Documento da lógica – Processo 2

**Pegar arma no chão – 01**

O jogador ao andar pelo cenário, irá encontrar a arma (única arma) jogada no chão, podendo pegá-la ao chegar perto (aparecendo um botão ao lado ou encima da arma). Quando pega, o objeto (arma) irá ser armazenada em um Array (no momento pois não teremos ainda o banco de dados), mas quando pega podemos ter duas opções:

1 – Pegou a arma, troca o Sprite com o personagem segurando a arma, ao mesmo tempo as animações do personagem segurando a arma estarão ativas no momento.

2 – O jogador tem a escolha de segurar a arma ou não, ou seja, podemos ter um botão de troca da arma (troca para pegar a arma ou guardar a arma)

Exemplo de um código:

using UnityEngine;

public class Player : MonoBehaviour

{

public Transform weaponSlot; // Referência ao local onde a arma será guardada

public KeyCode pickupKey = KeyCode.E; // Tecla para pegar a arma

private GameObject[] weapons; // Array para armazenar as armas coletadas

private int currentWeaponIndex = -1; // Índice da arma atualmente equipada

private void Start()

{

weapons = new GameObject[5]; // Definindo o tamanho do array de armas

}

private void Update()

{

if (Input.GetKeyDown(pickupKey))

{

// Verifica se há uma arma próxima para pegar

Collider2D weaponCollider = Physics2D.OverlapCircle(transform.position, 1f, LayerMask.GetMask("Weapon"));

if (weaponCollider != null)

{

// Obtém a referência para o GameObject da arma

GameObject weaponObject = weaponCollider.gameObject;

// Verifica se já possui essa arma no array

if (!ArrayContainsWeapon(weaponObject))

{

// Adiciona a arma no array

AddWeaponToArray(weaponObject);

// Desativa o objeto da arma

weaponObject.SetActive(false);

Debug.Log("Arma coletada!");

// Equipa automaticamente a nova arma

EquipWeapon(weaponObject);

}

}

}

}

private void AddWeaponToArray(GameObject weapon)

{

// Procura o próximo índice vazio no array de armas

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

{

if (weapons[i] == null)

{

weapons[i] = weapon;

return;

}

}

}

private bool ArrayContainsWeapon(GameObject weapon)

{

// Verifica se a arma já está presente no array

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

{

if (weapons[i] == weapon)

{

return true;

}

}

return false;

}

private void EquipWeapon(GameObject weapon)

{

// Desativa a arma atualmente equipada

if (currentWeaponIndex >= 0 && currentWeaponIndex < weapons.Length && weapons[currentWeaponIndex] != null)

{

weapons[currentWeaponIndex].SetActive(false);

}

// Procura o índice da arma no array

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

{

if (weapons[i] == weapon)

{

// Ativa a nova arma no slot designado

weapon.transform.position = weaponSlot.position;

weapon.transform.SetParent(weaponSlot);

weapon.SetActive(true);

currentWeaponIndex = i;

return;

}

}

}

}

Esse é um exemplo de o jogador com seu personagem pegar um objeto (arma) no cenário. Atenção ao conceito de física que será implementada.

**Ajuste da mecânica da arma – 02**

Precisa ajustar a animação da arma (especificamente o disparo). Basicamente foi feito uma animação de um flash do disparo da arma e uma da munição da arma.

Temos que controlar a animação e o momento que ela é ativada e desativada (pela decisão do jogador em apertar o botão).

Poderíamos usar um coroutine, como exemplo:

using UnityEngine;

public class WeaponFlash : MonoBehaviour

{

public GameObject flashObject;

public float flashDuration = 0.1f;

private void Start()

{

flashObject.SetActive(false);

}

public void ShowFlash()

{

StartCoroutine(FlashCoroutine());

}

private IEnumerator FlashCoroutine()

{

flashObject.SetActive(true);

yield return new WaitForSeconds(flashDuration);

flashObject.SetActive(false);

}

}

Para manter controlado a animação no objeto (arma), pode ser usado Animator Controller é adicionar o flash. Lembrando que isso precisa está aninhada em toda as direções em que o jogador esteja navegando no momento.

**Adicionando objetos no cenário – 03**

Temos alguns assents de objeto de cenário, como árvores, casa, papeis e outros. Vamos coloca-los no cenário de forma alinhada, porém antes temos que definir os tiles map e tiles sets, para isso criamos o Tile Palete (pintura do cenário).

Nisso usaremos essa classe para colocar os objetos de cenários (foi organizado em conjunto em uma imagem png), definindo suas camadas (frente e trás) e aplicando o Tilemap Collider 2D para as colisões. E fazendo os ajustes finais para consertar bugs de colisões com o Player ou disparo da arma.

Exemplo de um código de colisão aos tiles:

using UnityEngine;

using UnityEngine.Tilemaps;

public class TilemapCollision : MonoBehaviour

{

private Tilemap tilemap;

private TilemapCollider2D tilemapCollider;

private void Start()

{

tilemap = GetComponent<Tilemap>();

tilemapCollider = GetComponent<TilemapCollider2D>();

}

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject.CompareTag("Player"))

{

ContactPoint2D contact = collision.GetContact(0);

Vector3 hitPosition = contact.point;

Vector3Int cellPosition = tilemap.WorldToCell(hitPosition);

TileBase tile = tilemap.GetTile(cellPosition);

if (tile != null)

{

Debug.Log("Collision with tile: " + tile.name);

// Aqui você pode adicionar lógica adicional para lidar com a colisão com o tile específico.

}

}

}

}

**Disparando um projetil no cenário – 04**

Como deve saber, temos um Sprite representando a munição de uma arma. Nela tem uma animação do efeito da munição. E o objetivo é quando o jogador aperta o botão de disparar essa munição (projetil) saia do objeto arma (especificamente da boca da arma) e navegue com uma velocidade adequado.

E quando esse projetil se colidir com um outro objeto (inimigo, casa, barril e outros) seja destruído para não custa muita memória. Então devemos ter o controle da animação e do efeito no cenário. Consistindo em um código mais limpo.

Podemos definir uma forma genérica para que o projetil identifique quaisquer objetos que ela colidir e seja destruída.

Por se tratar de classe diferentes, uma é a arma e outro é o projétil, faça a relação delas e seus controles.