**Mục lục**

Chương 1 Tổng quan về Unity

* 1. Giới thiệu về Unity
  2. Tính năng chính
  3. Unity trong phát triển game
  4. Các ứng dụng khác của Unity

Chương 2 Một số vấn đề và giải pháp khi xây dựng game engine Unity

2.1 Quản lý hiệu suất

2.1.1 Tối ưu hóa mã nguồn

2.1.2 Kỹ thuật culling và LOD

2.2 Tối ưu hóa đồ họa

2.2.1 Sử dụng shader

2.2.2 Texture compression

2.2.3 Sử dụng batching

2.3 Quản lý bộ nhớ

2.3.1 Giải phóng bộ nhớ không cần thiết

2.3.2 Quản lý đối tượng game hợp lý

2.4 Đảm bảo tính tương thích

2.4.1 Công cụ kiểm thử của Unity

2.4.2 Debug trên nhiều nền tảng

Chương 3 Ứng dụng game phát triển trên Unity

3.1 Giới thiệu về Flappy Bird

3.1.1 Lịch sử và sự phát triển

3.1.2 Đặc điểm nổi bật

3.2 Các bước phát triển Flappy Bird trong Unity

3.2.1 Thiết lập dự án

3.2.2 Tạo nhân vật chính

3.2.3 Lập trình điều khiển

**Chương 1 Tổng quan về Unity**

#### 1.1 Giới thiệu về Unity

Unity là một công cụ phát triển phần mềm đa nền tảng, được sử dụng chủ yếu để phát triển trò chơi điện tử và các ứng dụng tương tác 3D/2D. Được ra mắt lần đầu vào năm 2005 bởi Unity Technologies, Unity nhanh chóng trở thành một trong những công cụ phát triển game phổ biến nhất trên thế giới. Điểm mạnh của Unity nằm ở khả năng hỗ trợ đa nền tảng, giao diện người dùng thân thiện, và một cộng đồng phát triển rộng lớn và năng động.

**1.2 Tính năng chính của Unity**

Unity cung cấp nhiều tính năng mạnh mẽ giúp các nhà phát triển dễ dàng tạo ra các sản phẩm chất lượng cao:

* **Đa nền tảng**: Unity cho phép xuất bản trò chơi và ứng dụng trên nhiều nền tảng khác nhau như Windows, MacOS, iOS, Android, PlayStation, Xbox, và thậm chí là trên các trình duyệt web thông qua plugin Unity Web Player. Tính năng này giúp các nhà phát triển tiết kiệm thời gian và công sức khi chỉ cần phát triển một lần nhưng có thể triển khai trên nhiều nền tảng.
* **Môi trường làm việc trực quan**: Unity cung cấp một giao diện đồ họa dễ sử dụng, với khả năng kéo và thả các đối tượng vào cảnh. Giao diện này giúp giảm bớt khó khăn cho những người mới bắt đầu và tăng tốc quá trình phát triển cho các chuyên gia.
* **Hỗ trợ VR/AR**: Unity có khả năng hỗ trợ phát triển các ứng dụng thực tế ảo (VR) và thực tế tăng cường (AR), mở ra nhiều cơ hội sáng tạo trong các lĩnh vực từ giải trí đến giáo dục và y tế.
* **Lập trình bằng C#**: Unity hỗ trợ ngôn ngữ lập trình C#, một ngôn ngữ mạnh mẽ và dễ học. Các nhà phát triển có thể sử dụng Visual Studio, một môi trường phát triển tích hợp (IDE) mạnh mẽ, để viết và debug mã của họ.
* **Asset Store**: Unity Asset Store là một kho tài nguyên trực tuyến, nơi các nhà phát triển có thể mua hoặc tải về miễn phí hàng ngàn tài sản, bao gồm mô hình 3D, âm thanh, script, và các công cụ. Điều này giúp tiết kiệm thời gian phát triển và tăng cường khả năng sáng tạo.

