using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MouseLook : MonoBehaviour

{

[SerializeField]

private Transform playerRoot, lookRoot;//khai bao phan nhin cho nhan vat

// Start is called before the first frame update

[SerializeField]

private bool invert;

[SerializeField]

private bool can\_Unlock=true;//de nhin thay crosshair

[SerializeField]

private float sensivity = 5f;

[SerializeField]

private int smooth\_Steps = 10;

[SerializeField]

private float smooth\_Weight = 0.4f;

[SerializeField]

private float roll\_Angle = 10f;

private float roll\_Speed = 3f;

[SerializeField]

private Vector2 default\_Look\_Limits = new Vector2(-70f,80f);

[SerializeField]

private Vector2 look\_Angles;

private Vector2 current\_Mouse\_Look;

private Vector2 smooth\_Move;

private float current\_Roll\_Angle;

private int last\_Look\_Frame;

void Start()

{

Cursor.lockState=CursorLockMode.Locked;

}

// Update is called once per frame

void Update()

{

LockAndUnlockCursor();

if (Cursor.lockState == CursorLockMode.Locked)

{

LookAround();

}

}

void LockAndUnlockCursor()

{

if (Input.GetKeyDown(KeyCode.Escape))

{

if (Cursor.lockState == CursorLockMode.Locked)

{

Cursor.lockState = CursorLockMode.None;

}

else

{

Cursor.lockState = CursorLockMode.Locked;

Cursor.visible=false;

}

}

}//lock and unlock

void LookAround()

{

current\_Mouse\_Look = new Vector2(Input.GetAxis(MouseAxis.MOUSE\_Y), Input.GetAxis(MouseAxis.MOUSE\_X));

look\_Angles.x += current\_Mouse\_Look.x \* sensivity \* (invert ? 1f : -1f);

look\_Angles.y += current\_Mouse\_Look.y \* sensivity;

//look\_Angles.y = look\_Angles.y + current\_Mouse\_Look.y\*sensivity

look\_Angles.x = Mathf.Clamp(look\_Angles.x, default\_Look\_Limits.x, default\_Look\_Limits.y);//stop the mouse moving overcome the limit of x and y using mathf clamp

current\_Roll\_Angle = Mathf.Lerp(current\_Roll\_Angle, Input.GetAxisRaw(MouseAxis.MOUSE\_X) \* roll\_Angle, Time.deltaTime \* roll\_Speed);

lookRoot.localRotation=Quaternion.Euler(look\_Angles.x,0f,current\_Roll\_Angle);

playerRoot.localRotation=Quaternion.Euler(0f,look\_Angles.y,0f);

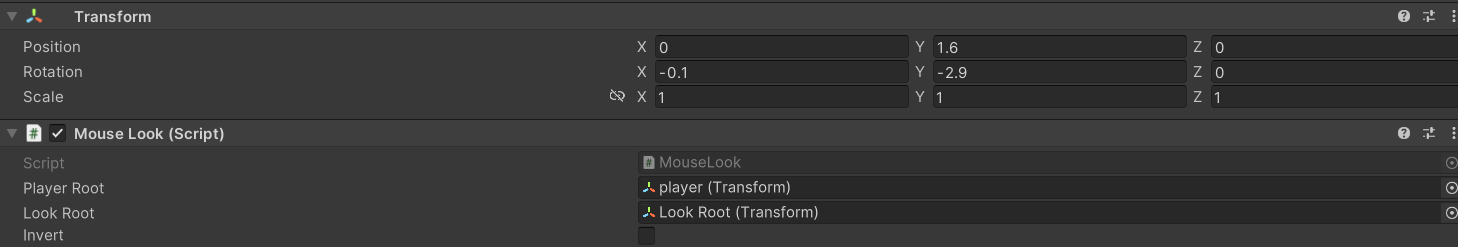
}//look around

}//class lock and unlock

[SerializeField]

private Transform playerRoot, lookRoot;//khai bao phan nhin cho nhan vat

=>khai báo playerRoot và lookRoot để có thể bỏ các object Look Root và Player vào



