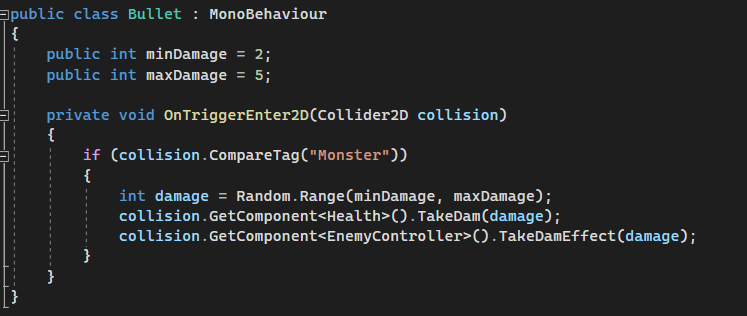
**\*Bullet:**

Hình ảnh minh họa:



-Tạo 2 biến lưu lượng sát thương nhỏ nhất và lớn nhất.

- Trong Script này, làm việc với các Collision và Trigger để xử lý va chạm của đạn với các đối tượng khác trong trò chơi. Làm sao để va chạm đạn vào quái?

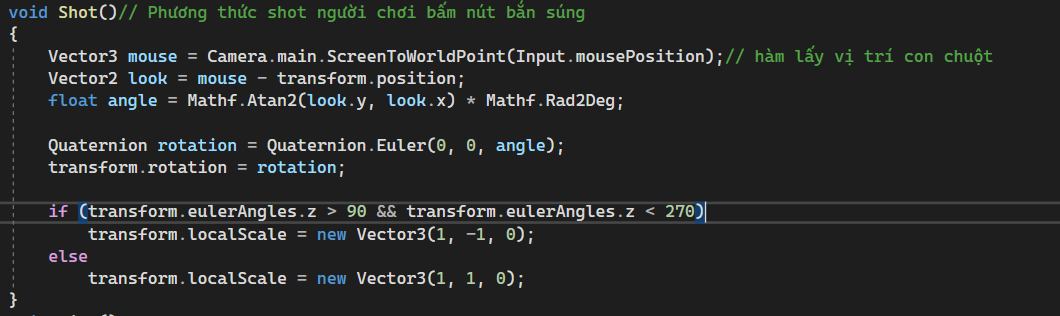
->Ta sử lý viên đạn va chạm tới cái Tag có tên "Monster" thì ta khởi tạo 1 biến gây sát thương cho biến chạy ngẫu nhiên từ sát thương nhỏ nhất tới sát thương lớn nhất. Tiếp theo sẽ cập nhật thanh máu và lưu sát thương đã gây trước đó vào hàm TakeDam.Và cuối cùng sẽ cập nhật bảng điều khiển của quái và hiển thị sát thương vào hàm TakeDamEffect.

**\*Weapon:**

**A. Trong Script này, làm việc với các Collider, Rigidbody để xác định vùng va chạm và điều khiển chuyển động của weapon**

1. Tạo 1 phương thức Shot() trong đó:

Hình ảnh minh họa:



Hình ảnh:

-> Lấy vị trí của con trỏ chuột trên màn hình và chuyển đổi thành vị trí thế giới sử dụng hàm ScreenToWorldPoint().

Hình ảnh:



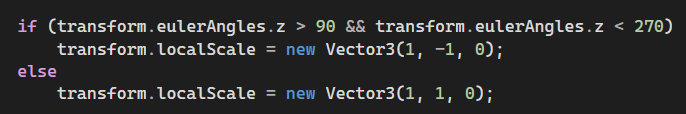
->Tính toán hướng của súng (được lưu trữ trong biến "look") bằng cách lấy vector giữa vị trí hiện tại của đối tượng và vị trí của chuột trên màn hình. Tính toán góc quay cần thiết để đối tượng quay hướng về vị trí của chuột sử dụng hàm Atan2() và chuyển đổi kết quả từ radian sang độ.

Hình ảnh:



-> Cập nhật hướng quay của đối tượng sử dụng phương thức rotation của đối tượng Transform.

