using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerAttack : MonoBehaviour

{

private WeaponManager weapon\_Manager;

public float fireRate = 15f;

private float nextTimeToFire;

public float damage = 20f;

private Animator zoomCameraAnim;

private bool zoomed;

private Camera mainCam;

private GameObject crosshair;

private bool is\_Aiming;

[SerializeField]

private GameObject arrow\_Prefab, spear\_Prefab;

[SerializeField]

private Transform arrow\_Bow\_StartPosition;

void Awake()

{

weapon\_Manager = GetComponent<WeaponManager>();

zoomCameraAnim = transform.Find(Tags.LOOK\_ROOT).transform.Find(Tags.ZOOM\_CAMERA).GetComponent<Animator>();

crosshair = GameObject.FindWithTag(Tags.CROSSHAIR);

mainCam=Camera.main;

}

// Start is called before the first frame update

void Start()

{

}

// Update is called once per frame

void Update()

{

WeaponShoot();

ZoomInAndOut();

}

void WeaponShoot()

{

//if we have assault rifle

if (weapon\_Manager.GetCurrentSelectWeapon().fireType == WeaponFireType.MULTIPLE)

{

//if we press and Hold left mouse click AND

//if Time is greater than the nextTimeToFire

if (Input.GetMouseButton(0) && Time.time > nextTimeToFire)

{

nextTimeToFire = Time.time + 1f / fireRate;

weapon\_Manager.GetCurrentSelectWeapon().ShootAnimation();

BulletFired();

}

}

else

{

if (Input.GetMouseButtonDown(0))

{//handle axe

if (weapon\_Manager.GetCurrentSelectWeapon().tag == Tags.AXE\_TAG)

{

weapon\_Manager.GetCurrentSelectWeapon().ShootAnimation();

}

//handle shoot

if (weapon\_Manager.GetCurrentSelectWeapon().bulletType == WeaponBulletType.BULLET)

{

weapon\_Manager.GetCurrentSelectWeapon().ShootAnimation();

BulletFired();

}

else

{//we have an arrow or bow

if (is\_Aiming)

{

weapon\_Manager.GetCurrentSelectWeapon().ShootAnimation();

if(weapon\_Manager.GetCurrentSelectWeapon().bulletType == WeaponBulletType.ARROW)

{

ThrowArrowOrSpear(true);

// throw arrow

}

else if (weapon\_Manager.GetCurrentSelectWeapon().bulletType == WeaponBulletType.SPEAR)

{

//throw spear

ThrowArrowOrSpear(false);

}

}

}

}//if input get mouse button 0

}

}

void ZoomInAndOut()

{

//we are going to aim wit our camera on the weapon

if (weapon\_Manager.GetCurrentSelectWeapon().weapon\_Aim == WeaponAim.AIM)

{

//if we press and hold right mouse button

if (Input.GetMouseButtonDown(1))

{

zoomCameraAnim.Play(AnimationTags.ZOOM\_IN\_ANIM);

crosshair.SetActive(false);

}

//release the right button click

if (Input.GetMouseButtonUp(1))

{

zoomCameraAnim.Play(AnimationTags.ZOOM\_OUT\_ANIM);

crosshair.SetActive(true);

}

}//if we need to zoom the weapon

if (weapon\_Manager.GetCurrentSelectWeapon().weapon\_Aim == WeaponAim.SELF\_AIM)

{

if (Input.GetMouseButtonDown(1))

{

weapon\_Manager.GetCurrentSelectWeapon().Aim(true);

is\_Aiming = true;

}

if (Input.GetMouseButtonUp(1))

{

weapon\_Manager.GetCurrentSelectWeapon().Aim(false);

is\_Aiming = false;

}

}//weapon self aim

}//zoom in and out

void ThrowArrowOrSpear(bool throwArrow)

{

if (throwArrow)

{

GameObject arrow = Instantiate(arrow\_Prefab);

arrow.transform.position = arrow\_Bow\_StartPosition.position;

arrow.GetComponent<ArrowBowScript>().Launch(mainCam);

}

else

{

GameObject spear = Instantiate(spear\_Prefab);

spear.transform.position = arrow\_Bow\_StartPosition.position;

spear.GetComponent<ArrowBowScript>().Launch(mainCam);

