồ họa	Các cấu phần tương ứng trong Unity
Ground	Ground.png	`Material`
Background	Background.png	`Material`
Bird	Bird_01.png Bird_02.png Bird_03.png	`Sprite`
Pipe	Pipe.png	`Prefab`
PlayButton	PlayButton.png	`Button` trên `Canvas`
GameOver	GameOver.png	`Image` trên `Canvas`
Score		`Text` trên `Canvas`

Bảng 2.1: Các đối tượng trong Flappy Bird và cấu phần tương ứng trong Unity

2.4.2 Các cấu phần vật lí trong Unity

Nếu các cấu phần giao diện đồ họa là cái đẹp đẽ trưng ra trước mặt người chơi, thì khối hình vật lí dùng trong các logic tính toán va chạm, tác động, rơi tự do, bay lượn... lại thường là một cấu phần vật lí (Physics component) được khai báo kèm theo cấu phần giao diện đó. Ở đây, cấu phần vật lí đó trong Unity là `Rigidbody`. Các điều khiển vật lí trên Unity sẽ gắn với `Rigidbody`.

Một cấu phần vật lí khác là `Collider` chịu trách nhiệm kích hoạt các sự kiện chồng lấn giữa các đối tượng trong trò chơi. Nếu đã từng chơi trò chơi, tinh ý một chút, bạn sẽ nhận ra, đây là lí do vì sao nhiều khi chơi trò chơi, bạn thấy rõ ràng vũ khí đã chạm vào kẻ địch nhưng chương trình lại không ghi nhận tương tác đó. Đơn giản là vì `Collider` vũ khí của bạn chưa hề đụng đến `Collider` của kẻ địch, chỉ có cấu phần giao diện chồng đè lên nhau mà thôi. Trong các trò chơi

đơn giản thì `Collider` thường có phạm vi và hình dạng cố định, trong khi cấu phần giao diện lại chạy nhiều hiệu ứng biến đổi phong phú hơn nhiều. Các trò chơi phức tạp hơn sẽ cho `Collider` theo sát với kết cấu của cấu phần giao diện.

Trong Flappy Bird

- Ứng với *Bird* là một `Circle Collider 2D` và một `Regidbody 2D` với thuộc tính `Body Type` là `Kinematic`.
- Ứng với *Ground* là một `Box Collider 2D`
- Ứng với mỗi *Pipe* là một `Box Collider 2D`. Một cặp *Pipe* sẽ có 1 cặp `Box Collider 2D` tương ứng. Khoảng trống ở giữa cặp *Pipe* cũng có một `Box Collider 2D`. Ta sẽ cần Collider này để kích hoạt quá trình tính điểm khi `Circle Collider 2D` của *Bird* vượt qua.

Đến đây, ta có bảng sau:

Đối tượng trong Flappy Bird	Đồ họa	Các cấu phần tương ứng trong Unity
Ground	Ground.png	`Material` `Box Collider 2D`
Background	Background.png	`Material`

Bird	Bird_01.png
	Bird_02.png
	Bird_03.png
		`Sprite`
		`Rigidbody 2D`
		`Circle Collider 2D`

Pipe	Pipe.png	`Prefab` 3 `Box Collider 2D`
PlayButton	PlayButton.png	`Button` trên `Canvas`
GameOver	GameOver.png	`Image` trên `Canvas`
Score		`Text` trên `Canvas`

Bảng 2.2: Các đối tượng trong Flappy Bird và cấu phần tương ứng trong Unity

2.4.3 Điều khiển đối tượng bằng C#

Trong Flappy Bird

- Ứng với *Bird* sẽ có `Player.cs`

- Ứng với *Ground* sẽ có `Ground.cs`
- Ứng với *Background* sẽ có `Parallax.cs`
- Ứng với mỗi cặp ống cống - *Pipe* sẽ có `Pipes.cs`

Ngoài ra:

- Để điều khiển việc sinh các cặp *Pipe* mới ngẫu nhiên và hủy các cặp *Pipe* cũ (khi đã vượt ra ngoài màn hình bên trái), ta có thêm `Spawner.cs`.
- Mỗi trò chơi sẽ có một tập tin C# đặc biệt gọi là `GameManager` đóng vai trò khởi tạo và điều khiển các quá trình tổng thể như *PlayButton*, *GameOver*, *Score*.

