**Chương 1: Giới thiệu**

C# là một ngôn ngữ lập trình ra đời từ năm 2001. Đây là một trong các ngôn ngữ lập trình được tích hợp trong .NET của Microsoft và có thể được sử dụng để phát triển nhiều loại chương trình khác nhau, bao gồm cả trò chơi. Tại sao chúng tôi chọn C# để dạy trong cuốn sách này? Vì hai lý do quan trọng:

1. Đây là một ngôn ngữ mạnh mẽ, hiện đại và tương đối dễ học cho người mới bắt đầu.
2. Quan trọng hơn, C# là ngôn ngữ ưa thích để viết script trong Unity, công cụ tạo game phổ biến. Vì các script là thành phần thực hiện gameplay trong game, nên việc biết cách viết chúng là rất quan trọng khi bạn phát triển game của riêng mình.

**1.1. Tổ chức phần cứng**

Trước khi nói về cách phần cứng máy tính được tổ chức, phần cứng là gì? Cách dễ nhất để nghĩ về nó là: phần cứng sẽ làm bạn đau nếu bạn thả nó rơi vào chân! Nói cách khác, phần cứng là các bộ phận của máy tính mà bạn có thể nhìn và chạm vào. Phần lớn máy tính, đặc biệt là PC, bao gồm 4 loại phần cứng khác nhau: **CPU (Bộ xử lý trung tâm), bộ nhớ (RAM và lưu trữ), thiết bị nhập và thiết bị xuất**.

**CPU (Central Processing Unit)**

CPU là “bộ não” của máy tính. Máy tính thực tế ngu hơn bạn nghĩ – nó chỉ có thể thực hiện các lệnh, tuần tự, cho đến khi bạn tắt nó! CPU nhận các lệnh từ **bộ nhớ chính (RAM)** để thực hiện. Bộ nhớ này có hai loại:

* **ROM (Read-Only Memory):** Chỉ lưu trữ thông tin cố định, không thay đổi.
* **RAM (Random Access Memory):** Lưu thông tin tạm thời và bị mất khi tắt máy.

**Thiết bị nhập và xuất**

Thiết bị nhập, như bàn phím, chuột, hoặc tay cầm chơi game, cho phép bạn tương tác với máy tính. Trong khi đó, thiết bị xuất, như màn hình hoặc loa, hiển thị thông tin từ máy tính.

Phần cứng được kết nối với nhau thông qua **bus**, một hệ thống giao tiếp chung giúp các thành phần "nói chuyện" với nhau.

**1.2. Phần mềm máy tính**

Phần mềm máy tính là tập hợp các lệnh mà máy tính thực thi. Các ngôn ngữ lập trình hiện đại như C# giúp lập trình viên viết mã dễ dàng hơn, thay vì phải viết mã nhị phân (1 và 0) như trước đây.  
C# là ngôn ngữ kết hợp giữa **biên dịch (compiled)** và **diễn dịch (interpreted)**, mang lại tính linh hoạt và hiệu suất cao.

**1.3. Viết và chạy chương trình C#**

Để viết chương trình C#, bạn cần một **môi trường phát triển tích hợp (IDE)** như Visual Studio. Các bước cơ bản bao gồm:

1. **Viết mã:** Dùng IDE để nhập mã nguồn.
2. **Biên dịch mã:** Chuyển mã nguồn thành lệnh máy tính có thể hiểu.
3. **Chạy chương trình:** Kiểm tra kết quả chương trình trên màn hình.

**Chương 2: Bắt đầu viết mã**

Trong chương trước, bạn đã nhập, biên dịch, và chạy ứng dụng console đầu tiên bằng C#. Điều này giúp bạn hiểu quy trình trước khi chúng ta đi sâu vào việc sử dụng C# để giải quyết các vấn đề. Bây giờ, hãy xem xét kỹ hơn về cách mà các chương trình C# hoạt động.

**2.1. Một chương trình C# trông như thế nào?**

Câu hỏi này thực sự phức tạp hơn bạn nghĩ! Vì C# là một ngôn ngữ hướng đối tượng, chúng ta sẽ viết rất nhiều thứ được gọi là **lớp (classes)**. Các lớp này là khối xây dựng chính để giải quyết vấn đề. Mặc dù các chương trình của chúng ta sẽ sử dụng nhiều lớp, chúng ta cũng cần một tệp .cs chính để chạy chương trình. Tệp này thường được gọi là **lớp ứng dụng (application class)** vì nó điều khiển cách chương trình chạy.

Dưới đây là cú pháp cơ bản của một lớp ứng dụng C#:

using directives for namespaces namespace NamespaceName { class documentation comment internal class Program { method documentation comment static void Main(string[] args) { constant declarations variable declarations executable statements } } }

Khi thấy các từ **in nghiêng** (như *class documentation comment*), điều đó có nghĩa bạn sẽ tự quyết định nội dung. Các từ không in nghiêng (như namespace, class, static, và void) là các từ khóa C# và phải được viết chính xác như vậy.

**2.2. Sử dụng namespace và lớp khác**

Dòng đầu tiên trong lớp ứng dụng là nơi chúng ta khai báo sử dụng các namespace và lớp khác. **Namespace** là các tập hợp mã C# hữu ích mà người khác đã viết, thử nghiệm và kiểm tra lỗi (để bạn không phải làm điều đó). Ví dụ, để sử dụng lớp tạo số ngẫu nhiên Random từ namespace System, bạn cần thêm dòng:

using System;

Điều này báo cho trình biên dịch rằng bạn muốn sử dụng các lớp trong namespace System. Các IDE như Visual Studio đôi khi tự động thêm các namespace cần thiết.

**2.3. Namespace**

Namespace trong C# giúp nhóm các đoạn mã liên quan với nhau. Hầu hết mã bạn viết trong cuốn sách này sẽ nằm trong một namespace duy nhất. IDE tự động tạo phần namespace này dựa trên tên bạn chọn khi tạo dự án.

**2.4. Chú thích (Comments)**

Chú thích không bắt buộc trong C#, nhưng rất hữu ích để giải thích mã. Có ba loại chú thích thường được sử dụng:

1. **Chú thích tài liệu (documentation comments):** Bắt đầu bằng /// và thường chứa thông tin được định dạng theo XML, ví dụ:

*/// <summary>* */// In thông báo về mưa* */// </summary>*

Các công cụ như **Sandcastle** có thể chuyển chú thích này thành tài liệu HTML.

1. **Chú thích dòng (line comments):** Bắt đầu bằng //. Thường dùng để giải thích ngắn gọn một hoặc vài dòng mã.
2. **Chú thích khối (block comments):** Bao gồm nhiều dòng mã, bắt đầu bằng /\* và kết thúc bằng \*/.

**2.5. Lớp (Class)**

Một lớp trong C# được định nghĩa bằng từ khóa class. IDE tự động tô màu tên lớp để giúp bạn dễ phân biệt.

**2.6. Tên định danh (Identifiers)**

Tên định danh là tên bạn đặt cho các thành phần trong chương trình, như biến hoặc hàm. Một số quy tắc đặt tên định danh trong C#:

* Có thể chứa chữ cái, số, và dấu gạch dưới \_, nhưng không được bắt đầu bằng số.
* Không chứa khoảng trắng.
* Không được trùng với các từ khóa C#.

Ví dụ:

* **Hợp lệ:** Weapon, playerName, DelayTime
* **Không hợp lệ:** 2\_Name, int

**2.7. Phương thức Main**

Phương thức Main là điểm bắt đầu của mọi chương trình C#. Nó thường được định nghĩa như sau:

static void Main(string[] args) { *// Code thực thi* }

Câu lệnh string[] args cho phép bạn truyền các đối số dòng lệnh, nhưng chúng ta sẽ không sử dụng trong cuốn sách này.

**2.8. Biến và hằng số**

Bạn có thể khai báo biến và hằng số trong C# như sau:

const double C = 299792458; *// Hằng số* double energy; *// Biến* double mass;

**2.9. Xuất dữ liệu ra console**

Để xuất dữ liệu ra màn hình console, bạn sử dụng phương thức Console.WriteLine. Ví dụ:

Console.WriteLine("Xin chào thế giới!");

Phương thức này sẽ in chuỗi "Xin chào thế giới!" và xuống dòng. Nếu không muốn xuống dòng, bạn có thể sử dụng Console.Write.