**1.3 Unity trong phát triển game**

Unity được sử dụng rộng rãi trong ngành công nghiệp game nhờ vào khả năng tạo ra các trò chơi 3D/2D chất lượng cao, hỗ trợ đa nền tảng và cộng đồng phát triển đông đảo. Nhiều tựa game nổi tiếng đã được phát triển bằng Unity, bao gồm "Hearthstone" của Blizzard Entertainment, "Monument Valley" của Ustwo Games, và "Pokémon Go" của Niantic. Unity cung cấp nhiều công cụ và tính năng giúp tối ưu hóa quy trình phát triển game, từ thiết kế đồ họa, lập trình, đến kiểm thử và triển khai.

**1.4 Các ứng dụng khác của Unity**

Ngoài việc phát triển game, Unity còn được sử dụng trong nhiều lĩnh vực khác nhau nhờ vào khả năng tạo ra các ứng dụng tương tác mạnh mẽ:

* **Thực tế ảo và tăng cường**: Unity là một công cụ mạnh mẽ cho việc phát triển các ứng dụng VR và AR. Các ứng dụng này được sử dụng rộng rãi trong giáo dục, y tế, bất động sản, và các lĩnh vực khác để tạo ra những trải nghiệm phong phú và tương tác.
* **Mô phỏng và trực quan hóa**: Unity được sử dụng để phát triển các ứng dụng mô phỏng và trực quan hóa dữ liệu. Ví dụ, trong ngành công nghiệp ô tô, Unity được sử dụng để tạo ra các mô hình 3D của xe hơi và mô phỏng các điều kiện lái xe thực tế.
* **Kiến trúc và xây dựng**: Unity được sử dụng để tạo ra các mô hình 3D của các tòa nhà và dự án xây dựng. Các kiến trúc sư và kỹ sư có thể sử dụng các mô hình này để trình bày ý tưởng của họ cho khách hàng và nhà đầu tư một cách trực quan và sống động.
* **Giáo dục và đào tạo**: Unity được sử dụng để phát triển các ứng dụng giáo dục và đào tạo. Ví dụ, các trường học và cơ sở giáo dục có thể sử dụng Unity để tạo ra các bài học tương tác và mô phỏng khoa học. Trong lĩnh vực y tế, Unity có thể được sử dụng để tạo ra các mô phỏng phẫu thuật và đào tạo kỹ thuật viên y tế.
* **Phim ảnh và hoạt hình**: Unity cũng được sử dụng trong ngành công nghiệp phim ảnh và hoạt hình để tạo ra các hiệu ứng đặc biệt và cảnh quay 3D. Unity cho phép các nhà làm phim tạo ra những cảnh quay phức tạp với độ chân thực cao mà không cần tốn kém nhiều thời gian và chi phí.

### Chương 2: Một số vấn đề và giải pháp khi xây dựng game engine Unity

#### 2.1 Quản lý hiệu suất

Quản lý hiệu suất là một trong những thách thức lớn nhất khi phát triển trò chơi. Hiệu suất kém có thể làm giảm trải nghiệm của người chơi và ảnh hưởng đến doanh thu của game. Dưới đây là một số vấn đề phổ biến và các giải pháp để tối ưu hóa hiệu suất trong Unity.

##### **2.1.1 Tối ưu hóa mã nguồn**

Mã nguồn không tối ưu có thể dẫn đến hiệu suất kém và gây ra lag hoặc giảm tốc độ khung hình. Để tối ưu hóa mã nguồn:

* **Sử dụng các thuật toán hiệu quả**: Đảm bảo rằng các thuật toán được sử dụng là hiệu quả nhất có thể. Tránh các thuật toán có độ phức tạp cao trong các vòng lặp hoặc phần mã được gọi thường xuyên.
* **Giảm thiểu số lượng hàm gọi**: Các hàm gọi thường xuyên có thể tiêu tốn nhiều tài nguyên CPU. Sử dụng các kỹ thuật như caching để lưu trữ kết quả của các hàm gọi tốn kém.
* **Tối ưu hóa các vòng lặp**: Tránh việc lặp lại các tác vụ không cần thiết bên trong các vòng lặp. Đảm bảo rằng mỗi vòng lặp chỉ thực hiện công việc cần thiết.