}

}

void BulletFired() {

RaycastHit hit;

if (Physics.Raycast(mainCam.transform.position, mainCam.transform.forward, out hit)) {

if (hit.transform.tag == Tags.ENEMY\_TAG) {

hit.transform.GetComponent<HealthScript>().ApplyDamage(damage);

print("We hit: " + hit.transform.gameObject.name);

}

}

}

}//class

Giải thích code

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerAttack : MonoBehaviour

{

private WeaponManager weapon\_Manager;

public float fireRate = 15f;

private float nextTimeToFire;

public float damage = 20f;

private Animator zoomCameraAnim;

private bool zoomed;

private Camera mainCam;

private GameObject crosshair;

private bool is\_Aiming;

[SerializeField]

private GameObject arrow\_Prefab, spear\_Prefab;

[SerializeField]

private Transform arrow\_Bow\_StartPosition;

Các biến trong script PlayerAttack phục vụ nhiều mục đích khác nhau liên quan đến việc xử lý tấn công bằng vũ khí, nhắm bắn, và hiệu ứng camera trong Unity.

**weapon\_Manager**: Một thể hiện của lớp WeaponManager, có khả năng chịu trách nhiệm quản lý vũ khí của người chơi, bao gồm việc chuyển đổi giữa các vũ khí và lấy vũ khí hiện tại được chọn.

1. **fireRate**: Một giá trị kiểu float xác định tần suất mà người chơi có thể bắn vũ khí. Ví dụ, fireRate là 15f nghĩa là người chơi có thể bắn 15 phát mỗi giây.
2. **nextTimeToFire**: Một giá trị kiểu float giữ thời gian khi nào người chơi có thể bắn phát tiếp theo. Nó được sử dụng để triển khai tốc độ bắn, đảm bảo rằng vũ khí không bắn thường xuyên hơn mức cho phép bởi fireRate.
3. **damage**: Một giá trị kiểu float đại diện cho lượng sát thương mỗi phát bắn hoặc tấn công gây ra cho mục tiêu.
4. **zoomCameraAnim**: Một thành phần Animator để xử lý các hoạt ảnh zoom của camera. Nó có thể được sử dụng để tạo hiệu ứng zoom khi người chơi nhắm bắn.
5. **zoomed**: Một cờ boolean chỉ ra liệu camera hiện tại có đang được zoom hay không.
6. **mainCam**: Một tham chiếu tới camera chính trong cảnh. Nó có thể được sử dụng để điều khiển các hành vi của camera, như zoom hoặc điều chỉnh trường nhìn khi nhắm bắn.
7. **crosshair**: Một đối tượng GameObject đại diện cho phần tử giao diện ngắm bắn, giúp người chơi nhắm. Nó thường được hiển thị ở trung tâm của màn hình.
8. **is\_Aiming**: Một cờ boolean chỉ ra liệu người chơi có đang nhắm bắn hay không. Nó có thể ảnh hưởng đến các hành vi khác nhau, như zoom của camera và hiển thị giao diện ngắm bắn.
9. **arrow\_Prefab**: Một tham chiếu tới prefab được sử dụng cho mũi tên. Prefab là các GameObject đã được cấu hình sẵn có thể được tạo ra trong trò chơi.
10. **spear\_Prefab**: Một tham chiếu tới prefab được sử dụng cho giáo. Giống như prefab của mũi tên, nó là một GameObject đã được cấu hình sẵn.
11. **arrow\_Bow\_StartPosition**: Một Transform đại diện cho vị trí bắt đầu khi bắn mũi tên. Đây có thể là vị trí mà mũi tên được tạo ra khi người chơi bắn chúng.

void Awake()

{

weapon\_Manager = GetComponent<WeaponManager>();

zoomCameraAnim = transform.Find(Tags.LOOK\_ROOT).transform.Find(Tags.ZOOM\_CAMERA).GetComponent<Animator>();

crosshair = GameObject.FindWithTag(Tags.CROSSHAIR);

mainCam=Camera.main;

}

1.weapon\_Manager = GetComponent<WeaponManager>();

Dòng này lấy thành phần WeaponManager từ GameObject hiện tại mà script này được gắn vào. WeaponManager có thể là một script khác quản lý các vũ khí của người chơi.