**2.10. Dấu ngoặc nhọn (Curly Braces)**

C# sử dụng dấu ngoặc nhọn { } để nhóm các đoạn mã liên quan. Mỗi dấu mở { phải có một dấu đóng } tương ứng. Ví dụ:

class Program { static void Main(string[] args) { Console.WriteLine("Hello!"); } }

**Tóm tắt chương 2**

Chúng ta đã xem xét cấu trúc cơ bản của một chương trình C#. Bằng cách hiểu từng phần, bạn sẽ dễ dàng viết và bảo trì mã của mình hơn. Trong chương tiếp theo, chúng ta sẽ đi sâu vào các kiểu dữ liệu, biến, và hằng số trong C#.

**Chương 3: Kiểu dữ liệu, biến và hằng số**

Trong chương này, chúng ta sẽ tìm hiểu các **kiểu dữ liệu (data types)**, cách khai báo **biến (variables)**, và **hằng số (constants)** trong C#. Đây là những thành phần cơ bản để viết chương trình.

**3.1. Biến là gì?**

**Biến (variables)** là những "hộp lưu trữ" trong bộ nhớ mà bạn sử dụng để lưu trữ dữ liệu. Mỗi biến có một **tên (identifier)** và một **kiểu dữ liệu (data type)**. Kiểu dữ liệu xác định loại dữ liệu mà biến có thể lưu trữ, chẳng hạn như số nguyên, số thập phân, hoặc chuỗi ký tự.

Ví dụ, khai báo biến trong C#:

int score; *// Khai báo một biến kiểu số nguyên* double mass; *// Khai báo một biến kiểu số thực* string name; *// Khai báo một biến kiểu chuỗi*

Bạn cũng có thể gán giá trị ngay khi khai báo:

int score = 100; double mass = 72.5; string name = "Alice";

**3.2. Các loại kiểu dữ liệu**

C# hỗ trợ nhiều kiểu dữ liệu. Chúng được chia thành hai loại chính:

1. **Kiểu giá trị (Value Types):** Dữ liệu được lưu trữ trực tiếp trong vùng nhớ của biến.
2. **Kiểu tham chiếu (Reference Types):** Biến lưu trữ địa chỉ của dữ liệu thay vì chính dữ liệu.

**3.3. Số nguyên (Integer)**

* Sử dụng kiểu int cho các giá trị số nguyên.
* Kích thước: 4 byte, giá trị từ -2,147,483,648 đến 2,147,483,647.

Ví dụ:

int age = 25;

Ngoài int, C# còn hỗ trợ các kiểu khác như short, long, và byte với kích thước và phạm vi giá trị khác nhau.

**3.4. Số thực (Floating Point Numbers)**

* Sử dụng kiểu float hoặc double cho số thực.
  + float: Độ chính xác thấp hơn, kích thước 4 byte.
  + double: Độ chính xác cao hơn, kích thước 8 byte.

Ví dụ:

float height = 1.75f; *// Ký tự 'f' dùng để chỉ kiểu float* double distance = 123.456;

**3.5. Số thập phân chính xác cao (Decimal)**

* Sử dụng kiểu decimal cho số thập phân yêu cầu độ chính xác cao, ví dụ: số tiền.
* Kích thước: 16 byte.

Ví dụ:

decimal price = 199.99m; *// Ký tự 'm' dùng để chỉ kiểu decimal*

**3.6. Ký tự (Character)**

* Sử dụng kiểu char để lưu một ký tự.
* Giá trị được đặt trong dấu nháy đơn '.

Ví dụ:

char initial = 'A';

**3.7. Kiểu logic (Boolean)**

* Sử dụng kiểu bool để lưu giá trị logic: true hoặc false.

Ví dụ:

bool isGameOver = false;

**3.8. Biến và hằng số**

**Hằng số**

* Hằng số là giá trị không thay đổi trong suốt quá trình chạy chương trình.
* Sử dụng từ khóa const để khai báo.

Ví dụ:

const double Pi = 3.14159;

**Lựa chọn giữa biến và hằng số**

* Sử dụng biến khi giá trị có thể thay đổi.
* Sử dụng hằng số khi giá trị cố định, ví dụ: tốc độ ánh sáng.

**3.9. Gán giá trị cho biến**

Bạn có thể gán giá trị cho biến sau khi khai báo. Tuy nhiên, biến phải được gán giá trị trước khi sử dụng, nếu không chương trình sẽ báo lỗi.

Ví dụ:

int score; score = 10; *// Gán giá trị*

**3.10. Ép kiểu dữ liệu (Type Conversion)**

C# hỗ trợ hai loại ép kiểu:

1. **Ép kiểu ngầm định (Implicit):** Khi chuyển đổi an toàn (ví dụ: từ int sang double).
2. **Ép kiểu tường minh (Explicit):** Khi chuyển đổi không an toàn, cần dùng cú pháp (type).

Ví dụ:

int i = 10; double d = i; *// Ép kiểu ngầm định* int j = (int)d; *// Ép kiểu tường minh*

**3.11. Kết hợp biến trong ứng dụng**

Chúng ta sẽ áp dụng các khái niệm về biến và kiểu dữ liệu vào ứng dụng console đơn giản. Dưới đây là một ví dụ:

**Ví dụ: Tính diện tích hình tròn**

using System; class Program { static void Main() { const double Pi = 3.14159; double radius = 5.0; double area = Pi \* radius \* radius; Console.WriteLine($"Diện tích hình tròn: {area}"); } }

Kết quả:

bash

Diện tích hình tròn: 78.53975

**Tóm tắt chương 3**

* Bạn đã học cách khai báo và sử dụng biến, hằng số.
* Hiểu các kiểu dữ liệu cơ bản trong C# và cách gán giá trị.
* Biết cách ép kiểu dữ liệu.

# Chương 4: Lớp và đối tượng

C# là một ngôn ngữ lập trình hướng đối tượng, vì vậy việc hiểu về **lớp (class)** và **đối tượng (object)** là rất quan trọng. Trong chương này, chúng ta sẽ tìm hiểu cách tạo và sử dụng các lớp và đối tượng để giải quyết vấn đề.

**4.1. Giới thiệu về lớp và đối tượng**

**Lớp là gì?**

* + **Lớp (class)** là một mẫu thiết kế (blueprint) mô tả các thuộc tính (properties) và hành vi (methods) của một đối tượng.
  + Lớp không chiếm bộ nhớ cho đến khi bạn tạo ra các **đối tượng (objects)** từ nó.

Ví dụ về một lớp:

class Circle { public double Radius; *// Thuộc tính: bán kính* public double CalculateArea() *// Hành vi: tính diện tích* { return Math.PI \* Radius \* Radius; } }

**Đối tượng là gì?**

* + **Đối tượng (object)** là một thực thể cụ thể được tạo từ một lớp.
  + Bạn có thể tạo nhiều đối tượng từ một lớp và mỗi đối tượng sẽ có các giá trị thuộc tính riêng.

Ví dụ:

Circle c1 = new Circle(); c1.Radius = 5.0; Circle c2 = new Circle(); c2.Radius = 10.0; Console.WriteLine(c1.CalculateArea()); *// In diện tích của c1* Console.WriteLine(c2.CalculateArea()); *// In diện tích của c2*

**4.2. Viết chương trình C# đầu tiên với lớp**

Dưới đây là một ví dụ chương trình đơn giản để minh họa cách sử dụng lớp và đối tượng:

**Ví dụ: Tính diện tích hình tròn**

using System; class Circle { public double Radius; public double CalculateArea() { return Math.PI \* Radius \* Radius; } } class Program { static void Main() { Circle circle = new Circle(); circle.Radius = 7.0; Console.WriteLine($"Diện tích: {circle.CalculateArea()}"); } }

Kết quả:

Diện tích: 153.93804002589985

**4.3. Gọi phương thức (Calling Methods)**

Phương thức trong C# là một khối mã thực thi một tác vụ cụ thể. Để gọi một phương thức:

* + Tạo đối tượng từ lớp chứa phương thức.
  + Gọi phương thức thông qua đối tượng.

Ví dụ:

Circle c = new Circle(); c.Radius = 3.0; double area = c.CalculateArea(); *// Gọi phương thức tính diện tích* Console.WriteLine(area);

**4.4. Tham chiếu đối tượng trong bộ nhớ**

C# lưu trữ các đối tượng trong vùng **heap** và sử dụng con trỏ (reference) để truy cập chúng. Điều này có nghĩa là khi bạn gán một đối tượng cho một biến khác, cả hai biến đều trỏ đến cùng một đối tượng.