Đối tượng trong Flappy Bird	Đồ họa	Các cấu phần tương ứng trong Unity	Mã C# điều khiển tương ứng
Ground	Ground.png	`Material` `Box Collider 2D`	`Ground.cs`
Background	Background.png	`Material`	`Parallax.cs`

Bird	Bird_01.png
	Bird_02.png
	Bird_03.png
		`Sprite`
		`Rigidbody 2D`
		`Circle Collider 2D`
			`Player.cs`

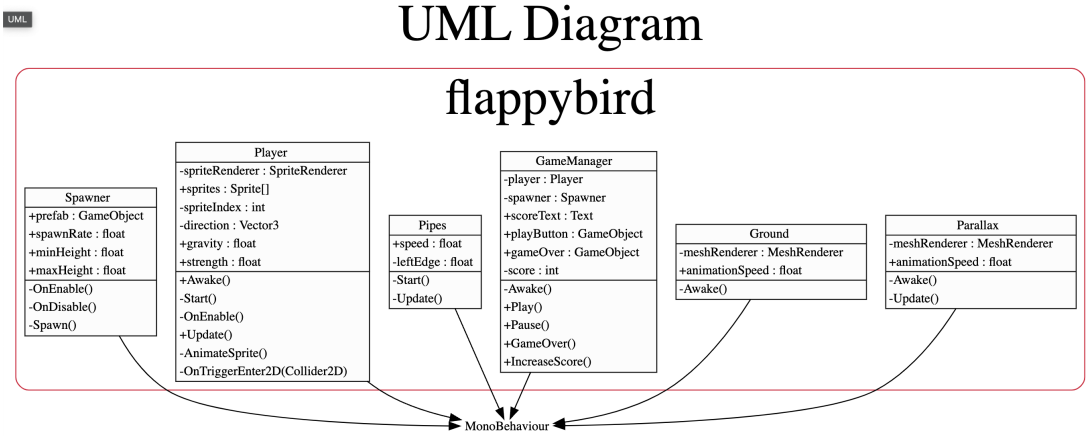
Pipe	Pipe.png	`Prefab` 3 `Box Collider 2D`	`Pipes.cs` `Spawner.cs`
PlayButton	PlayButton.png	`Button` trên `Canvas`	`GameManager.cs`
GameOver	GameOver.png	`Image` trên `Canvas`	`GameManager.cs`
Score		`Text` trên `Canvas`	`GameManager.cs`