##### **2.1.2 Kỹ thuật culling và LOD**

* **Culling**: Loại bỏ các đối tượng không cần thiết trong khung nhìn để tiết kiệm tài nguyên. Unity hỗ trợ các kỹ thuật culling như Frustum Culling và Occlusion Culling để tối ưu hóa việc render.
* **LOD (Level of Detail)**: Giảm độ chi tiết của các đối tượng khi chúng ở xa camera. Unity cho phép thiết lập nhiều mức độ chi tiết khác nhau cho các đối tượng, giúp giảm tải cho GPU khi render các cảnh phức tạp.

#### 2.2 Tối ưu hóa đồ họa

Đồ họa chất lượng cao có thể tiêu tốn nhiều tài nguyên, do đó cần có các biện pháp tối ưu hóa để đảm bảo trò chơi chạy mượt mà.

##### **2.2.1 Sử dụng shader**

* **Shader tối ưu**: Sử dụng các shader tối ưu để giảm tải cho GPU. Unity cung cấp nhiều loại shader khác nhau và cho phép tùy chỉnh shader để phù hợp với yêu cầu cụ thể của trò chơi.
* **Shader LOD**: Sử dụng shader LOD để giảm độ phức tạp của shader khi các đối tượng ở xa camera, giúp tiết kiệm tài nguyên GPU.

##### **2.2.2 Texture compression**

* **Nén texture**: Sử dụng các kỹ thuật nén texture để giảm dung lượng và tăng hiệu suất. Unity hỗ trợ nhiều định dạng nén texture như DXT, PVR và ETC.
* **Mipmaps**: Sử dụng mipmaps để giảm thiểu aliasing và tăng hiệu suất khi render các texture ở xa camera.

##### **2.2.3 Sử dụng batching**

* **Static batching**: Áp dụng static batching cho các đối tượng không di chuyển trong suốt quá trình chơi game. Static batching giúp gộp nhiều đối tượng lại thành một mesh duy nhất, giảm số lượng draw calls.
* **Dynamic batching**: Áp dụng dynamic batching cho các đối tượng di chuyển trong game. Dynamic batching giúp gộp các đối tượng nhỏ lại với nhau trong mỗi frame.

#### 2.3 Quản lý bộ nhớ

Quản lý bộ nhớ hiệu quả là cần thiết để tránh các vấn đề như tràn bộ nhớ, lag và crash.

##### **2.3.1 Giải phóng bộ nhớ không cần thiết**

* **Quản lý đối tượng**: Giải phóng bộ nhớ của các đối tượng không cần thiết bằng cách sử dụng các phương pháp như Destroy() hoặc Object Pooling.
* **Garbage collection**: Tối ưu hóa việc sử dụng bộ nhớ để giảm thiểu garbage collection. Tránh tạo ra quá nhiều đối tượng tạm thời và sử dụng lại các đối tượng đã tồn tại.

##### **2.3.2 Quản lý đối tượng game hợp lý**

* **Object Pooling**: Sử dụng kỹ thuật Object Pooling để quản lý các đối tượng game, đặc biệt là các đối tượng được tạo và hủy liên tục như đạn hoặc hiệu ứng đặc biệt. Điều này giúp giảm thiểu việc tạo và hủy đối tượng liên tục, cải thiện hiệu suất.
* **Texture và tài nguyên**: Tải các texture và tài nguyên cần thiết trước khi vào game và giải phóng chúng khi không còn sử dụng.

#### 2.4 Đảm bảo tính tương thích

Đảm bảo rằng trò chơi hoạt động tốt trên nhiều nền tảng và thiết bị khác nhau là một yếu tố quan trọng để đạt được sự thành công rộng rãi.

##### **2.4.1 Công cụ kiểm thử của Unity**

* **Profiler**: Sử dụng Profiler của Unity để phân tích hiệu suất và xác định các vấn đề tiềm ẩn trong trò chơi. Profiler cung cấp các thông tin chi tiết về CPU, GPU, bộ nhớ và các tài nguyên khác.
* **Frame Debugger**: Sử dụng Frame Debugger để kiểm tra quá trình render từng frame, giúp xác định các vấn đề liên quan đến đồ họa và draw calls.