Ví dụ:

Circle c1 = new Circle(); c1.Radius = 5.0; Circle c2 = c1; *// c2 trỏ đến cùng đối tượng với c1* c2.Radius = 10.0; Console.WriteLine(c1.Radius); *// Kết quả: 10.0*

**4.5. Đưa tất cả lại với nhau trong ứng dụng console**

Dưới đây là một ví dụ hoàn chỉnh sử dụng các khái niệm đã học:

**Ví dụ: Tính diện tích và chu vi hình tròn**

using System; class Circle { public double Radius; public double CalculateArea() { return Math.PI \* Radius \* Radius; } public double CalculateCircumference() { return 2 \* Math.PI \* Radius; } } class Program { static void Main() { Circle circle = new Circle(); Console.Write("Nhập bán kính: "); circle.Radius = double.Parse(Console.ReadLine()); Console.WriteLine($"Diện tích: {circle.CalculateArea()}"); Console.WriteLine($"Chu vi: {circle.CalculateCircumference()}"); } }

Khi chạy chương trình:

css

Nhập bán kính: 4 Diện tích: 50.26548245743669 Chu vi: 25.132741228718345

**4.6. Đưa vào Unity**

Bạn có thể sử dụng các lớp và đối tượng tương tự trong Unity để xây dựng game. Ví dụ, bạn có thể viết script quản lý vị trí hoặc tốc độ của một đối tượng game.

**Ví dụ: Script Unity**

using UnityEngine; public class CircleBehavior : MonoBehaviour { public float Radius; void Start() { float area = Mathf.PI \* Radius \* Radius; Debug.Log($"Diện tích: {area}"); } }

Khi thêm script vào một GameObject trong Unity và đặt giá trị Radius, Unity sẽ tính diện tích khi bắt đầu game.

**Tóm tắt chương 4**

* + Lớp là mẫu thiết kế cho đối tượng, chứa thuộc tính và hành vi.
  + Đối tượng được tạo từ lớp, lưu trữ dữ liệu và thực hiện các phương thức.
  + Bạn có thể áp dụng các khái niệm về lớp và đối tượng cả trong C# console lẫn Unity.

# Chương 5: Chuỗi ký tự

Chuỗi ký tự (string) là một trong những kiểu dữ liệu cơ bản và rất quan trọng trong C#. Trong chương này, bạn sẽ học cách làm việc với chuỗi, từ việc khai báo và gán giá trị cho đến sử dụng các phương thức và lớp hỗ trợ để xử lý chuỗi.

**5.1. Lớp String**

Lớp String trong C# là một kiểu dữ liệu tham chiếu được dùng để lưu trữ và quản lý chuỗi ký tự. Mỗi chuỗi trong C# thực chất là một đối tượng của lớp String.

**Khai báo chuỗi**

Bạn có thể khai báo và gán giá trị cho chuỗi như sau:

string message = "Hello, world!";

**Phương thức thông dụng trong lớp String**

Một số phương thức phổ biến của lớp String:

1. **Length:** Lấy độ dài của chuỗi.

string name = "Alice"; Console.WriteLine(name.Length); *// Kết quả: 5*

1. **ToUpper() và ToLower():** Chuyển đổi chữ hoa và chữ thường.

string text = "Hello"; Console.WriteLine(text.ToUpper()); *// Kết quả: HELLO* Console.WriteLine(text.ToLower()); *// Kết quả: hello*

1. **Substring():** Trích xuất một phần chuỗi.

string text = "Unity3D"; Console.WriteLine(text.Substring(0, 5)); *// Kết quả: Unity*

1. **Contains():** Kiểm tra xem chuỗi có chứa một chuỗi con không.

string text = "Game Development"; Console.WriteLine(text.Contains("Game")); *// Kết quả: True*

1. **Replace():** Thay thế chuỗi con.

string text = "Hello, World!"; Console.WriteLine(text.Replace("World", "Unity")); *// Kết quả: Hello, Unity!*

**5.2. Lớp StringBuilder**

Mặc dù lớp String rất hữu ích, nhưng việc thay đổi chuỗi nhiều lần có thể làm giảm hiệu suất. Để giải quyết vấn đề này, C# cung cấp lớp StringBuilder, tối ưu hơn khi thao tác với chuỗi lớn hoặc thay đổi chuỗi nhiều lần.

**Khởi tạo StringBuilder**

using System.Text; StringBuilder sb = new StringBuilder("Hello");

**Các phương thức phổ biến của StringBuilder**

1. **Append():** Thêm chuỗi vào cuối.

sb.Append(", Unity!"); Console.WriteLine(sb.ToString()); *// Kết quả: Hello, Unity!*

1. **Insert():** Chèn chuỗi vào vị trí chỉ định.

sb.Insert(5, " World"); Console.WriteLine(sb.ToString()); *// Kết quả: Hello World, Unity!*

1. **Remove():** Xóa chuỗi con từ vị trí chỉ định.

sb.Remove(5, 6); Console.WriteLine(sb.ToString()); *// Kết quả: Hello, Unity!*

1. **Replace():** Thay thế chuỗi con.

sb.Replace("Unity", "C#"); Console.WriteLine(sb.ToString()); *// Kết quả: Hello, C#!*

**5.3. Nhập dữ liệu chuỗi từ người dùng**

Bạn có thể dùng phương thức Console.ReadLine() để nhận chuỗi đầu vào từ người dùng.

**Ví dụ: Nhập và xử lý chuỗi**

Console.Write("Nhập tên của bạn: "); string name = Console.ReadLine(); Console.WriteLine($"Xin chào, {name}!");

Kết quả:

css

Nhập tên của bạn: Alice Xin chào, Alice!

**5.4. Kết hợp mọi thứ**

Dưới đây là một ví dụ hoàn chỉnh sử dụng các phương thức làm việc với chuỗi:

**Ví dụ: Kiểm tra từ khóa trong một chuỗi**

using System; using System.Text; class Program { static void Main() { Console.Write("Nhập một câu: "); string input = Console.ReadLine(); Console.Write("Nhập từ khóa cần tìm: "); string keyword = Console.ReadLine(); if (input.Contains(keyword)) { Console.WriteLine($"Câu có chứa từ khóa: {keyword}"); } else { Console.WriteLine($"Câu không chứa từ khóa: {keyword}"); } } }

Khi chạy chương trình:

less

Nhập một câu: Tôi yêu lập trình Unity Nhập từ khóa cần tìm: Unity Câu có chứa từ khóa: Unity

**5.5. Lỗi thường gặp khi làm việc với chuỗi**

1. **NullReferenceException:** Sử dụng chuỗi khi nó chưa được khởi tạo.

string text = null; Console.WriteLine(text.Length); *// Lỗi*

1. **IndexOutOfRangeException:** Truy cập ngoài giới hạn độ dài chuỗi.

string text = "Unity"; Console.WriteLine(text[10]); *// Lỗi*

1. **Thay đổi chuỗi quá nhiều lần mà không dùng StringBuilder.**
   * Nếu cần thay đổi chuỗi liên tục, hãy sử dụng StringBuilder để cải thiện hiệu suất.

**Tóm tắt chương 5**

* Lớp String hỗ trợ nhiều phương thức để xử lý chuỗi như cắt, tìm kiếm, thay thế.
* Lớp StringBuilder tối ưu hơn khi làm việc với chuỗi lớn hoặc cần thay đổi nhiều lần.
* Khi làm việc với chuỗi, hãy cẩn thận với các lỗi thường gặp như null hoặc truy cập ngoài giới hạn.

**Chương 6: Cơ bản về Unity 2D**

Trong chương này, chúng ta sẽ bắt đầu làm quen với Unity và cách tạo ra các trò chơi 2D cơ bản. Unity là một công cụ mạnh mẽ để phát triển game, và việc hiểu các thành phần cơ bản trong Unity sẽ giúp bạn xây dựng những trò chơi thú vị.

**6.1. Trò chơi tệ nhất mọi thời đại**

Bắt đầu với một ví dụ đơn giản, chúng ta sẽ tạo ra một trò chơi 2D đơn giản nhất có thể. Đây sẽ là một trò chơi mà bạn chỉ cần điều khiển một đối tượng trên màn hình mà không có nhiều yếu tố phức tạp.

**Ví dụ: Trò chơi không phải game (Not A Game)**

Chúng ta sẽ tạo ra một dự án Unity mới và đặt tên là **NotAGame**. Trong trò chơi này, chúng ta sẽ điều khiển một đối tượng đơn giản di chuyển trên màn hình bằng các phím mũi tên.

