|  |  |
| --- | --- |
| **hàm Start() và Update()** | **Start():** Hàm này được gọi chỉ một lần khi kịch bản (script) được khởi chạy. Nó thường được sử dụng để thiết lập các giá trị ban đầu, khởi tạo các thành phần, gắn kết các đối tượng, hoặc làm bất kỳ công việc nào cần thiết khi script bắt đầu chạy.    **Update():** Hàm này được gọi mỗi frame trong quá trình chạy của trò chơi. Nó thường được sử dụng để cập nhật logic và xử lý của đối tượng trong trò chơi như di chuyển, xử lý đầu vào người chơi, xử lý va chạm và nhiều hơn nữa.    Khi viết script trong Unity, bạn có thể sử dụng **Start()** để khởi tạo và cấu hình, còn **Update()** để cập nhật logic của trò chơi dựa trên thời gian thực. Tuy nhiên, cần lưu ý rằng việc sử dụng **Update()** có thể ảnh hưởng đến hiệu suất của trò chơi nếu không được sử dụng một cách hiệu quả, đặc biệt là khi có nhiều đối tượng cần được cập nhật mỗi frame. |
| **Hàm Start()** | Hàm Start() được gọi một lần khi đối tượng được khởi tạo. Nó thường được sử dụng để thiết lập các giá trị ban đầu, khởi tạo các đối tượng, hoặc tìm kiếm các thành phần cần thiết để sử dụng trong đối tượng đó. Hàm này chỉ chạy một lần khi đối tượng được khởi tạo. |
| **Hàm Update()** | Hàm Update() được gọi mỗi frame trong game, tức là khoảng 60 lần mỗi giây (tùy thuộc vào tốc độ khung hình của game). Hàm này thường được sử dụng để cập nhật thông tin, logic hoặc hành vi của đối tượng dựa trên sự thay đổi của thời gian hoặc các sự kiện diễn ra trong trò chơi. |
| **Console.WriteLine()** | Console.WriteLine() là một hàm được sử dụng để xuất thông điệp hoặc dữ liệu ra màn hình console trong môi trường lập trình.  Tùy thuộc vào ngữ cảnh sử dụng, Console.WriteLine() thường được dùng để in ra màn hình console các thông điệp, giá trị biến, hoặc thông tin khác trong quá trình phát triển và debug chương trình. Trong Unity, nếu bạn đang viết các script trong Editor, thông điệp từ Console.WriteLine() sẽ xuất hiện trong cửa sổ Console của Unity Editor.    Tuy nhiên, trong các ứng dụng thực tế trong Unity, thay vì sử dụng Console.WriteLine() để ghi thông điệp ra console, thì thường người lập trình sẽ sử dụng hàm Debug.Log() để ghi log thông điệp ra cửa sổ Console của Unity. |
| **Debug.Log()** | Debug.Log() là một hàm rất quan trọng trong Unity được sử dụng để ghi log thông điệp hoặc giá trị của biến ra cửa sổ Console của Unity trong quá trình phát triển và debug game.  Hàm Debug.Log() cho phép bạn hiển thị thông điệp hoặc giá trị của biến trong quá trình chạy game. Điều này rất hữu ích để kiểm tra xem các giá trị của biến, thông điệp hoặc thông tin khác có đúng như bạn mong đợi hay không, giúp bạn phát hiện và sửa lỗi. |
| **Console.WriteLine()** | Console.WriteLine() là một phương thức trong C# được sử dụng để xuất một dòng văn bản kèm theo ký tự xuống dòng đến luồng đầu ra tiêu chuẩn (mặc định là cửa sổ console). Nó thường được sử dụng để hiển thị thông điệp, giá trị hoặc bất kỳ thông tin nào ra cửa sổ console. |
| **Console.Write()** | để in ra màn hình mà không kèm theo ký tự xuống dòng, bạn có thể sử dụng Console.Write() thay vì Console.WriteLine(). Phương thức Console.Write() cũng giống như Console.WriteLine() nhưng không tự động thêm ký tự xuống dòng (\n) sau mỗi lần ghi ra. |
| **Console.ReadLine()** | phương thức Console.ReadLine() được sử dụng để đọc một dòng văn bản từ luồng nhập chuẩn (standard input stream), mà trong hầu hết các trường hợp là từ bàn phím khi chương trình đang chạy trong môi trường console. |
| **INPUT** | |