Bảng 2.3: Các đối tượng trong Flappy Bird và cấu phần tương ứng trong Unity

Phần 3

Cấu trúc mã nguồn C#

3.1 MonoBehaviour



Hình 3.1: Cấu trúc mã nguồn trò chơi Flappy Bird trên Unity

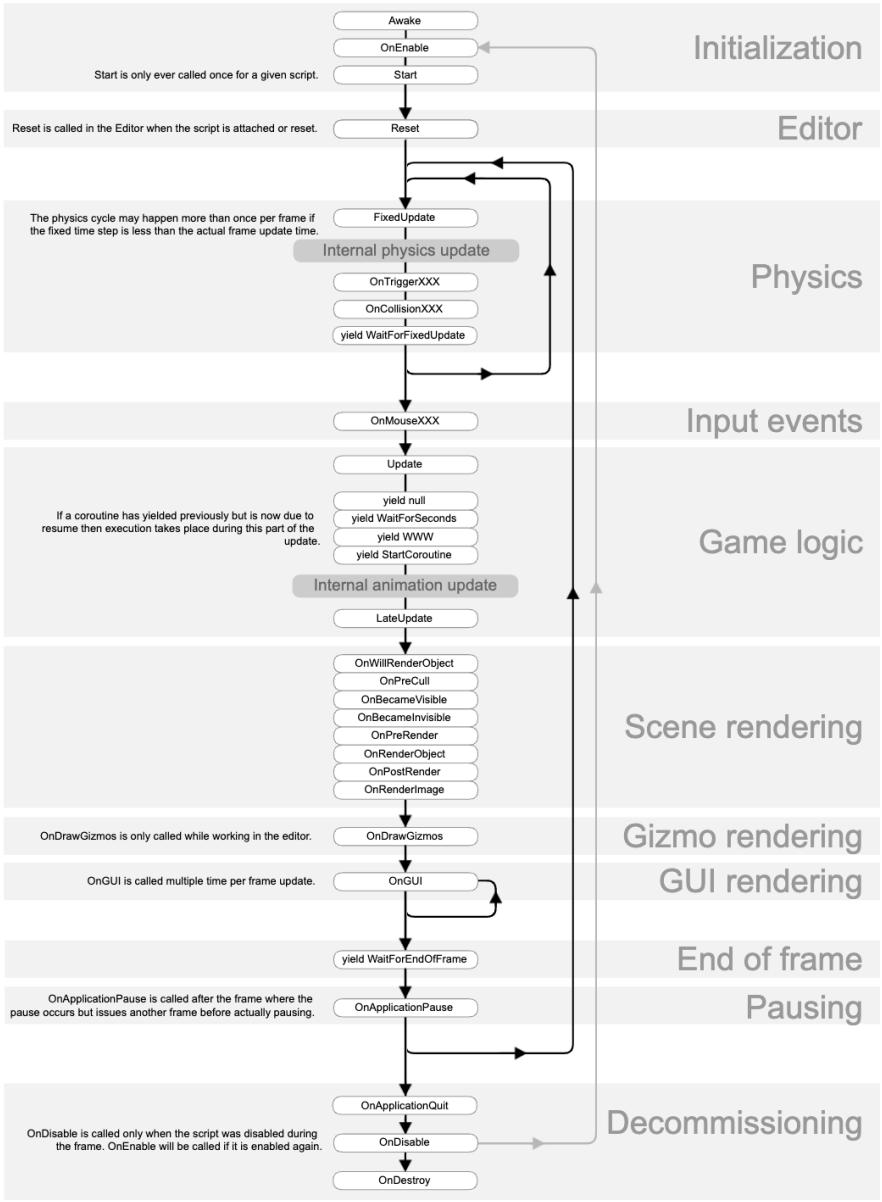
Tất cả các lớp đối tượng đều kế thừa `MonoBehaviour`. Đây là lớp đối tượng đặt biệt của Unity. Xem tại đây [Uni22a]. `MonoBehaviour` là lớp đối tượng cơ sở mà mọi mã nguồn C# ứng với đối tượng nào trong trò chơi cũng đều phải kế thừa và mở rộng từ đó.

Kết cấu của một giải thuật trò chơi thường là tính toán và kết xuất ra các khung hình (frame) liên tiếp nhau với tốc độ phù hợp (framerate).

Trong Unity, trước khi kết xuất 1 khung hình, chương trình sẽ duyệt qua và tính toán các thay đổi, các hành vi của tất cả các thể hiện (instance) của tất cả các lớp đối tượng (class) kế thừa `MonoBehaviour`. Trong trường hợp Flappy Bird này chính là tính toán trên: `GameManager`, `Player`, `Ground`, `Parallax`, `Pipes`,

`Spawner`.