1. **Tạo một đối tượng GameObject:**
   * Mở Unity và tạo một dự án 2D mới.
   * Chọn **GameObject > 2D Object > Sprite** để tạo ra một đối tượng.
   * Bạn có thể kéo một hình ảnh vào trong Sprite Renderer để thay đổi hình dạng của đối tượng.
2. **Thêm Script vào GameObject:**
   * Chọn đối tượng vừa tạo và vào **Inspector**.
   * Nhấn nút **Add Component**, tìm và thêm **New Script**.
   * Đặt tên script là PlayerController và bắt đầu viết mã trong script này.

**Mã điều khiển đối tượng di chuyển:**

using UnityEngine; public class PlayerController : MonoBehaviour { public float speed = 5f; *// Tốc độ di chuyển* void Update() { *// Lấy input từ người chơi* float horizontal = Input.GetAxis("Horizontal"); float vertical = Input.GetAxis("Vertical"); *// Tính toán chuyển động* Vector2 movement = new Vector2(horizontal, vertical) \* speed \* Time.deltaTime; *// Di chuyển đối tượng* transform.Translate(movement); } }

Trong script này:

* **Input.GetAxis("Horizontal")** và **Input.GetAxis("Vertical")** giúp bạn nhận giá trị đầu vào từ phím mũi tên hoặc các phím WASD.
* **transform.Translate()** di chuyển đối tượng theo vector bạn tính toán.

1. **Chạy trò chơi:**
   * Sau khi thêm script, nhấn **Play** trong Unity để thử nghiệm trò chơi. Bạn sẽ thấy đối tượng có thể di chuyển lên, xuống, trái, phải.

**6.2. Prefabs**

**Prefabs** là các mẫu đối tượng có thể tái sử dụng trong Unity. Khi bạn tạo một đối tượng, bạn có thể biến nó thành một prefab để sử dụng nhiều lần mà không cần tạo lại từ đầu.

**Cách tạo Prefab:**

1. Tạo đối tượng trong scene.
2. Kéo đối tượng vào thư mục **Assets** trong cửa sổ **Project**. Unity sẽ tự động biến nó thành một prefab.
3. Bạn có thể tạo nhiều bản sao của prefab này trong scene mà không phải thay đổi các thuộc tính của từng đối tượng.

**6.3. Unity Circles Revisited (Làm lại hình tròn trong Unity)**

Chúng ta sẽ tiếp tục phát triển trò chơi bằng cách thêm các hình tròn vào scene và làm cho chúng tương tác với người chơi.

**Thêm đối tượng hình tròn vào Unity:**

1. Vào **GameObject > 2D Object > Circle** để thêm hình tròn vào scene.
2. Bạn có thể thay đổi kích thước và màu sắc của hình tròn trong cửa sổ **Inspector**.

**Thêm Script cho hình tròn:**

* Tạo một script mới gọi là CircleController và gắn vào đối tượng hình tròn.
* Viết mã để di chuyển hình tròn giống như đối tượng người chơi.

using UnityEngine; public class CircleController : MonoBehaviour { public float speed = 3f; void Update() { float horizontal = Input.GetAxis("Horizontal"); float vertical = Input.GetAxis("Vertical"); Vector2 movement = new Vector2(horizontal, vertical) \* speed \* Time.deltaTime; transform.Translate(movement); } }

Giống như PlayerController, script này sẽ giúp hình tròn di chuyển dựa trên input của người chơi.

**6.4. Kết hợp mọi thứ lại với nhau**

Bây giờ, chúng ta đã có một đối tượng có thể di chuyển và một hình tròn có thể di chuyển trên màn hình. Bạn có thể thử nghiệm với các đối tượng khác và tạo ra những trò chơi phức tạp hơn bằng cách sử dụng các thành phần như **Collider** (để phát hiện va chạm), **Rigidbody2D** (để thêm lực vật lý), và **UI Elements** (để tạo giao diện người dùng).

**Ví dụ: Thêm va chạm**

1. Thêm một **Rigidbody2D** vào đối tượng hình tròn.
2. Thêm một **Collider2D** vào đối tượng để xác định phạm vi va chạm.

Bạn có thể điều chỉnh các thuộc tính này trong **Inspector** để thử nghiệm với các hiệu ứng vật lý và va chạm trong Unity.

**Tóm tắt chương 6**

* Unity cung cấp các công cụ mạnh mẽ để phát triển game 2D, từ việc tạo đối tượng đến điều khiển chuyển động và xử lý va chạm.
* Prefabs giúp bạn tái sử dụng các đối tượng mà không phải tạo lại từ đầu.
* Bạn có thể sử dụng các component như **Rigidbody2D** và **Collider2D** để thêm các hiệu ứng vật lý và tương tác giữa các đối tượng.

**Chương 7: Cấu trúc lựa chọn (Selection)**

Trong chương này, chúng ta sẽ học cách sử dụng các cấu trúc điều kiện trong C# để kiểm tra các điều kiện và thực thi các đoạn mã khác nhau dựa trên những điều kiện đó. Cấu trúc lựa chọn là một phần quan trọng trong lập trình, giúp chương trình có khả năng "quyết định" hành động tiếp theo dựa trên các điều kiện cụ thể.

**7.1. Cấu trúc điều kiện lựa chọn (Selection Control Structure)**

Cấu trúc lựa chọn trong C# cho phép bạn kiểm tra một hoặc nhiều điều kiện và thực hiện hành động tùy thuộc vào kết quả của những điều kiện đó.

**Các loại cấu trúc điều kiện chính:**

1. **Câu lệnh if**
2. **Câu lệnh else**
3. **Câu lệnh switch**

**7.2. Kiểm tra cấu trúc điều kiện (Testing Selection Control Structures)**

**Câu lệnh if**

Câu lệnh if là cách cơ bản để kiểm tra một điều kiện và thực thi một đoạn mã nếu điều kiện đó đúng.

**Cú pháp:**

if (condition) { *// Mã sẽ được thực thi nếu điều kiện là đúng* }

**Ví dụ:**

int age = 18; if (age >= 18) { Console.WriteLine("Bạn đủ tuổi trưởng thành."); }

Kết quả:

css

Bạn đủ tuổi trưởng thành.

**7.3. Câu lệnh else**

Bạn có thể kết hợp câu lệnh else với câu lệnh if để chỉ định hành động nếu điều kiện không đúng.

**Cú pháp:**

if (condition) { *// Mã sẽ được thực thi nếu điều kiện là đúng* } else { *// Mã sẽ được thực thi nếu điều kiện là sai* }

**Ví dụ:**

int age = 16; if (age >= 18) { Console.WriteLine("Bạn đủ tuổi trưởng thành."); } else { Console.WriteLine("Bạn chưa đủ tuổi trưởng thành."); }

Kết quả:

css

Bạn chưa đủ tuổi trưởng thành.

**7.4. Câu lệnh else if**

Nếu bạn muốn kiểm tra nhiều điều kiện khác nhau, bạn có thể sử dụng câu lệnh else if để kiểm tra từng điều kiện riêng biệt.

**Cú pháp:**

if (condition1) { *// Mã thực thi khi điều kiện 1 đúng* } else if (condition2) { *// Mã thực thi khi điều kiện 2 đúng* } else { *// Mã thực thi khi không điều kiện nào đúng* }

**Ví dụ:**

int temperature = 30; if (temperature >= 30) { Console.WriteLine("Hôm nay trời nóng."); } else if (temperature >= 20) { Console.WriteLine("Hôm nay trời ấm."); } else { Console.WriteLine("Hôm nay trời lạnh."); }

Kết quả:

css

Hôm nay trời nóng.

**7.5. Câu lệnh switch**

Khi bạn cần kiểm tra nhiều giá trị khác nhau cho một biến, câu lệnh switch là một lựa chọn hiệu quả hơn so với sử dụng nhiều câu lệnh if-else.

**Cú pháp:**

switch (variable) { case value1: *// Mã thực thi khi variable == value1* break; case value2: *// Mã thực thi khi variable == value2* break; default: *// Mã thực thi khi không có case nào khớp* break; }

**Ví dụ:**

int day = 3; switch (day) { case 1: Console.WriteLine("Thứ Hai"); break; case 2: Console.WriteLine("Thứ Ba"); break; case 3: Console.WriteLine("Thứ Tư"); break; case 4: Console.WriteLine("Thứ Năm"); break; case 5: Console.WriteLine("Thứ Sáu"); break; default: Console.WriteLine("Ngày không hợp lệ"); break; }

Kết quả:

css

Thứ Tư

**7.6. Thời gian (Timers)**

Cấu trúc lựa chọn cũng có thể được sử dụng với **timer** trong Unity hoặc trong các ứng dụng khác để kiểm tra điều kiện theo thời gian.