| **GetKey()** | Hàm **Input.GetKey()** trong Unity được sử dụng để kiểm tra xem một phím cụ thể trên bàn phím có được nhấn xuống hay không tại thời điểm hiện tại trong game. Đây là một cách để kiểm tra trạng thái của phím trong mỗi frame của game. Hàm này trả về giá trị kiểu boolean ( true hoặc false )        Trong Unity, sử dụng KeyCode.A và "a" trong Input.GetKey() là hai cách để xác định phím A trên bàn phím, nhưng chúng có sự khác nhau nhất định:  **KeyCode.A:** Là một enum (kiểu liệt kê) trong Unity, đại diện cho mã key code của phím A trên bàn phím. Đây là một cách chính xác để xác định một phím dựa trên mã key code của nó.  **"a"** trong **GetKey():** Là việc xác định một phím dựa trên ký tự của nó. Trong trường hợp này, "a" là một chuỗi đại diện cho ký tự "a" trên bàn phím.  Input.GetKey("space") được sử dụng để kiểm tra xem người dùng có đang nhấn một phím cụ thể trên bàn phím, trong trường hợp này là phím Space. Nó trả về true liên tục trong mỗi frame trong khi người dùng đang nhấn phím đó.  Thường được sử dụng cho các hành động liên tục như di chuyển, giữ phím để tiếp tục hành động nào đó. |
| **GetButtonDown()** | Hàm **Input.GetButtonDown()** trong Unity được sử dụng để kiểm tra xem người chơi đã nhấn một nút trên bàn phím hoặc thiết bị đầu vào khác và trả về **true** trong một frame cụ thể khi nút đó được nhấn lần đầu tiên.  Input.GetButtonDown("Jump") được sử dụng để kiểm tra xem người dùng đã nhấn một phím được gán trong phần cấu hình Input (trong tab Input của Project Settings) và được gán tên là "Jump" (thường là phím để nhảy) và chỉ trả về true một lần duy nhất khi phím đó được nhấn lần đầu tiên.  Thường được sử dụng cho các hành động như nhảy, bắn, hoặc các hành động cần phản hồi từ một sự kiện chỉ khi phím được nhấn lần đầu tiên. |
| **GetComponent** | Trong Unity, **GetComponent** là một phương thức của lớp **GameObject**, cho phép bạn truy cập các thành phần (components) được gắn kèm với một đối tượng (GameObject). Điều này cho phép bạn tương tác và điều khiển các thành phần khác nhau trên một đối tượng trong game. |
| **GetAxis()** | Trong Unity, **Input.GetAxis("Horizontal")** là một phương thức trong hệ thống đầu vào của Unity được sử dụng để nhận đầu vào từ bàn phím hoặc các thiết bị điều khiển ngang (ví dụ: các nút mũi tên hoặc joystick) để điều khiển di chuyển ngang của đối tượng trong không gian 2D.  Khi bạn sử dụng Input.GetAxis("Horizontal"), nó sẽ trả về một giá trị từ -1 đến 1 tương ứng với hướng di chuyển ngang (trái hoặc phải). Giá trị -1 thường đại diện cho hướng trái, 1 là hướng phải và 0 khi không có đầu vào nào. |
| **So sánh Input.GetAxis và input.getkey** | Cả hai hàm **Input.GetAxis()** và **Input.GetKey()** đều kiểm tra trạng thái của các phím trong game, nhưng chúng có cách thức hoạt động khác nhau.  **Input.GetAxis("Horizontal")** thường được sử dụng để nhận giá trị từ các trục ngang (trái/phải) trên bàn phím hoặc các thiết bị đầu vào khác như joystick.  Hàm này trả về một giá trị liên tục từ -1 đến 1, đại diện cho trạng thái của trục ngang. Giá trị -1 khi người chơi nhấn phím trái, 1 khi nhấn phím phải và 0 khi không có phím nào được nhấn.  **GetAxis()** thường được sử dụng để điều khiển di chuyển liên tục như việc di chuyển nhân vật hoặc các đối tượng khác theo hướng ngang.  **Input.GetAxis("Horizontal")** trả về một giá trị liên tục từ -1 đến 1, thích hợp cho việc kiểm tra di chuyển liên tục như di chuyển ngang.  **Input.GetKey("space")** trả về trạng thái của một phím cụ thể, thích hợp cho các hành động không liên tục hoặc cần phản hồi từ mỗi frame khi phím được nhấn. |