`MonoBehaviour` dựng sẵn những mốc sự kiện (hook) và sẽ định kì kích hoạt vào một giai đoạn nào đó trong chu trình hoạt động (life-cycle) của nó. Xem hình 3.2



Hình 3.2: Chu trình hoạt động của lớp đối tượng `MonoBehaviour`

3.2 Các mốc sự kiện dùng trong Flappy Bird

3.2.1 Awake

Unity sẽ gọi phương thức Awake mỗi khi bắt đầu tạo ra một thể hiện (instance) mới. Tức là nó chỉ chạy 1 lần duy nhất trên mỗi thể hiện. Thường dùng để khởi tạo các cài đặt liên quan đến thể hiện đó mà thôi.

3.2.2 OnEnable và OnDisable

Unity sẽ gọi 2 phương thức này tương ứng mỗi khi thể hiện được kích hoạt hoặc bất hoạt. Cụ thể, khi `PlayButton` và `GameOver` chuyển từ trạng thái ẩn sang trạng thái hiện ra trên `Scene` thì Unity sẽ gọi phương thức `OnEnable`, nếu ẩn 2 thể hiện này đi thì sẽ gọi phương thức `OnDisable`. Trong Flappy Bird, ta sử dụng để tạm dừng trò chơi khi kết thúc (chú chim chạm đất hoặc chạm ống cống) và bắt đầu lại trò chơi khi người chơi nhấp chuột vào `PlayButton`.

3.2.3 OnTriggeredEnter2D

Mỗi khi Unity phát hiện chồng lấn giữa 2 cấu phần `Collider2D`.

3.2.4 Start

Chỉ gọi trên mỗi khung hình ngay trước lúc gọi phương thức `Update` lần đầu tiên.

3.2.5 Update

Unity sẽ gọi phương thức này theo mỗi khung hình khi `MonoBehaviour` được kích hoạt.

3.3 Toàn văn mã nguồn C#

GameManager.cs

- `Awake`: Thiết lập cấu hình framerate của trò chơi. Tìm và gắn 2 thể hiện quan trọng là `player` và `spawner`. Lưu ý, ta sẽ tạm dừng trò chơi. Chính người chơi sẽ kích hoạt trò chơi khi cần.

- `Play` Cho điểm số về 0, hiển thị lên màn hình. Ẩn hình "GameOver" và nút "PlayButton" đi. Thiết lập thông số thời gian và kích hoạt player. Tìm kiếm và xóa bỏ toàn bộ các ống cống `Pipes`.
- `Pause` Dừng toàn bộ tiến trình của trò chơi bằng cách dừng thời gian và bất hoạt player.
- `GameOver` Hiển thị hình "GameOver". Hiển thị nút "PlayButton" để người chơi có thể nhấp chuột và chơi lại. Gọi phương thức Pause để dừng toàn bộ tiến trình logic của trò chơi.
- `IncreaseScore` Tăng điểm số và xuất điểm số ra màn hình thông qua scoreText - tương ứng với cấu phần giao diện Text trên Canvas.

Listing 3.1: GameManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class GameManager : MonoBehaviour
{
    private Player player;
    private Spawner spawner;
    public Text scoreText;
    public GameObject playButton;
    public GameObject gameOver;
    private int score;

    private void Awake() {
        Application.targetFrameRate = 60;
        player = FindObjectOfType<Player>();
        spawner = FindObjectOfType<Spawner>();

        Pause();
    }
    public void Play() {
        score = 0;
        scoreText.text = score.ToString();
        gameOver.SetActive(false);
        playButton.SetActive(false);

```

```
28            Time.timeScale = 1f;
29            player.enabled = true;
30
31            Pipes[] pipes = FindObjectsOfType<Pipes>();
32            for(int i=0; i<pipes.Length; i++) {
33                Destroy(pipes[i].gameObject);
34            }
35        }
36        public void Pause() {
37            Time.timeScale = 0f;
38            player.enabled = false;
39        }
40        public void GameOver() {
41            gameOver.SetActive(true);
42            playButton.SetActive(true);
43            Pause();
44        }
45        public void IncreaseScore() {
46            score++;
47            scoreText.text = score.ToString();
48        }
49    }
```

Ground.cs

- `Awake`: Tìm và gắn cấu phần `Material` tương ứng.
- `Update` Tính toán thông số đồ hộ `Material` tương ứng để xuất ra khung hình (frame).

Listing 3.2: Ground.cs