**Ví dụ sử dụng Timer trong C#:**

using System; class Program { static void Main() { DateTime startTime = DateTime.Now; DateTime endTime = startTime.AddSeconds(5); while (DateTime.Now < endTime) { Console.WriteLine("Đang đếm..."); } Console.WriteLine("Đã hết thời gian!"); } }

Kết quả:

css

Đang đếm... Đang đếm... Đang đếm... Đang đếm... Đang đếm... Đã hết thời gian!

**7.7. Kết hợp mọi thứ lại với nhau**

Bạn có thể kết hợp cấu trúc lựa chọn với các đối tượng hoặc trạng thái trong game. Ví dụ, trong Unity, bạn có thể sử dụng các câu lệnh if, else if, switch để quyết định hành động của nhân vật trong game dựa trên các sự kiện hoặc trạng thái trò chơi.

**Ví dụ trong Unity (Sử dụng câu lệnh**switch**để thay đổi trạng thái nhân vật):**

using UnityEngine; public class Player : MonoBehaviour { public enum State { Idle, Walking, Running, Jumping } public State currentState; void Update() { switch (currentState) { case State.Idle: Debug.Log("Nhân vật đang đứng yên."); break; case State.Walking: Debug.Log("Nhân vật đang đi."); break; case State.Running: Debug.Log("Nhân vật đang chạy."); break; case State.Jumping: Debug.Log("Nhân vật đang nhảy."); break; default: Debug.Log("Trạng thái không hợp lệ."); break; } } }

**Tóm tắt chương 7**

* **Câu lệnh if** và **else** giúp bạn kiểm tra điều kiện và thực thi mã tùy theo điều kiện đó.
* **Câu lệnh switch** rất hữu ích khi bạn cần kiểm tra một biến với nhiều giá trị khác nhau.
* **Cấu trúc lựa chọn** giúp chương trình có thể ra quyết định dựa trên các điều kiện khác nhau, điều này rất quan trọng khi phát triển game và các ứng dụng phức tạp.

**Chương 8: Unity - Chuột, Bàn phím và Tay cầm chơi game**

Trong chương này, chúng ta sẽ học cách xử lý đầu vào từ chuột, bàn phím và tay cầm chơi game trong Unity. Đầu vào là một phần rất quan trọng trong các trò chơi vì nó cho phép người chơi tương tác với thế giới trong game.

**8.1. Input Manager**

Unity sử dụng một hệ thống được gọi là **Input Manager** để quản lý đầu vào từ các thiết bị như chuột, bàn phím và tay cầm chơi game. Bạn có thể cấu hình các nút bấm, trục (axis) và các hành động tương ứng trong cửa sổ **Edit > Project Settings > Input Manager**.

**Các mục cơ bản trong Input Manager:**

1. **Axes:** Đại diện cho các trục điều khiển, ví dụ: "Horizontal" và "Vertical" cho các phím WASD hoặc cần điều khiển.
2. **Buttons:** Đại diện cho các nút trên bàn phím hoặc tay cầm chơi game, ví dụ: "Jump" hoặc "Fire1".

**8.2. Xử lý đầu vào từ chuột**

**Nhận vị trí con trỏ chuột**

Bạn có thể sử dụng Input.mousePosition để lấy tọa độ của con trỏ chuột trên màn hình.

**Ví dụ: Hiển thị vị trí con trỏ chuột**

using UnityEngine;

public class MouseInput : MonoBehaviour

{

void Update()

{

Vector3 mousePosition = Input.mousePosition;

Debug.Log($"Vị trí chuột: {mousePosition}");

}

}

**Nhấp chuột**

Unity cung cấp các phương thức để kiểm tra xem người chơi có nhấn chuột hay không:

* **Input.GetMouseButton(0):** Kiểm tra nếu nút chuột trái đang được nhấn.
* **Input.GetMouseButtonDown(0):** Kiểm tra khi nút chuột trái vừa được nhấn.
* **Input.GetMouseButtonUp(0):** Kiểm tra khi nút chuột trái vừa được thả.

**Ví dụ: Kiểm tra nhấp chuột**

using UnityEngine;

public class MouseClick : MonoBehaviour

{

void Update()

{

if (Input.GetMouseButtonDown(0))

{

Debug.Log("Nhấn nút chuột trái.");

}

if (Input.GetMouseButtonUp(0))

{

Debug.Log("Thả nút chuột trái.");

}

}

}

**8.3. Xử lý đầu vào từ bàn phím**

Unity sử dụng các phương thức của lớp Input để kiểm tra đầu vào từ bàn phím:

* **Input.GetKey(KeyCode):** Kiểm tra nếu một phím đang được giữ.
* **Input.GetKeyDown(KeyCode):** Kiểm tra khi phím vừa được nhấn.
* **Input.GetKeyUp(KeyCode):** Kiểm tra khi phím vừa được thả.

**Ví dụ: Điều khiển đối tượng bằng bàn phím**

using UnityEngine;

public class KeyboardInput : MonoBehaviour

{

public float speed = 5f;

void Update()

{

Vector3 movement = Vector3.zero;

if (Input.GetKey(KeyCode.W))

{

movement += Vector3.up; // Di chuyển lên

}

if (Input.GetKey(KeyCode.S))

{

movement += Vector3.down; // Di chuyển xuống

}

if (Input.GetKey(KeyCode.A))

{

movement += Vector3.left; // Di chuyển trái

}

if (Input.GetKey(KeyCode.D))

{

movement += Vector3.right; // Di chuyển phải

}

transform.Translate(movement \* speed \* Time.deltaTime);

}

}

**8.4. Xử lý đầu vào từ tay cầm chơi game**

Unity hỗ trợ các tay cầm chơi game thông qua Input Manager. Các trục và nút bấm của tay cầm được ánh xạ tương tự như bàn phím và chuột.

**Nhận giá trị trục (Axis)**

Bạn có thể sử dụng Input.GetAxis() để nhận giá trị từ các trục của tay cầm, ví dụ:

* **Horizontal:** Trục ngang của cần điều khiển.
* **Vertical:** Trục dọc của cần điều khiển.

**Ví dụ: Điều khiển đối tượng bằng tay cầm**

using UnityEngine;

public class GamepadInput : MonoBehaviour

{

public float speed = 5f;

void Update()

{

float horizontal = Input.GetAxis("Horizontal");

float vertical = Input.GetAxis("Vertical");

Vector3 movement = new Vector3(horizontal, vertical, 0) \* speed \* Time.deltaTime;

transform.Translate(movement);

}

}

**Nhận giá trị nút bấm**

Bạn có thể sử dụng các nút bấm trên tay cầm tương tự như phím bàn phím:

* **Input.GetButton("Jump"):** Kiểm tra nút nhảy (Jump).
* **Input.GetButtonDown("Fire1"):** Kiểm tra khi nút bắn (Fire1) vừa được nhấn.

**Ví dụ: Sử dụng nút bắn từ tay cầm**

using UnityEngine;

public class FireAction : MonoBehaviour

{

void Update()

{

if (Input.GetButtonDown("Fire1"))

{

Debug.Log("Bắn đạn!");

}

}

}

**8.5. Kết hợp mọi thứ**

Bây giờ, chúng ta sẽ kết hợp chuột, bàn phím và tay cầm để tạo một hệ thống điều khiển linh hoạt cho trò chơi.

**Ví dụ: Điều khiển nhân vật với nhiều thiết bị**

using UnityEngine;

public class MultiInputController : MonoBehaviour

{

public float speed = 5f;

void Update()

{

// Điều khiển bằng bàn phím hoặc tay cầm

float horizontal = Input.GetAxis("Horizontal");

float vertical = Input.GetAxis("Vertical");

Vector3 movement = new Vector3(horizontal, vertical, 0) \* speed \* Time.deltaTime;

transform.Translate(movement);

// Kiểm tra nhấn nút bắn

if (Input.GetButtonDown("Fire1") || Input.GetMouseButtonDown(0))

{

Debug.Log("Bắn đạn!");

}

// Kiểm tra nhấn ESC để thoát

if (Input.GetKeyDown(KeyCode.Escape))

{

Debug.Log("Thoát trò chơi.");

Application.Quit();

}

}

}

**Tóm tắt chương 8**

* Unity cung cấp các phương thức mạnh mẽ để xử lý đầu vào từ chuột, bàn phím và tay cầm.
* **Input Manager** cho phép cấu hình và ánh xạ các hành động trong trò chơi với các phím hoặc nút cụ thể.
* Bạn có thể kết hợp nhiều thiết bị đầu vào để tạo trải nghiệm chơi game linh hoạt và thú vị.