2.zoomCameraAnim= transform.Find(Tags.LOOK\_ROOT).transform.Find(Tags.ZOOM\_CAMERA).GetComponent<Animator>();

Dòng này tìm kiếm một thành phần Animator trong cây đối tượng con của GameObject hiện tại. Đầu tiên, nó tìm đối tượng con với tên được định nghĩa trong Tags.LOOK\_ROOT, sau đó tìm tiếp đối tượng con của nó với tên trong Tags.ZOOM\_CAMERA, và cuối cùng lấy thành phần Animator từ đối tượng đó. Animator này có thể được sử dụng để điều khiển các hoạt ảnh zoom của camera.

3.crosshair = GameObject.FindWithTag(Tags.CROSSHAIR);

Dòng này tìm kiếm một đối tượng trong cảnh với tag Tags.CROSSHAIR và gán nó cho biến crosshair. crosshair có thể là một UI element giúp người chơi nhắm bắn.

4.mainCam = Camera.main;

Dòng này lấy tham chiếu tới camera chính trong cảnh và gán nó cho biến mainCam. mainCam có thể được sử dụng để điều khiển các hành vi của camera, chẳng hạn như zoom hoặc thay đổi trường nhìn khi người chơi nhắm bắn.

Kết luận:

Hàm Awake() được sử dụng để khởi tạo các biến quan trọng cho script PlayerAttack. Nó thiết lập các tham chiếu đến WeaponManager, Animator của camera zoom, đối tượng crosshair, và camera chính. Điều này đảm bảo rằng các biến này được thiết lập chính xác trước khi bất kỳ logic nào khác trong script bắt đầu chạy.

void WeaponShoot()

{

//if we have assault rifle

if (weapon\_Manager.GetCurrentSelectWeapon().fireType == WeaponFireType.MULTIPLE)

{

//if we press and Hold left mouse click AND

//if Time is greater than the nextTimeToFire

if (Input.GetMouseButton(0) && Time.time > nextTimeToFire)

{

nextTimeToFire = Time.time + 1f / fireRate;

weapon\_Manager.GetCurrentSelectWeapon().ShootAnimation();

BulletFired();

}

}

else

{

if (Input.GetMouseButtonDown(0))

{//handle axe

if (weapon\_Manager.GetCurrentSelectWeapon().tag == Tags.AXE\_TAG)

{

weapon\_Manager.GetCurrentSelectWeapon().ShootAnimation();

}

//handle shoot

if (weapon\_Manager.GetCurrentSelectWeapon().bulletType == WeaponBulletType.BULLET)

{

weapon\_Manager.GetCurrentSelectWeapon().ShootAnimation();

BulletFired();

}

else

{//we have an arrow or bow

if (is\_Aiming)

{

weapon\_Manager.GetCurrentSelectWeapon().ShootAnimation();

if(weapon\_Manager.GetCurrentSelectWeapon().bulletType == WeaponBulletType.ARROW)

{

ThrowArrowOrSpear(true);

// throw arrow

}

else if (weapon\_Manager.GetCurrentSelectWeapon().bulletType == WeaponBulletType.SPEAR)

{

//throw spear

ThrowArrowOrSpear(false);

}

}

}

}//if input get mouse button 0

}

}

 **Kiểm tra loại vũ khí**:

* Nếu vũ khí hiện tại là súng trường tấn công (WeaponFireType.MULTIPLE):
  + Nếu người chơi nhấn và giữ chuột trái (Input.GetMouseButton(0)) và thời gian hiện tại (Time.time) lớn hơn nextTimeToFire:
    - Cập nhật nextTimeToFire để giới hạn tốc độ bắn theo fireRate.
    - Gọi hàm ShootAnimation() để phát hoạt ảnh bắn súng.
    - Gọi hàm BulletFired() để thực hiện hành động bắn đạn.

 **Kiểm tra các loại vũ khí khác**:

* Nếu người chơi nhấn chuột trái (Input.GetMouseButtonDown(0)):
  + Nếu vũ khí hiện tại là rìu (Tags.AXE\_TAG):
    - Gọi hàm ShootAnimation() để phát hoạt ảnh tấn công bằng rìu.
  + Nếu vũ khí hiện tại sử dụng đạn (WeaponBulletType.BULLET):
    - Gọi hàm ShootAnimation() để phát hoạt ảnh bắn súng.
    - Gọi hàm BulletFired() để thực hiện hành động bắn đạn.
  + Nếu vũ khí hiện tại là cung tên hoặc giáo (WeaponBulletType.ARROW hoặc WeaponBulletType.SPEAR):
    - Nếu người chơi đang nhắm bắn (is\_Aiming):
      * Gọi hàm ShootAnimation() để phát hoạt ảnh bắn cung hoặc ném giáo.
      * Nếu vũ khí là cung tên (WeaponBulletType.ARROW):
        + Gọi hàm ThrowArrowOrSpear(true) để thực hiện hành động ném tên.
      * Nếu vũ khí là giáo (WeaponBulletType.SPEAR):
        + Gọi hàm ThrowArrowOrSpear(false) để thực hiện hành động ném giáo.