```
1
2   using System.Collections;
3   using System.Collections.Generic;
4   using UnityEngine;
5
6   public class Ground : MonoBehaviour {
7     private MeshRenderer meshRenderer;
8     public float animationSpeed = 1f;
9
10    private void Awake() {
11      meshRenderer = GetComponent<MeshRenderer>();
```

```
    }

    void Update() {
      Vector2 deltaOffset = new Vector2(
        animationSpeed + Time.deltaTime, 0);

      meshRenderer.material.mainTextureOffset +=
        deltaOffset;
    }
  }
```

Parallax.cs

- `Awake`: Tìm và gắn cấu phần `Material` tương ứng.
- `Update` Tính toán thông số đồ hộ `Material` tương ứng để xuất ra khung hình (frame).

Listing 3.3: Parallax.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Parallax : MonoBehaviour {
  private MeshRenderer meshRenderer;
  public float animationSpeed = 1f;

  private void Awake() {
    meshRenderer = GetComponent<MeshRenderer>();
  }

  private void Update() {
    meshRenderer.material.mainTextureOffset += newVector2(
   animationSpeed * Time.deltaTime, 0);
  }
}
```

Pipes.cs

- `Start`: Ngay khi tro chơi bắt đầu, ta sẽ lấy thông số độ rộng màn hình, tìm mép trái `leftEnge` để làm căn cứ so sánh và hủy các cặp `Pipes` sau này.

- `Update` Tính toán thông số vị trí của `Pipes` tương ứng để xuất ra từng khung hình (frame). Nếu `Pipes` đã ra khỏi mép trái khung hình thì xóa bỏ để tiết kiệm bộ nhớ.

Listing 3.4: Pipes.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Pipes : MonoBehaviour {
  public float speed = 5f;
  private float leftEdge;
  private void Start() {
    leftEdge = Camera.
                main.
                ScreenToWorldPoint(Vector3.zero).x - 1f;
  }
  private void Update() {
    transform.position += Vector3.left *
                          speed * Time.deltaTime;