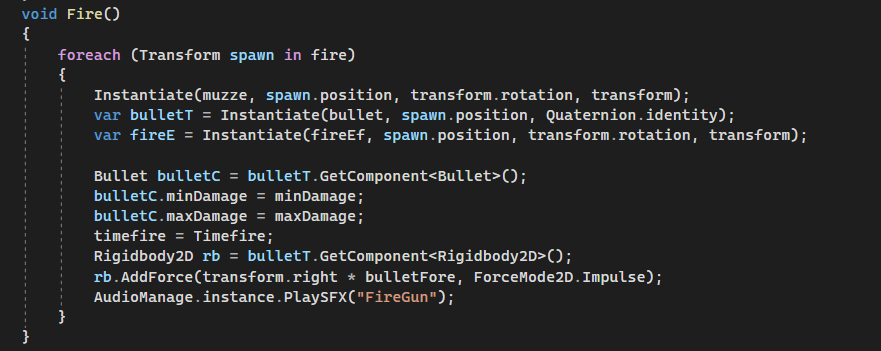
Hình ảnh:



-> Nếu góc quay của đối tượng nằm trong khoảng từ 90 đến 270 độ, thì đối tượng sẽ bị lật ngược (flip) theo trục y bằng cách thay đổi scale của đối tượng. Nếu không, đối tượng sẽ giữ nguyên scale.

2. Tạo 1 phương thức Fire():

Hình ảnh minh họa:



Hình ảnh:



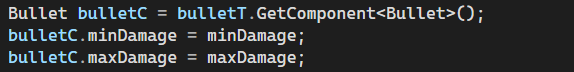
-> Đối tượng muzze được tạo ra tại vị trí spawn với hướng quay của đối tượng gốc (transform) và được đặt con trỏ bố của đối tượng mới tạo.

Hình ảnh:



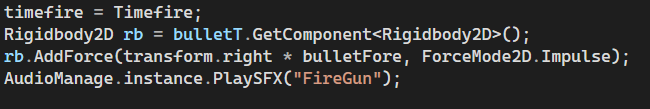
-> Đối tượng đạn (bullet) và hiệu ứng bắn (fireE) được tạo ra tại vị trí spawn với hướng quay của đối tượng gốc và được đặt con trỏ bố của đối tượng mới tạo.

Hình ảnh:



->Phương thức GetComponent<Bullet> () được sử dụng để truy cập thành phần Bullet của đối tượng đạn được tạo ra và đặt giá trị minDamage và maxDamage cho đối tượng Bullet này.

Hình ảnh:



->Thời gian giữa các lần bắn đạn được đặt bằng thời gian được cài đặt trong biến Timefire.

->Đối tượng Rigidbody2D được truy cập từ đối tượng đạn được tạo ra và được thêm lực đẩy với hướng của transform.right và một lực đẩy Impulse với giá trị được đặt trong biến bulletFore.

->Âm thanh bắn súng được phát ra sử dụng AudioManage.instance.PlaySFX("FireGun").

3. Trong phương thức Update():

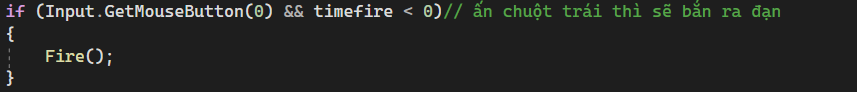
Hình ảnh:



->Phương thức Shot() được gọi để cập nhật hướng bắn của đối tượng.

->Thời gian giữa các lần bắn đạn được cập nhật bằng cách giảm thời gian đổ về 0 dựa trên thời gian thực đi qua (Time.deltaTime).

Hình ảnh:



-> Nếu người chơi ấn chuột trái và thời gian giữa các lần bắn đạn đã đạt đến 0 hoặc giá trị âm, phương thức Fire() được gọi để bắn ra các viên đạn từ nhiều vị trí khác nhau trên đối tượng.

\*BulletEnemy:

Hình ảnh minh họa:

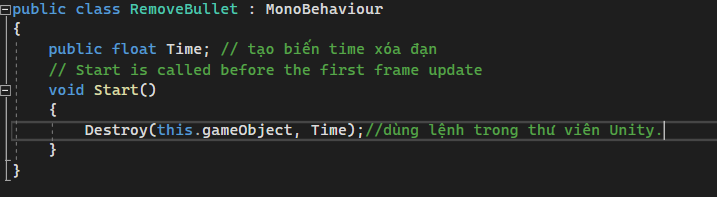


-> Cũng tương tự như phần bullet của nhân vật thay vì ta sử lý viên đạn va chạm tới cái Tag có tên "Monster" thì ta sẽ cho viên đạn va chạm tới Tag có tên “Player” và quy trình thực hiện vẫn như code bullet nhân vật. Khác ở chỗ ta sẽ cập nhật bảng điều khiển của nhân vật và hiển thị sát thương vào hàm TakeDamEffect, tiếp theo ta cho nhân vật nhận sát thương thực, và cuối cùng ta xóa đạn khi va chạm ta gọi hàm Destroy.