Kết luận:

Hàm WeaponShoot() xử lý logic bắn hoặc ném vũ khí dựa trên loại vũ khí và loại đạn hiện tại, cũng như trạng thái của các nút chuột và thời gian. Nó kiểm tra và gọi các hoạt ảnh bắn hoặc ném tương ứng và thực hiện hành động bắn đạn, ném tên, hoặc ném giáo.

void ZoomInAndOut()

{

//we are going to aim wit our camera on the weapon

if (weapon\_Manager.GetCurrentSelectWeapon().weapon\_Aim == WeaponAim.AIM)

{

//if we press and hold right mouse button

if (Input.GetMouseButtonDown(1))

{

zoomCameraAnim.Play(AnimationTags.ZOOM\_IN\_ANIM);

crosshair.SetActive(false);

}

//release the right button click

if (Input.GetMouseButtonUp(1))

{

zoomCameraAnim.Play(AnimationTags.ZOOM\_OUT\_ANIM);

crosshair.SetActive(true);

}

}//if we need to zoom the weapon

if (weapon\_Manager.GetCurrentSelectWeapon().weapon\_Aim == WeaponAim.SELF\_AIM)

{

if (Input.GetMouseButtonDown(1))

{

weapon\_Manager.GetCurrentSelectWeapon().Aim(true);

is\_Aiming = true;

}

if (Input.GetMouseButtonUp(1))

{

weapon\_Manager.GetCurrentSelectWeapon().Aim(false);

is\_Aiming = false;

}

}//weapon self aim

}//zoom in and out

 **Zoom camera với vũ khí**:

* **Kiểm tra nếu vũ khí hiện tại yêu cầu nhắm bằng camera (WeaponAim.AIM)**:
  + Nếu người chơi nhấn và giữ nút chuột phải (Input.GetMouseButtonDown(1)):
    - Phát hoạt ảnh zoom in (zoomCameraAnim.Play(AnimationTags.ZOOM\_IN\_ANIM)).
    - Ẩn crosshair (crosshair.SetActive(false)).
  + Nếu người chơi thả nút chuột phải (Input.GetMouseButtonUp(1)):
    - Phát hoạt ảnh zoom out (zoomCameraAnim.Play(AnimationTags.ZOOM\_OUT\_ANIM)).
    - Hiển thị lại crosshair (crosshair.SetActive(true)).

 **Tự nhắm với vũ khí**:

* **Kiểm tra nếu vũ khí hiện tại yêu cầu tự nhắm (WeaponAim.SELF\_AIM)**:
  + Nếu người chơi nhấn và giữ nút chuột phải (Input.GetMouseButtonDown(1)):
    - Gọi hàm Aim(true) của vũ khí hiện tại để bật chế độ nhắm (weapon\_Manager.GetCurrentSelectWeapon().Aim(true)).
    - Đặt cờ is\_Aiming thành true.
  + Nếu người chơi thả nút chuột phải (Input.GetMouseButtonUp(1)):
    - Gọi hàm Aim(false) của vũ khí hiện tại để tắt chế độ nhắm (weapon\_Manager.GetCurrentSelectWeapon().Aim(false)).
    - Đặt cờ is\_Aiming thành false.

Kết luận:

Hàm ZoomInAndOut() xử lý logic khi người chơi nhấn hoặc thả nút chuột phải để nhắm bắn. Nếu vũ khí yêu cầu zoom bằng camera, nó sẽ phát các hoạt ảnh zoom in/out và ẩn/hiện crosshair. Nếu vũ khí yêu cầu tự nhắm, nó sẽ gọi các hàm tương ứng để bật/tắt chế độ nhắm và cập nhật cờ is\_Aiming.

void ThrowArrowOrSpear(bool throwArrow)

{

if (throwArrow)

{

GameObject arrow = Instantiate(arrow\_Prefab);

arrow.transform.position = arrow\_Bow\_StartPosition.position;

arrow.GetComponent<ArrowBowScript>().Launch(mainCam);

}

else

{

GameObject spear = Instantiate(spear\_Prefab);

spear.transform.position = arrow\_Bow\_StartPosition.position;

spear.GetComponent<ArrowBowScript>().Launch(mainCam);

}

}

 **Tham số đầu vào throwArrow**:

* Biến throwArrow là một biến boolean được truyền vào hàm. Nó xác định xem đối tượng cần tạo là một mũi tên (nếu true) hay một giáo (nếu false).