[SerializeField]

private bool can\_Unlock=true;//de nhin thay crosshair

=>dùng để có thể nhìn thấy crosshair sau này sẽ áp dụng

[SerializeField]

private float sensivity = 5f;

[SerializeField]

private int smooth\_Steps = 10;

[SerializeField]

private float smooth\_Weight = 0.4f;

[SerializeField]

private float roll\_Angle = 10f;

private float roll\_Speed = 3f;

[SerializeField]

private Vector2 default\_Look\_Limits = new Vector2(-70f,80f);

[SerializeField]

private Vector2 look\_Angles;

private Vector2 current\_Mouse\_Look;

private Vector2 smooth\_Move;

private float current\_Roll\_Angle;

private int last\_Look\_Frame;

=>Khai báo các biến private để dùng cho các hàm sau này

void Start()

{

Cursor.lockState=CursorLockMode.Locked;

}

=>sử dụng để khóa con trỏ chuột vào giữa màn hình trò chơi và ẩn nó, tạo ra trải nghiệm chơi trò chơi mượt mà hơn, đặc biệt là trong các trò chơi 3D hoặc game bắn súng góc nhìn thứ nhất (FPS).

Trong ngữ cảnh này:

* Cursor.lockState là thuộc tính dùng để thiết lập trạng thái của con trỏ chuột.
* CursorLockMode.Locked là một giá trị enum trong CursorLockMode chỉ định rằng con trỏ sẽ bị khóa ở giữa màn hình và sẽ không di chuyển khỏi vị trí đó.

Khi thuộc tính này được đặt thành CursorLockMode.Locked, con trỏ chuột sẽ bị khóa vào giữa màn hình và bị ẩn. Điều này thường được sử dụng để đảm bảo rằng chuyển động của chuột sẽ chỉ ảnh hưởng đến hướng nhìn của nhân vật trong trò chơi mà không làm mất sự tập trung của người chơi khỏi trò chơi do việc con trỏ chuột di chuyển ra khỏi cửa sổ trò chơi.

void LockAndUnlockCursor()

{

if (Input.GetKeyDown(KeyCode.Escape))

{

if (Cursor.lockState == CursorLockMode.Locked)

{

Cursor.lockState = CursorLockMode.None;

}

else

{

Cursor.lockState = CursorLockMode.Locked;

Cursor.visible=false;

}

}

if (Input.GetKeyDown(KeyCode.Escape))

**=>** **Phát hiện phím "Escape" được nhấn thì mã bên trong sẽ được thực thi**

if (Cursor.lockState == CursorLockMode.Locked)

=>Kiểm tra xem con trỏ hiện đang bị khóa hay không.

Cursor.lockState = CursorLockMode.None;

Nếu con trỏ đang bị khóa, nó sẽ được mở khóa (CursorLockMode.None) và hiển thị lại.

Cursor.lockState = CursorLockMode.Locked;

Cursor.visible=false;

Nếu con trỏ không bị khóa, nó sẽ bị khóa (CursorLockMode.Locked) và bị ẩn.

void Update()

{

LockAndUnlockCursor();

if (Cursor.lockState == CursorLockMode.Locked)

{

LookAround();

}

}

=>nếu mà con trỏ bị khóa thì sẽ thực hiện hàm LookAround();

void LookAround()

{

current\_Mouse\_Look = new Vector2(Input.GetAxis(MouseAxis.MOUSE\_Y), Input.GetAxis(MouseAxis.MOUSE\_X));

look\_Angles.x += current\_Mouse\_Look.x \* sensivity \* (invert ? 1f : -1f);

look\_Angles.y += current\_Mouse\_Look.y \* sensivity;