\*Removebullet:

-Trong script này ta chỉ viết ở phương thức Start.

Hình ảnh minh họa:

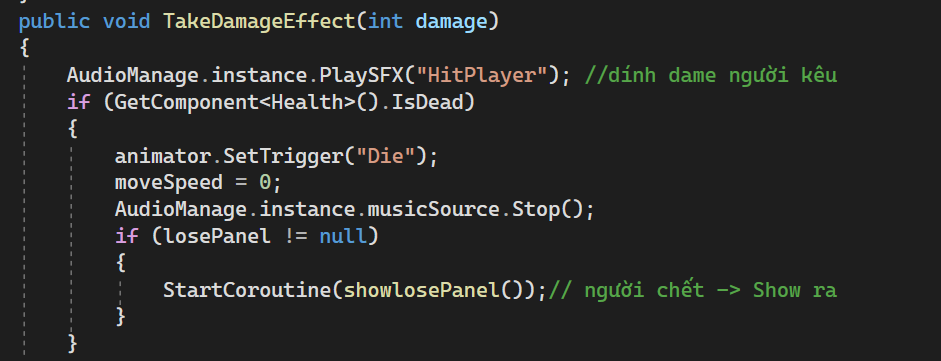


->Tạo 1 biến Time dùng để điều chỉnh thời gian xóa đạn, trong phương thức Start ta gọi lệnh Destroy ( lệnh này có sẵn trong thư viện của Unity), ta gọi lệnh Destroy để xóa gameobject ở thời điểm mà ta thiết lập sẵn.

**\*Player:**

- Tạo 1 phương thức có tên TakeDamageEffect:

Hình ảnh minh họa:

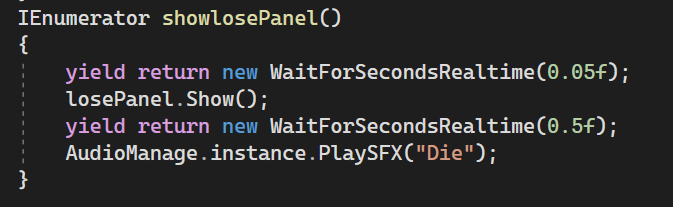


->Khi người bị dính sát thương thì hiệu ứng âm thanh được lấy từ class audiomanage được phát ra.

Nếu class Health trỏ tới IsDead = true nghĩa là người chơi chết, lúc này sẽ thực hiện Animation ( die ),

tốc độ di chuyển sẽ trở về 0, và nhạc nền sẽ tắt. Nếu như losePanel != null thì thực hiện Show ra losePanel với thời gian chờ được thực hiện trong phương thức Ienumerator.

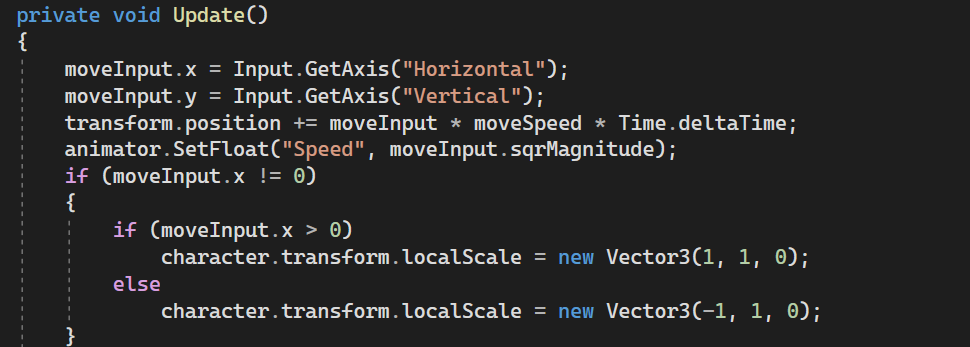
Hình ảnh minh họa:



->Trong phương thức IEnumerator showlosePanel(),trả về với thời gian đợi (cụ thể ở đây là 0.05s),

sau đó class closePanel thực hiện phương thức Show(); tiếp theo sẽ trả về với thời gian đợi(cụ thể 0.5s),

sau đó phát hiệu ứng nhạc ("Die").