##### **2.4.2 Debug trên nhiều nền tảng**

* **Cross-platform testing**: Kiểm tra và debug trò chơi trên nhiều nền tảng và thiết bị khác nhau để đảm bảo tính tương thích và hiệu suất. Sử dụng các thiết bị thực tế cũng như các trình giả lập và công cụ kiểm thử để đánh giá hiệu suất và trải nghiệm người dùng.
* **Build settings**: Điều chỉnh các thiết lập build cho từng nền tảng để tối ưu hóa hiệu suất và đảm bảo rằng trò chơi hoạt động mượt mà trên tất cả các nền tảng được hỗ trợ.

### **Chương 3: Ứng dụng game phát triển trên Unity**

#### 3.1 Giới thiệu về Flappy Bird

##### **3.1.1 Lịch sử và sự phát triển**

Flappy Bird là một tựa game di động nổi tiếng, được phát triển bởi Nguyễn Hà Đông, một lập trình viên người Việt Nam. Trò chơi được phát hành lần đầu tiên vào tháng 5 năm 2013, nhưng chỉ thực sự nổi tiếng vào đầu năm 2014. Flappy Bird nhanh chóng trở thành một hiện tượng toàn cầu với hàng triệu lượt tải về trên cả hai nền tảng iOS và Android. Trò chơi gây ấn tượng bởi lối chơi đơn giản nhưng lại cực kỳ thử thách, yêu cầu người chơi phải giữ cho chú chim không va chạm vào các ống cản.

##### **3.1.2 Đặc điểm nổi bật**

Flappy Bird có một số đặc điểm nổi bật sau:

* **Lối chơi đơn giản**: Người chơi chỉ cần chạm vào màn hình để điều khiển chú chim bay lên và tránh các ống cản. Mỗi lần chạm vào màn hình, chú chim sẽ bay lên một chút và sau đó rơi xuống do trọng lực.
* **Độ khó cao**: Mặc dù cơ chế chơi đơn giản, nhưng độ khó của trò chơi rất cao, yêu cầu người chơi phải có phản xạ tốt và khả năng tập trung cao độ.
* **Đồ họa pixel**: Trò chơi sử dụng đồ họa pixel đơn giản nhưng cuốn hút, gợi nhớ về các tựa game cổ điển.
* **Tính gây nghiện**: Lối chơi đơn giản nhưng thử thách khiến người chơi muốn chơi đi chơi lại để cải thiện thành tích của mình.

#### 3.2 Các bước phát triển Flappy Bird trong Unity

##### **3.2.1 Thiết lập dự án**

###### **3.2.1.1 Tạo dự án mới**

Để bắt đầu, bạn cần mở Unity và tạo một dự án mới. Đây là bước quan trọng để thiết lập các tham số cơ bản và chuẩn bị môi trường làm việc.

1. Mở Unity và chọn "New Project".
2. Đặt tên cho dự án và chọn vị trí lưu trữ.
3. Chọn loại dự án là 2D.
4. Nhấn "Create" để tạo dự án mới.

###### **3.2.1.2 Cài đặt các tham số cơ bản**

Sau khi tạo dự án mới, bạn cần cài đặt các tham số cơ bản để đảm bảo dự án hoạt động mượt mà:

1. Chuyển đến "Edit" > "Project Settings".
2. Trong "Player", cài đặt các thông số như tên ứng dụng, biểu tượng và các tùy chọn khác.
3. Đảm bảo rằng chế độ nền tảng phù hợp đã được chọn (ví dụ: iOS, Android, Windows).

##### **3.2.2 Tạo nhân vật chính**

###### **3.2.2.1 Thiết kế nhân vật chim**

Nhân vật chính trong Flappy Bird là một chú chim nhỏ. Bạn có thể sử dụng các công cụ đồ họa như Photoshop hoặc Illustrator để tạo hình ảnh cho nhân vật chim, sau đó nhập vào Unity.

1. Tạo một sprite mới cho nhân vật chim.
2. Kéo và thả sprite vào cửa sổ Scene.
3. Sử dụng Sprite Renderer để hiển thị hình ảnh của nhân vật chim.