//look\_Angles.y = look\_Angles.y + current\_Mouse\_Look.y\*sensivity

look\_Angles.x = Mathf.Clamp(look\_Angles.x, default\_Look\_Limits.x, default\_Look\_Limits.y);//stop the mouse moving overcome the limit of x and y using mathf clamp

current\_Roll\_Angle = Mathf.Lerp(current\_Roll\_Angle, Input.GetAxisRaw(MouseAxis.MOUSE\_X) \* roll\_Angle, Time.deltaTime \* roll\_Speed);

lookRoot.localRotation=Quaternion.Euler(look\_Angles.x,0f,current\_Roll\_Angle);

playerRoot.localRotation=Quaternion.Euler(0f,look\_Angles.y,0f);

}//look around

}//class lock and unlock

Khai báo MouseAxis ở TagHolder

public class MouseAxis

{

public const string MOUSE\_X = "Mouse X";

public const string MOUSE\_Y = "Mouse Y";

}

=>giúp cho gọi biến tránh sai sót

void LookAround()

{

current\_Mouse\_Look = new Vector2(Input.GetAxis(MouseAxis.MOUSE\_Y), Input.GetAxis(MouseAxis.MOUSE\_X));

lấy dữ liệu đầu vào từ chuột và lưu trữ nó vào biến current\_Mouse\_Look dưới dạng một vector 2D (Vector2).

 **Input.GetAxis**:

* Input.GetAxis là một phương thức trong Unity dùng để lấy giá trị đầu vào từ các trục (axes) được cấu hình trong Unity's Input Manager.
* MouseAxis.MOUSE\_Y và MouseAxis.MOUSE\_X là các trục đầu vào được định nghĩa cho chuột. Giá trị trả về của Input.GetAxis sẽ nằm trong khoảng từ -1 đến 1, tùy thuộc vào sự di chuyển của chuột.

 **new Vector2(...)**:

* new Vector2(...) tạo ra một đối tượng Vector2 mới với hai giá trị trục Y và X tương ứng với đầu vào từ chuột.
* Vector2 là một cấu trúc dữ liệu trong Unity dùng để biểu diễn một vector 2D với hai thành phần x và y.

 **current\_Mouse\_Look**:

* current\_Mouse\_Look là một biến (có thể là biến thành viên của một lớp hoặc biến cục bộ trong một phương thức) lưu trữ giá trị đầu vào từ chuột dưới dạng một vector 2D.

=>đoạn code trên lấy giá trị đầu vào từ chuyển động của chuột theo hai trục X và Y, rồi lưu trữ chúng vào một vector 2D để có thể sử dụng sau này, chẳng hạn để xoay camera hoặc nhân vật dựa trên chuyển động của chuột.

look\_Angles.x += current\_Mouse\_Look.x \* sensivity \* (invert ? 1f : -1f);

look\_Angles.y += current\_Mouse\_Look.y \* sensivity;

=> ập nhật góc nhìn của nhân vật trong trò chơi dựa trên đầu vào từ chuột. Các giá trị đầu vào từ chuột được nhân với độ nhạy (sensitivity) để điều chỉnh tốc độ xoay của góc nhìn, và có thể bị đảo ngược tùy thuộc vào cài đặt.

look\_Angles.x += current\_Mouse\_Look.x \* sensitivity \* (invert ? 1f : -1f);

* Giá trị current\_Mouse\_Look.x (chuyển động theo trục Y của chuột) được nhân với sensitivity.
* Nếu invert là true, giá trị sẽ không thay đổi (\* 1f). Nếu invert là false, giá trị sẽ bị đảo ngược (\* -1f).
* Giá trị này sau đó được cộng vào look\_Angles.x, cập nhật góc nhìn theo trục X của nhân vật.

look\_Angles.y += current\_Mouse\_Look.y \* sensitivity;

* Giá trị current\_Mouse\_Look.y (chuyển động theo trục X của chuột) được nhân với sensitivity.
* Giá trị này sau đó được cộng vào look\_Angles.y, cập nhật góc nhìn theo trục Y của nhân vật.