| **velocity** | **velocity** trong lập trình game là một thuộc tính quan trọng được sử dụng để điều khiển chuyển động của các đối tượng trong không gian game, đặc biệt là khi làm việc với vật lý. Trong Unity, **velocity** thường được sử dụng với các thành phần vật lý như **Rigidbody** để xác định và điều chỉnh tốc độ và hướng chuyển động của đối tượng.  **Cách hoạt động và tác dụng của velocity:**  **Điều khiển chuyển động:**  velocity đại diện cho vận tốc của một đối tượng trong không gian game. Nó xác định tốc độ và hướng di chuyển của đối tượng.  Khi bạn đặt giá trị cho velocity, đối tượng sẽ di chuyển với vận tốc và hướng được chỉ định mỗi frame.  **Vật lý và di chuyển:**  Trong Unity, khi bạn gắn kèm một Rigidbody hoặc Rigidbody2D vào một GameObject, thuộc tính velocity của nó sẽ xác định cách di chuyển của đối tượng trong không gian.  Bằng cách thay đổi giá trị của velocity, bạn có thể điều chỉnh tốc độ, hướng di chuyển và thậm chí làm thay đổi quỹ đạo của đối tượng.  **Ảnh hưởng đến vật lý và chuyển động của đối tượng:**  velocity thường được sử dụng trong các tình huống liên quan đến vật lý, chẳng hạn như khi di chuyển các đối tượng trong không gian 2D hoặc 3D, áp dụng lực, hoặc xử lý va chạm với các đối tượng khác.  **Cập nhật liên tục:**  Để duy trì chuyển động của đối tượng, bạn cần cập nhật giá trị velocity mỗi frame bằng cách sử dụng các phương thức như Update() hoặc FixedUpdate() trong script của bạn.      Vận tốc không được tính như bình thường mà sẽ là theo pixel |
| **rb.velocity.y** | **rb.velocity** là một thuộc tính của đối tượng **Rigidbody2D** trong Unity. Nó trả về một Vector2, đại diện cho vận tốc hiện tại của đối tượng theo hai trục, x và y.  **rb.velocity** trong Unity là một thuộc tính của Rigidbody (hoặc Rigidbody2D trong không gian 2D) mà bạn có thể sử dụng để điều khiển vận tốc của một đối tượng trong không gian game. Nó có nhiều ứng dụng quan trọng trong việc thực hiện di chuyển, vật lý và xử lý va chạm của đối tượng. |
| **SpriteRenderer** | **SpriteRenderer** là một lớp (class) dùng để hiển thị các sprite (hình ảnh 2D) trên các đối tượng trong không gian game.  Lớp SpriteRenderer là một phần của API (Application Programming Interface) của Unity, cho phép bạn thao tác và điều khiển việc hiển thị các sprite trên các đối tượng trong trò chơi của mình thông qua mã nguồn.      - **flipX** là một thuộc tính của SpriteRenderer, cho phép bạn đảo ngược (flip) hình ảnh theo chiều ngang trên đối tượng được gắn SpriteRenderer.  Khi bạn đặt giá trị của flipX là true, nó sẽ làm đảo ngược hình ảnh theo chiều ngang của sprite được hiển thị. Nghĩa là hình ảnh sẽ được đảo ngược theo chiều ngang và xuất hiện như một hình ảnh phản chiếu qua trục đứng. Điều này thường được sử dụng để thực hiện việc đổi hướng nhân vật, chẳng hạn khi nhân vật đang di chuyển từ phải sang trái hoặc ngược lại.      - **flipY** là một thuộc tính của SpriteRenderer, cho phép bạn đảo ngược (flip) hình ảnh theo chiều dọc trên đối tượng được gắn SpriteRenderer.  Khi bạn đặt giá trị của flipX là true, nó sẽ làm đảo ngược hình ảnh theo chiều dọc của sprite được hiển thị. Nghĩa là hình ảnh sẽ được đảo ngược theo chiều dọc và xuất hiện như một hình ảnh phản chiếu qua trục ngang . Tương tự như flipX. |
| **SerializeField** | SerializeField là một thuộc tính trong Unity được sử dụng để làm cho biến private hoặc protected trở nên hiển thị trong Inspector. Trong Unity, Inspector là một cửa sổ giúp bạn chỉnh sửa các thành phần của một đối tượng, thường được sử dụng để tinh chỉnh giá trị của các biến.    Trong unity sẽ được như này để dễ chỉnh sửa hơn |
|  |  |