###### **3.2.2.2 Thiết kế chướng ngại vật**

Chướng ngại vật trong Flappy Bird là các ống cản mà người chơi phải tránh. Bạn cũng có thể thiết kế chúng bằng các công cụ đồ họa và nhập vào Unity.

1. Tạo một sprite mới cho các ống cản.
2. Kéo và thả sprite vào cửa sổ Scene.
3. Sử dụng Sprite Renderer để hiển thị hình ảnh của các ống cản.

##### **3.2.3 Lập trình điều khiển**

###### **Viết mã C# cho chuyển động**

Để có thể chuyển động ta phải viết đoạn mã C# cho chương trình

 Tạo một script C# mới và đặt tên là " **Player** ".

 Gán script này cho nhân vật chim trong cửa sổ Scene.

 Mở script và thêm mã sau để điều khiển chuyển động của chim:

using UnityEngine;

public class Player : MonoBehaviour

{

    public Sprite[] sprites;

    public float strength = 5f;

    public float gravity = -9.81f;

    public float tilt = 5f;

    private SpriteRenderer spriteRenderer;

    private Vector3 direction;

    private int spriteIndex;

    private 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;

    }

    private void Update()

    {

        if (Input.GetKeyDown(KeyCode.Space) || Input.GetMouseButtonDown(0)) {

            direction = Vector3.up \* strength;

        }

        // Apply gravity and update the position

        direction.y += gravity \* Time.deltaTime;

        transform.position += direction \* Time.deltaTime;

        // Tilt the bird based on the direction

        Vector3 rotation = transform.eulerAngles;

        rotation.z = direction.y \* tilt;

        transform.eulerAngles = rotation;

    }

    private void AnimateSprite()

    {

        spriteIndex++;

        if (spriteIndex >= sprites.Length) {

            spriteIndex = 0;

        }

        if (spriteIndex < sprites.Length && spriteIndex >= 0) {

            spriteRenderer.sprite = sprites[spriteIndex];

        }

    }

    private void OnTriggerEnter2D(Collider2D other)

    {

        if (other.gameObject.CompareTag("Obstacle")) {

            GameManager.Instance.GameOver();

        } else if (other.gameObject.CompareTag("Scoring")) {

            GameManager.Instance.IncreaseScore();

        }

    }

}

Tiếp đến cần thêm chuyển động hình nền cho chương trình sử dụng đoạn mã có tên là **Parallax:**

using UnityEngine;

public class Parallax : MonoBehaviour

{

    public float animationSpeed = 1f;

    private MeshRenderer meshRenderer;

    private void Awake()

    {

        meshRenderer = GetComponent<MeshRenderer>();

    }

    private void Update()

    {

        meshRenderer.material.mainTextureOffset += new Vector2(animationSpeed \* Time.deltaTime, 0);

    }

}

**Tiếp đến là đoạn mã đẻ spawn ra cac chướng ngại vật có tên là Pipes và Spawner:**

**Thứ nhất là với Pipes:**

using UnityEngine;

public class Pipes : MonoBehaviour

{

    public Transform top;

    public Transform bottom;

    public float speed = 5f;

    public float gap = 3f;

    private float leftEdge;

    private void Start()

    {

        leftEdge = Camera.main.ScreenToWorldPoint(Vector3.zero).x - 1f;

        top.position += Vector3.up \* gap / 2;

        bottom.position += Vector3.down \* gap / 2;

    }

    private void Update()

    {

        transform.position += speed \* Time.deltaTime \* Vector3.left;

        if (transform.position.x < leftEdge) {

            Destroy(gameObject);

        }

    }

}

**Tiếp đó là Spawner:**

using UnityEngine;

public class Spawner : MonoBehaviour

{

    public Pipes prefab;

    public float spawnRate = 1f;

    public float minHeight = -1f;

    public float maxHeight = 2f;

    public float verticalGap = 3f;

    private void OnEnable()

    {

        InvokeRepeating(nameof(Spawn), spawnRate, spawnRate);

    }

    private void OnDisable()

    {

        CancelInvoke(nameof(Spawn));

    }

    private void Spawn()