=>Các giá trị đầu vào từ chuột được sử dụng để cập nhật các thành phần x và y của look\_Angles, từ đó thay đổi hướng nhìn của nhân vật.

look\_Angles.x = Mathf.Clamp(look\_Angles.x, default\_Look\_Limits.x, default\_Look\_Limits.y);//stop the mouse moving overcome the limit of x and y using mathf clamp

Đoạn code này thực hiện chức năng giới hạn giá trị của look\_Angles.x trong một phạm vi cụ thể, được xác định bởi default\_Look\_Limits.x và default\_Look\_Limits.y

look\_Angles.x: Đây là giá trị của góc nhìn theo trục x. Có thể là góc nghiêng lên/xuống của camera hoặc nhân vật trong trò chơi.

Mathf.Clamp(value, min, max): Đây là một hàm của Unity trong lớp Mathf, có chức năng giới hạn giá trị của value trong khoảng từ min đến max. Nếu value nhỏ hơn min, hàm sẽ trả về min. Nếu value lớn hơn max, hàm sẽ trả về max. Nếu value nằm trong khoảng từ min đến max, hàm sẽ trả về chính giá trị của value.

look\_Angles.x = Mathf.Clamp(look\_Angles.x, default\_Look\_Limits.x, default\_Look\_Limits.y); có nghĩa là:

* Giới hạn giá trị của look\_Angles.x trong khoảng từ default\_Look\_Limits.x đến default\_Look\_Limits.y.
* Nếu look\_Angles.x nhỏ hơn default\_Look\_Limits.x, nó sẽ được đặt lại thành default\_Look\_Limits.x.
* Nếu look\_Angles.x lớn hơn default\_Look\_Limits.y, nó sẽ được đặt lại thành default\_Look\_Limits.y.
* Nếu look\_Angles.x nằm trong khoảng từ default\_Look\_Limits.x đến default\_Look\_Limits.y, giá trị của nó sẽ không thay đổi.

Điều này giúp kiểm soát góc nhìn của camera hoặc nhân vật trong một phạm vi nhất định, tránh việc góc nhìn bị xoay quá mức.

current\_Roll\_Angle = Mathf.Lerp(current\_Roll\_Angle, Input.GetAxisRaw(MouseAxis.MOUSE\_X) \* roll\_Angle, Time.deltaTime \* roll\_Speed);

Dòng mã này cũng thuộc ngôn ngữ lập trình C# và được sử dụng trong Unity để tính toán và làm mượt (interpolate) giá trị current\_Roll\_Angle dựa trên đầu vào từ trục x của chuột (MouseAxis.MOUSE\_X).

 Mathf.Lerp(a, b, t): Đây là một hàm trong lớp Mathf của Unity, thực hiện phép nội suy tuyến tính giữa hai giá trị a và b dựa trên một hệ số t. Nếu t bằng 0, hàm sẽ trả về a. Nếu t bằng 1, hàm sẽ trả về b. Nếu t nằm giữa 0 và 1, hàm sẽ trả về giá trị giữa a và b, tỷ lệ với t.

 current\_Roll\_Angle: Giá trị hiện tại của góc xoay (roll angle).

 Input.GetAxisRaw(MouseAxis.MOUSE\_X): Lấy giá trị đầu vào thô từ trục x của chuột. Đây là giá trị không được làm mượt, đại diện cho chuyển động của chuột theo trục x.

 roll\_Angle: Một hằng số hoặc biến lưu giá trị góc xoay mong muốn, xác định mức độ ảnh hưởng của chuyển động chuột đến góc xoay.

 Time.deltaTime: Thời gian trôi qua kể từ khung hình trước đó, giúp chuẩn hóa phép nội suy theo thời gian thực để đảm bảo sự mượt mà không phụ thuộc vào tốc độ khung hình.

 roll\_Speed: Tốc độ làm mượt của góc xoay.