-Trong phương thức Update():

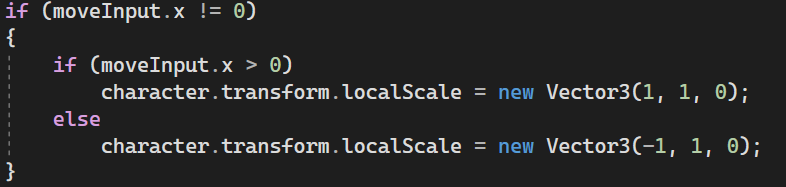
->Ở đây, hai biến "moveInput.x" và "moveInput.y" được cập nhật để lưu trữ đầu vào của người chơi về các phím di chuyển được thiết lập trong Unity gồm A, S, W, D hoặc các dấu mũi tên lên, xuống, trái, phải.

->Tính vị trí của người chơi 

+transform.position: vị trí hiện tại của người chơi được chỉ định bởi biến "moveInput" và tốc độ di chuyển "moveSpeed". Tốc độ di chuyển được nhân với delta time để đảm bảo rằng đối tượng di chuyển một cách mượt mà và không phụ thuộc vào tốc độ khung hình của trò chơi.

Hình ảnh minh họa: 

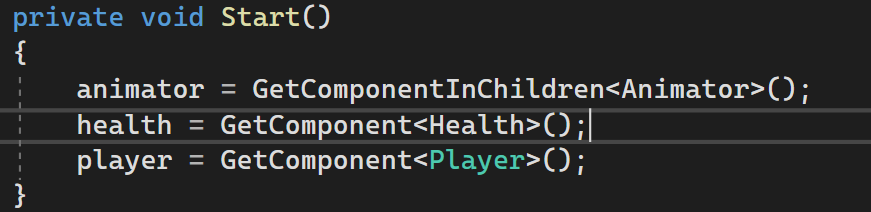
->Dòng này cập nhật giá trị của tham số "Speed" trong animator của đối tượng để điều khiển các hoạt hình liên quan đến tốc độ di chuyển. Giá trị được đặt bằng bình phương của độ dài vector "moveInput".

Hình ảnh minh họa: 

->Đoạn cuối cùng này cập nhật hướng mặt của đối tượng để nó hướng về phía người chơi điều khiển. Nếu giá trị của "moveInput.x" khác 0, đối tượng sẽ được quay về phía bên phải hoặc bên trái tùy thuộc vào giá trị của "moveInput.x". Nếu giá trị của "moveInput.x" là 0, đối tượng sẽ giữ nguyên hướng mặt của nó.

-Trong phương thức Start():

Hình ảnh minh họa:



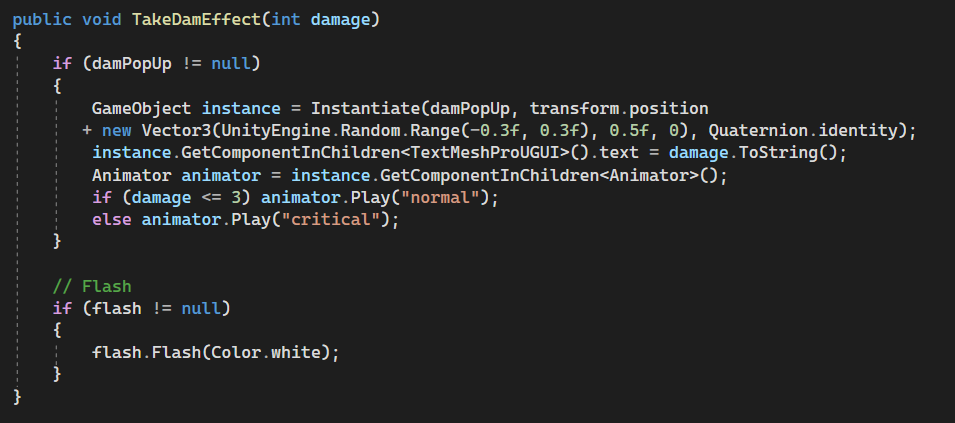
->Dòng đầu tiên tìm và lưu trữ component Animator được gắn trên đối tượng hiện tại hoặc các đối tượng con của nó.

->Dòng thứ 2 tìm và lưu trữ component Health được gắn trên đối tượng hiện tại.

->Dòng thứ 3 tìm và lưu trữ component Player được gắn trên đối tượng hiện tại.

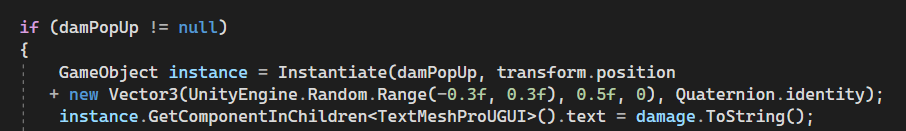
**\* PlayerController:**

Hình ảnh:



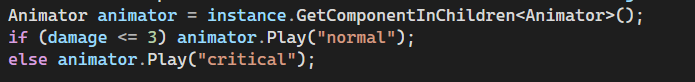
- Tạo 1 phương thức có tên TakeDamageEffect: Phương thức này nhận đầu vào một giá trị kiểu số nguyên "damage", đại diện cho số lượng sát thương mà đối tượng nhận được.