 **Nếu throwArrow là true**:

* **Tạo một mũi tên**:
  + GameObject arrow = Instantiate(arrow\_Prefab);
    - Tạo một đối tượng mũi tên mới từ prefab arrow\_Prefab.
  + arrow.transform.position = arrow\_Bow\_StartPosition.position;
    - Đặt vị trí của mũi tên mới tạo tại vị trí bắt đầu của cung tên (arrow\_Bow\_StartPosition.position).
  + arrow.GetComponent<ArrowBowScript>().Launch(mainCam);
    - Lấy thành phần ArrowBowScript của đối tượng mũi tên và gọi phương thức Launch(), truyền vào camera chính (mainCam) để bắn mũi tên.

 **Nếu throwArrow là false**:

* **Tạo một giáo**:
  + GameObject spear = Instantiate(spear\_Prefab);
    - Tạo một đối tượng giáo mới từ prefab spear\_Prefab.
  + spear.transform.position = arrow\_Bow\_StartPosition.position;
    - Đặt vị trí của giáo mới tạo tại vị trí bắt đầu của cung tên (arrow\_Bow\_StartPosition.position).
  + spear.GetComponent<ArrowBowScript>().Launch(mainCam);
    - Lấy thành phần ArrowBowScript của đối tượng giáo và gọi phương thức Launch(), truyền vào camera chính (mainCam) để ném giáo.

Kết luận:

Hàm ThrowArrowOrSpear() tạo và bắn ra một mũi tên hoặc giáo tùy thuộc vào giá trị của tham số throwArrow. Nếu throwArrow là true, nó sẽ tạo và bắn một mũi tên. Nếu throwArrow là false, nó sẽ tạo và ném một giáo. Cả hai đối tượng đều được tạo tại vị trí bắt đầu của cung tên và được bắn hoặc ném bằng cách gọi phương thức Launch() trên thành phần ArrowBowScript, với tham số là camera chính.

void BulletFired() {

RaycastHit hit;

if (Physics.Raycast(mainCam.transform.position, mainCam.transform.forward, out hit)) {

if (hit.transform.tag == Tags.ENEMY\_TAG) {

hit.transform.GetComponent<HealthScript>().ApplyDamage(damage);

print("We hit: " + hit.transform.gameObject.name);

}

}

}

}//class

 **RaycastHit hit**:

* RaycastHit là một cấu trúc chứa thông tin về việc đạn đã trúng vào đối tượng nào khi thực hiện raycast.

 **Physics.Raycast(mainCam.transform.position, mainCam.transform.forward, out hit)**:

* Phương thức này tạo ra một tia ray từ vị trí của camera chính (mainCam.transform.position) và hướng về phía trước của camera (mainCam.transform.forward).
* Nếu tia ray trúng một đối tượng nào đó, thông tin về đối tượng đó được lưu trữ trong hit.

 **if (hit.transform.tag == Tags.ENEMY\_TAG)**:

* Kiểm tra xem đối tượng mà tia ray trúng có tag là ENEMY\_TAG hay không. Tag này xác định rằng đối tượng đó là kẻ thù.

 **hit.transform.GetComponent<HealthScript>().ApplyDamage(damage)**:

* Nếu đối tượng trúng tia ray là kẻ thù, phương thức ApplyDamage(damage) từ thành phần HealthScript của đối tượng đó được gọi. Phương thức này sẽ áp dụng sát thương cho kẻ thù.
* damage là một biến float được khai báo trong script PlayerAttack và đại diện cho lượng sát thương mà đạn gây ra.

 **print("We hit: " + hit.transform.gameObject.name)**:

* Dòng này in ra tên của đối tượng bị trúng đạn trong console của Unity, giúp kiểm tra xem đối tượng nào đã bị trúng.