**Chương 9: Mảng và danh sách (Arrays and Lists)**

Trong chương này, chúng ta sẽ tìm hiểu về **mảng (arrays)** và **danh sách (lists)** trong C#. Đây là hai cấu trúc dữ liệu quan trọng để lưu trữ và quản lý một tập hợp dữ liệu.

**9.1. Mảng (Arrays)**

**Mảng** là một cấu trúc dữ liệu cho phép lưu trữ nhiều giá trị cùng loại. Các giá trị trong mảng được truy cập bằng chỉ số (index), bắt đầu từ 0.

**Khai báo và khởi tạo mảng**

Bạn có thể khai báo và khởi tạo mảng như sau:

**Cú pháp:**

type[] arrayName = new type[size];

**Ví dụ:**

int[] numbers = new int[5]; // Mảng chứa 5 số nguyên

numbers[0] = 10; // Gán giá trị cho phần tử đầu tiên

numbers[1] = 20; // Gán giá trị cho phần tử thứ hai

Bạn cũng có thể khởi tạo mảng với các giá trị sẵn có:

int[] numbers = { 10, 20, 30, 40, 50 };

**Duyệt qua mảng**

Bạn có thể sử dụng vòng lặp for hoặc foreach để duyệt qua mảng.

**Ví dụ:**

int[] numbers = { 1, 2, 3, 4, 5 };

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

{

Console.WriteLine($"Phần tử {i}: {numbers[i]}");

}

// Sử dụng foreach

foreach (int number in numbers)

{

Console.WriteLine($"Phần tử: {number}");

}

**Một số thao tác cơ bản trên mảng**

1. **Tìm độ dài của mảng:**

int length = numbers.Length;

Console.WriteLine($"Độ dài mảng: {length}");

1. **Sắp xếp mảng:** Sử dụng Array.Sort() để sắp xếp mảng.

Array.Sort(numbers);

1. **Đảo ngược mảng:** Sử dụng Array.Reverse() để đảo ngược thứ tự các phần tử.

Array.Reverse(numbers);

**9.2. Danh sách (Lists)**

**Danh sách (List)** là một cấu trúc dữ liệu linh hoạt hơn mảng. Bạn có thể thêm, xóa, hoặc thay đổi kích thước danh sách trong quá trình chạy chương trình. Danh sách trong C# được định nghĩa trong namespace System.Collections.Generic.

**Khai báo và khởi tạo danh sách**

Bạn có thể sử dụng lớp List<T> để tạo danh sách, trong đó T là kiểu dữ liệu của danh sách.

**Cú pháp:**

List<type> listName = new List<type>();

**Ví dụ:**

List<int> numbers = new List<int>();

numbers.Add(10); // Thêm phần tử vào danh sách

numbers.Add(20);

numbers.Add(30);

**Duyệt qua danh sách**

Bạn có thể duyệt qua danh sách bằng for hoặc foreach.

**Ví dụ:**

List<string> names = new List<string> { "Alice", "Bob", "Charlie" };

foreach (string name in names)

{

Console.WriteLine(name);

}

**Một số phương thức quan trọng trong danh sách**

1. **Thêm phần tử:**

numbers.Add(40); // Thêm phần tử 40

1. **Chèn phần tử vào vị trí cụ thể:**

numbers.Insert(1, 15); // Chèn số 15 vào vị trí thứ 1

1. **Xóa phần tử:**

numbers.Remove(10); // Xóa phần tử có giá trị 10

numbers.RemoveAt(0); // Xóa phần tử ở vị trí đầu tiên

numbers.Clear(); // Xóa toàn bộ danh sách

1. **Tìm kiếm phần tử:**

bool exists = numbers.Contains(20); // Kiểm tra xem danh sách có chứa số 20 không

int index = numbers.IndexOf(20); // Tìm vị trí của số 20

1. **Sắp xếp danh sách:**

numbers.Sort();

**9.3. So sánh mảng và danh sách**

| **Đặc điểm** | **Mảng** | **Danh sách** |
| --- | --- | --- |
| **Kích thước** | Cố định sau khi khởi tạo | Linh hoạt, có thể thay đổi kích thước |
| **Thao tác thêm/xóa** | Không thể thêm/xóa phần tử | Hỗ trợ thêm, xóa phần tử dễ dàng |
| **Hiệu suất** | Nhanh hơn khi kích thước không đổi | Chậm hơn một chút do phải quản lý động |
| **Sử dụng** | Tốt khi biết trước kích thước và dữ liệu | Phù hợp khi dữ liệu thay đổi liên tục |

**9.4. Ứng dụng trong Unity**

**Ví dụ: Sử dụng mảng để lưu trữ vị trí**

Bạn có thể sử dụng mảng để lưu trữ các vị trí của đối tượng trong Unity.

**Mã ví dụ:**

using UnityEngine;

public class ArrayExample : MonoBehaviour

{

public Vector3[] positions;

void Start()

{

foreach (Vector3 pos in positions)

{

Debug.Log($"Vị trí: {pos}");

}

}

}

Trong Unity Editor, bạn có thể gán các vị trí vào mảng positions trong cửa sổ **Inspector**.

**Ví dụ: Sử dụng danh sách để lưu trữ các đối tượng**

Bạn có thể sử dụng danh sách để lưu trữ các đối tượng được tạo động trong game.

**Mã ví dụ:**

using System.Collections.Generic;

using UnityEngine;

public class ListExample : MonoBehaviour

{

public GameObject prefab;

private List<GameObject> spawnedObjects = new List<GameObject>();

void Update()

{

if (Input.GetKeyDown(KeyCode.Space))

{

GameObject obj = Instantiate(prefab, Vector3.zero, Quaternion.identity);

spawnedObjects.Add(obj);

Debug.Log($"Đã tạo {spawnedObjects.Count} đối tượng.");

}

if (Input.GetKeyDown(KeyCode.Delete))

{

foreach (GameObject obj in spawnedObjects)

{

Destroy(obj);

}

spawnedObjects.Clear();

Debug.Log("Đã xóa tất cả đối tượng.");

}

}

}

**Tóm tắt chương 9**

* **Mảng (Arrays):** Lưu trữ các giá trị cố định, hiệu quả về hiệu suất.
* **Danh sách (Lists):** Linh hoạt hơn, hỗ trợ thêm, xóa, và thay đổi kích thước.
* Hiểu được cách sử dụng mảng và danh sách sẽ giúp bạn quản lý dữ liệu hiệu quả hơn trong lập trình C# và Unity.

**Chương 10: Lập trình hướng đối tượng (OOP)**

**Lập trình hướng đối tượng (Object-Oriented Programming)** là một mô hình lập trình dựa trên khái niệm về "đối tượng" (objects). Đối tượng là những thực thể đại diện cho các vật thể trong thế giới thực, và chúng có thuộc tính (properties) cũng như hành vi (methods). OOP là một trong những khái niệm quan trọng trong C#.

**10.1. Nguyên lý cơ bản của OOP**

OOP trong C# xoay quanh bốn nguyên lý chính:

1. **Tính đóng gói (Encapsulation):**
   * Giúp bảo vệ dữ liệu bằng cách ẩn chi tiết triển khai bên trong các lớp.
   * Chỉ cho phép truy cập vào dữ liệu thông qua các phương thức hoặc thuộc tính được chỉ định.
   * Sử dụng các từ khóa như private, public, protected.
2. **Tính kế thừa (Inheritance):**
   * Cho phép một lớp con (child class) kế thừa các thuộc tính và phương thức từ lớp cha (parent class).
   * Sử dụng từ khóa : ParentClass.
3. **Tính đa hình (Polymorphism):**
   * Cho phép các phương thức trong lớp con được ghi đè (override) để thay đổi hành vi của lớp cha.
   * Sử dụng từ khóa virtual, override, hoặc abstract.
4. **Tính trừu tượng (Abstraction):**
   * Giúp ẩn đi những chi tiết phức tạp và chỉ cung cấp các thông tin cần thiết cho người dùng.
   * Được triển khai bằng cách sử dụng lớp trừu tượng (abstract class) hoặc giao diện (interface).

**10.2. Lớp và đối tượng**

**Tạo lớp và đối tượng**

Lớp là một mẫu thiết kế, và đối tượng là thể hiện của lớp.