Hình ảnh:



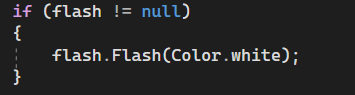
->Nếu biến "damPopUp" được gán giá trị, phương thức sẽ tạo ra một đối tượng mới tại vị trí của đối tượng hiện tại cộng thêm một vector ngẫu nhiên trong khoảng [-0.3, 0.3] trên trục x và 0.5 trên trục y, sử dụng hàm "Instantiate". Đối tượng được tạo ra sẽ có component TextMeshProUGUI, nó sẽ hiển thị số lượng sát thương "damage" lên đó.

Hình ảnh:



->Sau đó, phương thức sử dụng hàm "GetComponentInChildren" để tìm và trả về component Animator được gắn vào đối tượng mới tạo ra ở trên. Nếu giá trị của biến "damage" nhỏ hơn hoặc bằng 3 thì chạy animation có tên là "normal" trên component Animator, ngược lại thì chạy animation có tên là "critical".

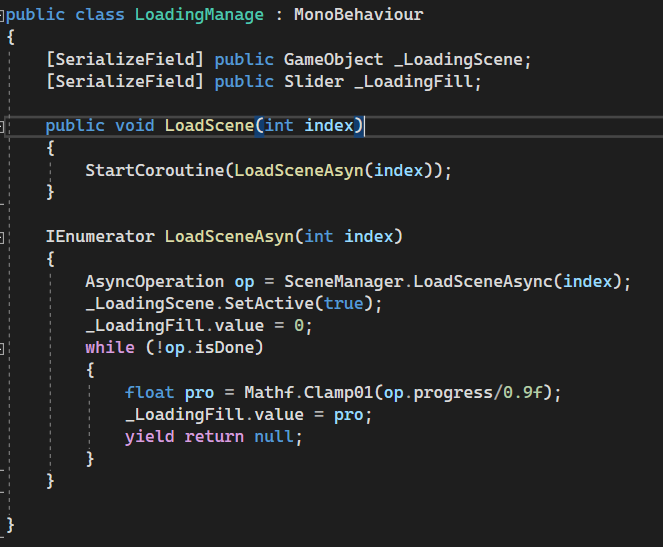
Hình ảnh:



->Sau khi thực hiện xong việc hiển thị sát thương, phương thức kiểm tra xem biến "flash" đã được gán giá trị hay chưa. Nếu biến này đã được gán giá trị thì gọi phương thức "Flash" trên đối tượng được lưu trữ trong biến "flash" để hiển thị hiệu ứng "flash" trên màn hình.

**\* LoadingManage:**

Hình ảnh:



-Phương thức IEnumerator có tên là "LoadSceneAsyn". Phương thức này nhận đầu vào một giá trị kiểu số nguyên "index".

Hình ảnh:



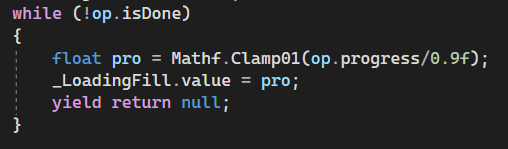
->gọi hàm SceneManager.LoadSceneAsync(index) để bắt đầu quá trình tải cảnh bất đồng bộ (asynchronously).Kết quả trả về là một đối tượng kiểu AsyncOperation, chứa thông tin về quá trình tải cảnh.

Hình ảnh:



->Tiếp theo, chúng ta gán giá trị "true" cho thuộc tính "activeSelf" của đối tượng "\_LoadingScene", và thiết lập giá trị của thanh đo "\_LoadingFill" bằng 0. Điều này có nghĩa là chúng ta đang hiển thị màn hình tải cảnh trên giao diện người dùng.

Hình ảnh:



->Vòng lặp While kiểm tra đối tượng AsyncOperation vừa tạo ra. Trong vòng lặp, chúng ta tính toán giá trị tiến trình (progress) hiện tại của quá trình tải cảnh, bằng cách chia giá trị tiến trình của AsyncOperation cho 0.9 và kẹp giá trị trong khoảng từ 0 đến 1 (phần trăm tiến trình). Gắn biến của thanh "\_LoadingFill" = pro biến được tính toán ở trên và trả về giá trị null.