Ghép tất cả lại với nhau, dòng mã current\_Roll\_Angle = Mathf.Lerp(current\_Roll\_Angle, Input.GetAxisRaw(MouseAxis.MOUSE\_X) \* roll\_Angle, Time.deltaTime \* roll\_Speed); có nghĩa là:

1. Lấy giá trị đầu vào từ trục x của chuột (Input.GetAxisRaw(MouseAxis.MOUSE\_X)).
2. Nhân giá trị này với roll\_Angle để xác định giá trị đích của góc xoay.
3. Sử dụng hàm Mathf.Lerp để nội suy tuyến tính từ giá trị hiện tại của góc xoay (current\_Roll\_Angle) đến giá trị đích, với hệ số nội suy là Time.deltaTime \* roll\_Speed.

Điều này làm cho current\_Roll\_Angle dần dần tiến tới giá trị mới dựa trên chuyển động của chuột, với tốc độ được điều chỉnh bởi roll\_Speed, đảm bảo sự thay đổi mượt mà theo thời gian.

lookRoot.localRotation=Quaternion.Euler(look\_Angles.x,0f,current\_Roll\_Angle);

playerRoot.localRotation=Quaternion.Euler(0f,look\_Angles.y,0f);

**lookRoot.localRotation = Quaternion.Euler(look\_Angles.x, 0f, current\_Roll\_Angle);**

* lookRoot: Đây có thể là một biến đại diện cho một đối tượng (GameObject) trong Unity, thường là một Transform hoặc một phần của camera.
* localRotation: Đây là thuộc tính của Transform, xác định góc xoay của đối tượng trong không gian cục bộ của nó (relative to its parent).
* Quaternion.Euler(x, y, z): Đây là một hàm tạo của lớp Quaternion trong Unity, dùng để tạo một quaternion từ các góc Euler (đo bằng độ). Quaternion được sử dụng để biểu diễn quay (rotations) một cách mượt mà và tránh hiện tượng khóa gimbal.

Trong dòng mã này:

* look\_Angles.x: Đây là góc xoay quanh trục x (pitch).
* 0f: Không có xoay quanh trục y.
* current\_Roll\_Angle: Đây là góc xoay quanh trục z (roll).

Vì vậy, lookRoot.localRotation = Quaternion.Euler(look\_Angles.x, 0f, current\_Roll\_Angle); có nghĩa là thiết lập góc xoay cục bộ của lookRoot bằng một quaternion được tạo ra từ các góc Euler: look\_Angles.x cho trục x, 0 cho trục y, và current\_Roll\_Angle cho trục z.

**=>** Thiết lập góc xoay cục bộ của lookRoot (có thể là camera hoặc một phần đầu của nhân vật) dựa trên góc nhìn theo trục x và góc xoay quanh trục z (roll).

**playerRoot.localRotation = Quaternion.Euler(0f, look\_Angles.y, 0f);**

* playerRoot: Đây có thể là một biến đại diện cho một đối tượng (GameObject) khác trong Unity, thường là phần thân của người chơi (player).

Trong dòng mã này:

* 0f: Không có xoay quanh trục x.
* look\_Angles.y: Đây là góc xoay quanh trục y (yaw).
* 0f: Không có xoay quanh trục z.

Vì vậy, playerRoot.localRotation = Quaternion.Euler(0f, look\_Angles.y, 0f); có nghĩa là thiết lập góc xoay cục bộ của playerRoot bằng một quaternion được tạo ra từ các góc Euler: 0 cho trục x, look\_Angles.y cho trục y, và 0 cho trục z.

**=>**Thiết lập góc xoay cục bộ của playerRoot (có thể là thân của nhân vật) dựa trên góc nhìn theo trục y (yaw).

Điều này giúp tách biệt chuyển động của đầu (hoặc camera) và thân của nhân vật, đảm bảo rằng camera có thể xoay tự do trong khi thân của nhân vật chỉ xoay theo trục y.

private Vector2 default\_Look\_Limits = new Vector2(-70f,80f);

=>giúp cho khi nhân vật cúi xuống, ngước lên hoặc quay sang phải không vượt quá x=-70f và y=80f