**Ví dụ:**

class Animal

{

public string Name; // Thuộc tính

public void Speak() // Hành vi

{

Console.WriteLine($"{Name} đang kêu.");

}

}

class Program

{

static void Main()

{

Animal dog = new Animal(); // Tạo đối tượng

dog.Name = "Chó";

dog.Speak(); // Gọi phương thức

}

}

Kết quả:

Chó đang kêu.

**10.3. Tính đóng gói (Encapsulation)**

**Sử dụng các thuộc tính truy cập (Access Modifiers)**

Các từ khóa truy cập trong C#:

* **public:** Có thể truy cập từ bất kỳ đâu.
* **private:** Chỉ có thể truy cập trong lớp hiện tại.
* **protected:** Chỉ truy cập được trong lớp hiện tại và lớp con.
* **internal:** Truy cập trong cùng một assembly.

**Ví dụ:**

class Animal

{

private string name;

public void SetName(string newName)

{

name = newName;

}

public string GetName()

{

return name;

}

}

class Program

{

static void Main()

{

Animal cat = new Animal();

cat.SetName("Mèo");

Console.WriteLine(cat.GetName());

}

}

Kết quả:

Mèo

**10.4. Tính kế thừa (Inheritance)**

Một lớp con có thể kế thừa thuộc tính và phương thức từ lớp cha.

**Ví dụ:**

class Animal

{

public string Name;

public void Speak()

{

Console.WriteLine($"{Name} đang kêu.");

}

}

class Dog : Animal

{

public void WagTail()

{

Console.WriteLine($"{Name} đang vẫy đuôi.");

}

}

class Program

{

static void Main()

{

Dog dog = new Dog();

dog.Name = "Chó";

dog.Speak();

dog.WagTail();

}

}

Kết quả:

css

Chó đang kêu.

Chó đang vẫy đuôi.

**10.5. Tính đa hình (Polymorphism)**

**Phương thức ghi đè (Method Overriding)**

Lớp con có thể thay đổi hành vi của lớp cha bằng cách ghi đè phương thức.

**Ví dụ:**

class Animal

{

public virtual void Speak()

{

Console.WriteLine("Con vật đang kêu.");

}

}

class Dog : Animal

{

public override void Speak()

{

Console.WriteLine("Chó đang sủa.");

}

}

class Program

{

static void Main()

{

Animal animal = new Animal();

animal.Speak();

Dog dog = new Dog();

dog.Speak();

}

}

Kết quả:

css

Con vật đang kêu.

Chó đang sủa.

**10.6. Tính trừu tượng (Abstraction)**

**Lớp trừu tượng**

Lớp trừu tượng được khai báo bằng từ khóa abstract và không thể tạo đối tượng trực tiếp từ lớp này.

**Ví dụ:**

abstract class Animal

{

public abstract void Speak(); // Phương thức trừu tượng

}

class Dog : Animal

{

public override void Speak()

{

Console.WriteLine("Chó đang sủa.");

}

}

class Program

{

static void Main()

{

Animal dog = new Dog();

dog.Speak();

}

}

Kết quả:

css

Chó đang sủa.

**Giao diện (Interface)**

Giao diện định nghĩa các hành vi mà một lớp phải triển khai.

**Ví dụ:**

interface IAnimal

{

void Speak();

}

class Cat : IAnimal

{

public void Speak()

{

Console.WriteLine("Mèo đang kêu.");

}

}

class Program

{

static void Main()

{

IAnimal cat = new Cat();

cat.Speak();

}

}

Kết quả:

Mèo đang kêu.

**10.7. Ứng dụng trong Unity**

**Tạo các lớp kế thừa trong Unity**

Bạn có thể sử dụng OOP để tạo các hệ thống đối tượng trong game.

**Ví dụ: Hệ thống kẻ thù**

using UnityEngine;

abstract class Enemy : MonoBehaviour

{

public abstract void Attack();

}

class Goblin : Enemy

{

public override void Attack()

{

Debug.Log("Goblin tấn công bằng gậy.");

}

}

class Orc : Enemy

{

public override void Attack()

{

Debug.Log("Orc tấn công bằng rìu.");

}

}

Trong Unity, bạn có thể gắn các script này vào các đối tượng tương ứng, và chúng sẽ hành xử theo cách riêng của chúng.

**Tóm tắt chương 10**

* **Lập trình hướng đối tượng (OOP)** trong C# giúp tổ chức mã nguồn tốt hơn bằng cách sử dụng lớp và đối tượng.
* Bốn nguyên lý OOP:
  + **Đóng gói (Encapsulation):** Bảo vệ dữ liệu.
  + **Kế thừa (Inheritance):** Chia sẻ mã giữa các lớp.
  + **Đa hình (Polymorphism):** Thay đổi hành vi của lớp cha.
  + **Trừu tượng (Abstraction):** Ẩn chi tiết triển khai.
* Áp dụng OOP trong Unity giúp xây dựng các hệ thống game phức tạp một cách rõ ràng và dễ bảo trì.

**Chương 11: Vòng lặp (Loops)**

Vòng lặp là một phần không thể thiếu trong lập trình, cho phép bạn lặp lại một đoạn mã nhiều lần mà không cần viết lại mã đó. Trong chương này, chúng ta sẽ tìm hiểu các loại vòng lặp trong C# và cách sử dụng chúng để xử lý các tác vụ lặp đi lặp lại.

**11.1. Các loại vòng lặp trong C#**

C# cung cấp bốn loại vòng lặp chính:

1. **Vòng lặp for:** Lặp với số lần xác định.
2. **Vòng lặp while:** Lặp dựa trên điều kiện.
3. **Vòng lặp do-while:** Lặp ít nhất một lần và kiểm tra điều kiện sau.
4. **Vòng lặp foreach:** Lặp qua các phần tử trong tập hợp (collection).

**11.2. Vòng lặp for**

Vòng lặp for được sử dụng khi bạn biết trước số lần lặp.

**Cú pháp:**

for (khởi\_tạo; điều\_kiện; tăng\_giảm)

{

// Mã thực thi

}

**Ví dụ: In các số từ 1 đến 10**

for (int i = 1; i <= 10; i++)

{

Console.WriteLine(i);

}

Kết quả:

python

1

2

3

...

10

**Ứng dụng trong Unity:**

using UnityEngine;

public class ForLoopExample : MonoBehaviour

{

void Start()

{

for (int i = 0; i < 5; i++)

{

Debug.Log($"Vòng lặp thứ {i + 1}");

}

}

}

**11.3. Vòng lặp while**

Vòng lặp while lặp lại khi điều kiện vẫn đúng (true).

**Cú pháp:**

while (điều\_kiện)

{

// Mã thực thi

}

**Ví dụ: Tính tổng các số từ 1 đến 5**

int i = 1;

int sum = 0;

while (i <= 5)

{

sum += i;

i++;

}

Console.WriteLine($"Tổng là: {sum}");

Kết quả:

r

Tổng là: 15

**11.4. Vòng lặp do-while**

Vòng lặp do-while luôn thực thi ít nhất một lần, sau đó kiểm tra điều kiện.

**Cú pháp:**

do

{

// Mã thực thi

} while (điều\_kiện);

**Ví dụ: Đoán số bí mật**

int secretNumber = 7;

int guess;

do

{

Console.Write("Đoán số: ");

guess = int.Parse(Console.ReadLine());

} while (guess != secretNumber);

Console.WriteLine("Bạn đã đoán đúng!");

**11.5. Vòng lặp foreach**

Vòng lặp foreach được sử dụng để duyệt qua các phần tử trong một tập hợp, ví dụ như mảng hoặc danh sách.

**Cú pháp:**

foreach (var phần\_tử in tập\_hợp)

{

// Mã thực thi

}

**Ví dụ: Duyệt qua một mảng**

string[] names = { "Alice", "Bob", "Charlie" };

foreach (string name in names)

{

Console.WriteLine($"Tên: {name}");

}

Kết quả:

makefile

Tên: Alice

Tên: Bob

Tên: Charlie

**Ứng dụng trong Unity:**

using UnityEngine;

public class ForeachExample : MonoBehaviour

{

public GameObject[] objects;

void Start()

{

foreach (GameObject obj in objects)

{

Debug.Log($"Tên đối tượng: {obj.name}");

}

}

}

**11.6. Từ khóa điều khiển vòng lặp**

**1. break**

Dừng vòng lặp ngay lập tức.