    if (transform.position.x < leftEdge) {
          Destroy(gameObject);
      }
  }
}
```

Player.cs

- `Awake`: Tìm và gắn cấu phần `Sprite` tương ứng.
- `AnimateSprite`: Theo dõi và xuất ra hình kết tiếp trong chuỗi `Bird_01.png`, `Bird_02.png`, `Bird_03.png`.
- `Start`: Khởi động tiến trình hiển thị chuỗi hình liên tiếp thông qua `AnimateSprite` để tạo chuyển động vỗ cánh cho chú chim `Bird`.
- `OnEnable`: Thiết lập lại vị trí xuất phát và vận tốc khởi động của chú chim.
- `Update`: Kiểm tra xem có sự kiện nhấp chuột hoặc gõ phím khoảng trắng (space) nào trong khung hình trước hay không. Nếu có thì sẽ thay đổi tốc

độ tức thời của chú chim. Dĩ nhiên, tốc độ tức thời mới này sẽ được tính toán lại dựa trên gia tốc trọng trường nữa. Cuối cùng, dựa vào tốc độ tức thời để tính ra tọa độ tức thời mới ứng với mỗi khung hình.

- `OnTriggerEnter2D` Kiểm tra chồng lấn. Nếu `Circle Collider 2D` của chú chim chồng lấn lên `Box Collider 2D` của `Ground` hay `Pipes` thì kết thúc trò chơi. Còn nếu chú chim chồng lấn lên `Box Collider 2D` của khoảng trống giữa cặp `Pipes` thì tích lũy thêm điểm.

Listing 3.5: Player.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Player : MonoBehaviour {
  private SpriteRenderer spriteRenderer;
  public Sprite[] sprites;
  private int spriteIndex;
  private Vector3 direction;
  public float gravity = -9.8f;
  public float strength = 5f;

  public void Awake() {
    spriteRenderer = GetComponent<SpriteRenderer>();
  }

  private void Start() {
    InvokeRepeating(nameof(AnimateSprite), 0.15f, 0.15f);
  }

  private void OnEnable()
  {
    Vector3 position = transform.position;
    position.y = 0f;
    transform.position = position;
    direction = Vector3.zero;
  }

  public void Update() {
    if (Input.GetKeyDown(KeyCode.Space) || Input.
 GetMouseButtonDown(0)) {
      direction = Vector3.up * strength;
```

```
32        }
33
34        direction.y += gravity * Time.deltaTime;
35        transform.position += direction * Time.deltaTime;
36      }
37
38      private void AnimateSprite() {
39        spriteIndex++;
40        if (spriteIndex >= sprites.Length) {
41          spriteIndex = 0;
42        }
43        spriteRenderer.sprite = sprites[spriteIndex];
44      }
45
46      private void OnTriggerEnter2D(Collider2D other) {
47        switch (other.gameObject.tag) {
48        case "Obstacle":
49          FindObjectOfType<GameManager>().GameOver();
50          return;
51        case "Scoring":
52          FindObjectOfType<GameManager>().IncreaseScore();
53          return;
54        }
55      }
56    }
```

Spawner.cs

- `OnEnable`: Kích hoạt quá trình sinh ngẫu nhiên các cặp ống cống `Pipes` theo thời gian. Ở đây, tốc độ sinh cặp ống cống sẽ phải tương thích với tốc độ dịch chuyển sang trái của các cặp ống cống đang có trên `Scene`.
- `OnDisable`: Ngừng quá trình sinh ngẫu nhiên các cặp ống cống `Pipes`.
- `Spawn`: Tạo thể hiện mới của lớp đối tượng `Pipes`, và thiết lập vị trí khởi đầu cho cặp ống cống `Pipes` đó. Ở đây, ta sẽ cho cặp này nằm phía ngoài biên phải của `Scene`.

Listing 3.6: Spawner.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
```

```
using UnityEngine;

public class Spawner : MonoBehaviour {
  public GameObject prefab;
  public float spawnRate = 1f;
  public float minHeight = -1f;
  public float maxHeight = 1f;

  private void OnEnable() {
    InvokeRepeating(nameof(Spawn), spawnRate, spawnRate);
  }

  private void OnDisable() {
    CancelInvoke(nameof(Spawn));
  }

  private void Spawn() {
    GameObject pipes = Instantiate(prefab, transform.position,
 Quaternion.identity);
    pipes.transform.position += Vector3.up * Random.Range(
 minHeight, maxHeight);
  }
}
```

Chú giải thuật ngữ

chồng lấn (collision). 8, 13, 18

cấu phần vật lí (Physics component). ii, 8

cấu phần giao diện (UI component). ii, 5, 7, 8, 9

cài đặt (setting). 7, 13

khoảng trắng (space). 1, 17

khung hình (frame). 11, 13, 15, 16, 17, 18

lớp đối tượng (class). 11, 12, 19

mốc sự kiện (hook). ii, 12, 13

phương thức (method). 13

thể hiện (instance). 11, 13, 19

trình-viết-trò-chơi (game engine). 2

xuyên nền tảng (cross-platform). 2

Tham khảo

[Uni22a] Unity. Executionorder, 2022. https://docs.unity3d.com/Manual/ExecutionOrder.html. 11

[Uni22b] Unity. Ml-agents, 2022. https://unity.com/products/machine-learning-agents. 3