    {

        Pipes pipes = Instantiate(prefab, transform.position, Quaternion.identity);

        pipes.transform.position += Vector3.up \* Random.Range(minHeight, maxHeight);

        pipes.gap = verticalGap;

    }

}

**Và để chương trình chạy ổn định cần thêm đoạn mã có tên là GameManager:**

using UnityEngine;

using UnityEngine.UI;

[DefaultExecutionOrder(-1)]

public class GameManager : MonoBehaviour

{

    public static GameManager Instance { get; private set; }

    [SerializeField] private Player player;

    [SerializeField] private Spawner spawner;

    [SerializeField] private Text scoreText;

    [SerializeField] private GameObject playButton;

    [SerializeField] private GameObject gameOver;

    public int score { get; private set; } = 0;

    private void Awake()

    {

        if (Instance != null) {

            DestroyImmediate(gameObject);

        } else {

            Instance = this;

        }

    }

    private void OnDestroy()

    {

        if (Instance == this) {

            Instance = null;

        }

    }

    private void Start()

    {

        Pause();

    }

    public void Pause()

    {

        Time.timeScale = 0f;

        player.enabled = false;

    }

    public void Play()

    {

        score = 0;

        scoreText.text = score.ToString();

        playButton.SetActive(false);

        gameOver.SetActive(false);

        Time.timeScale = 1f;

        player.enabled = true;

        Pipes[] pipes = FindObjectsOfType<Pipes>();

        for (int i = 0; i < pipes.Length; i++) {

            Destroy(pipes[i].gameObject);

        }

    }

    public void GameOver()

    {

        playButton.SetActive(true);

        gameOver.SetActive(true);

        Pause();

    }

    public void IncreaseScore()

    {

        score++;

        scoreText.text = score.ToString();

    }

}

### **Kết luận**

Trong chương này, chúng ta đã tìm hiểu về lịch sử và sự phát triển của Flappy Bird, một tựa game nổi tiếng được phát triển bằng Unity. Chúng ta cũng đã khám phá các bước cơ bản để phát triển trò chơi này trong Unity, từ thiết lập dự án, tạo nhân vật chính, đến lập trình điều khiển và tương tác với người chơi. Với Unity, việc phát triển một trò chơi chất lượng cao trở nên dễ dàng hơn bao giờ hết, mở ra nhiều cơ hội sáng tạo cho các nhà phát triển game.

**Tài liệu tham khảo**

**Unity Learn**: learn.unity.com - Trang web chính thức của Unity cung cấp nhiều khóa học, tutorial và hướng dẫn miễn phí về các chủ đề khác nhau liên quan đến Unity.

**GameDev.net**: gamedev.net - Cộng đồng phát triển game với nhiều bài viết, diễn đàn và tài liệu hữu ích về phát triển game.

YouTube: Zigurous

### **Chương 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN**

#### 5.1 Các kết quả đạt được

##### **5.1.1 Ứng dụng game**

Sau khi tìm hiểu các phương pháp xây dựng game với engine Unity, chúng em đã hoàn thành ứng dụng game Flappy Bird. Trong quá trình phát triển, chúng em đã gặp phải một số vấn đề nhất định và đã đưa ra các giải pháp nhằm giải quyết những vấn đề này. Nhờ đó, chúng em không chỉ nắm vững kỹ thuật phát triển game mà còn tích lũy được kinh nghiệm quý báu trong việc quản lý và tối ưu hóa dự án.

#### 5.2 Hướng phát triển

##### **5.2.1 Tối ưu hóa các xử lý giúp game chạy nhanh hơn**

Để nâng cao trải nghiệm người chơi, chúng em sẽ tiếp tục tối ưu hóa các xử lý trong game nhằm giảm thiểu thời gian tải và tăng cường tốc độ khung hình.

##### **5.2.2 Game sẽ hỗ trợ chế độ nhiều người chơi cùng một lúc qua Bluetooth hoặc Internet**

Việc phát triển chế độ nhiều người chơi qua Bluetooth hoặc Internet sẽ mở rộng khả năng tương tác và cạnh tranh giữa các người chơi, mang lại trải nghiệm phong phú và hấp dẫn hơn.