**Ví dụ: Dừng vòng lặp khi gặp số 5**

for (int i = 1; i <= 10; i++)

{

if (i == 5)

{

break;

}

Console.WriteLine(i);

}

Kết quả:

1

2

3

4

**2. continue**

Bỏ qua phần còn lại của vòng lặp hiện tại và chuyển sang lần lặp tiếp theo.

**Ví dụ: Bỏ qua số chẵn**

for (int i = 1; i <= 10; i++)

{

if (i % 2 == 0)

{

continue;

}

Console.WriteLine(i);

}

Kết quả:

1

3

5

7

9

**3. return**

Thoát khỏi phương thức hiện tại và kết thúc tất cả các vòng lặp trong phương thức.

**Ví dụ:**

void PrintNumbers()

{

for (int i = 1; i <= 10; i++)

{

if (i == 5)

{

return;

}

Console.WriteLine(i);

}

}

PrintNumbers();

Kết quả:

1

2

3

4

**11.7. Ứng dụng vòng lặp trong Unity**

Vòng lặp thường được sử dụng để tạo đối tượng, kiểm tra trạng thái, hoặc xử lý logic trong Unity.

**Ví dụ: Sinh các đối tượng trong Unity**

using UnityEngine;

public class SpawnObjects : MonoBehaviour

{

public GameObject prefab;

public int count = 5;

void Start()

{

for (int i = 0; i < count; i++)

{

Vector3 position = new Vector3(i \* 2.0f, 0, 0);

Instantiate(prefab, position, Quaternion.identity);

}

}

}

Khi chạy script này, 5 đối tượng sẽ được sinh ra dọc theo trục X, cách nhau 2 đơn vị.

**Tóm tắt chương 11**

* **Vòng lặp for:** Lặp số lần cố định.
* **Vòng lặp while:** Lặp khi điều kiện đúng.
* **Vòng lặp do-while:** Lặp ít nhất một lần.
* **Vòng lặp foreach:** Duyệt qua các phần tử trong tập hợp.
* Các từ khóa break, continue, và return giúp kiểm soát luồng thực thi trong vòng lặp.
* Vòng lặp là một công cụ mạnh mẽ để tự động hóa các tác vụ trong lập trình và phát triển game Unity.

**Chương 12: Quản lý lỗi và ngoại lệ (Error and Exception Handling)**

**Trong lập trình, lỗi (errors) là điều không thể tránh khỏi. Chúng có thể xảy ra vì lỗi logic, sai sót của lập trình viên, hoặc những điều kiện không lường trước như mất kết nối mạng hoặc nhập dữ liệu không hợp lệ. Quản lý lỗi hiệu quả giúp ứng dụng không bị gián đoạn và cung cấp phản hồi hữu ích cho người dùng.**

**12.1. Ngoại lệ (Exception)**

**Ngoại lệ (Exception)** trong C# là một loại lỗi xảy ra khi chương trình đang chạy. Khi một ngoại lệ được kích hoạt, chương trình sẽ tạm dừng và tìm kiếm cách xử lý lỗi.

**Ví dụ về ngoại lệ**

Dưới đây là một ví dụ về lỗi chia cho 0:

int a = 10;

int b = 0;

int result = a / b; // Lỗi ngoại lệ: DivideByZeroException

**12.2. Khối try-catch**

Khối try-catch được sử dụng để xử lý các ngoại lệ. Mã trong khối try được giám sát, và nếu có lỗi xảy ra, khối catch sẽ xử lý lỗi đó.

**Cú pháp:**

try

{

// Mã có thể gây ra lỗi

}

catch (ExceptionType ex)

{

// Xử lý lỗi

}

**Ví dụ: Bắt ngoại lệ chia cho 0**

try

{

int a = 10;

int b = 0;

int result = a / b;

}

catch (DivideByZeroException ex)

{

Console.WriteLine("Lỗi: Không thể chia cho 0.");

}

Kết quả:

less

Lỗi: Không thể chia cho 0.

**12.3. Khối finally**

Khối finally được sử dụng để thực thi mã bất kể ngoại lệ có xảy ra hay không. Nó thường được dùng để dọn dẹp tài nguyên (như đóng file, kết nối mạng).

**Cú pháp:**

try

{

// Mã có thể gây ra lỗi

}

catch (Exception ex)

{

// Xử lý lỗi

}

finally

{

// Dọn dẹp tài nguyên

}

**Ví dụ: Đảm bảo dọn dẹp tài nguyên**

try

{

Console.WriteLine("Thực thi mã...");

throw new Exception("Lỗi xảy ra!");

}

catch (Exception ex)

{

Console.WriteLine($"Lỗi: {ex.Message}");

}

finally

{

Console.WriteLine("Dọn dẹp tài nguyên.");

}

Kết quả:

less

Thực thi mã...

Lỗi: Lỗi xảy ra!

Dọn dẹp tài nguyên.

**12.4. Tạo ngoại lệ tùy chỉnh**

Bạn có thể tạo ra các ngoại lệ riêng bằng cách kế thừa từ lớp Exception.

**Ví dụ: Tạo ngoại lệ tùy chỉnh**

class InvalidScoreException : Exception

{

public InvalidScoreException(string message) : base(message)

{

}

}

class Program

{

static void Main()

{

int score = -10;

try

{

if (score < 0 || score > 100)

{

throw new InvalidScoreException("Điểm không hợp lệ!");

}

}

catch (InvalidScoreException ex)

{

Console.WriteLine($"Lỗi: {ex.Message}");

}

}

}

Kết quả:

less

Lỗi: Điểm không hợp lệ!

**12.5. Sử dụng throw để kích hoạt ngoại lệ**

Bạn có thể sử dụng từ khóa throw để kích hoạt ngoại lệ.

**Ví dụ:**

void CheckAge(int age)

{

if (age < 18)

{

throw new Exception("Bạn chưa đủ tuổi!");

}

}

try

{

CheckAge(16);

}

catch (Exception ex)

{

Console.WriteLine($"Lỗi: {ex.Message}");

}

Kết quả:

less

Lỗi: Bạn chưa đủ tuổi!

**12.6. Nhiều khối catch**

Bạn có thể sử dụng nhiều khối catch để xử lý các loại ngoại lệ khác nhau.

**Ví dụ:**

try

{

int[] numbers = { 1, 2, 3 };

Console.WriteLine(numbers[5]); // Lỗi: IndexOutOfRangeException

}

catch (DivideByZeroException)

{

Console.WriteLine("Lỗi: Chia cho 0.");

}

catch (IndexOutOfRangeException)

{

Console.WriteLine("Lỗi: Chỉ số ngoài phạm vi.");

}

catch (Exception)

{

Console.WriteLine("Lỗi không xác định.");

}

Kết quả:

less

Lỗi: Chỉ số ngoài phạm vi.

**12.7. Ứng dụng xử lý lỗi trong Unity**

**Ví dụ: Xử lý lỗi khi tải tài nguyên**

Khi phát triển game trong Unity, bạn có thể gặp lỗi như không tìm thấy tài nguyên.

**Mã ví dụ:**

using UnityEngine;

public class LoadResource : MonoBehaviour

{

void Start()

{

try

{

GameObject obj = Resources.Load<GameObject>("NonExistingResource");

if (obj == null)

{

throw new System.Exception("Không tìm thấy tài nguyên!");

}

Instantiate(obj);

}

catch (System.Exception ex)

{

Debug.LogError($"Lỗi: {ex.Message}");

}

}

}

Kết quả trong Unity Console:

less

Lỗi: Không tìm thấy tài nguyên!

**12.8. Các loại ngoại lệ thường gặp**

| **Ngoại lệ** | **Mô tả** |
| --- | --- |
| DivideByZeroException | Xảy ra khi chia cho 0. |
| NullReferenceException | Xảy ra khi truy cập đối tượng null. |
| IndexOutOfRangeException | Xảy ra khi truy cập chỉ số ngoài phạm vi trong mảng. |
| FileNotFoundException | Xảy ra khi không tìm thấy tệp. |
| InvalidOperationException | Xảy ra khi một thao tác không hợp lệ được thực hiện. |

**Tóm tắt chương 12**

* **Ngoại lệ** là các lỗi xảy ra khi chương trình đang chạy. Quản lý ngoại lệ giúp chương trình không bị gián đoạn.
* Sử dụng khối try-catch để xử lý lỗi và khối finally để dọn dẹp tài nguyên.
* Có thể tạo **ngoại lệ tùy chỉnh** để phù hợp với logic cụ thể của ứng dụng.
* Hiểu các ngoại lệ thường gặp giúp bạn xử lý lỗi một cách hiệu quả.