¿Qué es la generación procedural?

La generación procedural es una técnica de programación que permite crear contenido de manera aleatoria utilizando un conjunto de reglas y parámetros predefinidos.

En el contexto de los videojuegos, se utiliza para generar entornos, niveles, personajes, objetos y otros elementos.

Motivo.

El motivo de esta herramienta es ser un complemento para un proyecto mas grande, esta herramienta se encarga de generar cuartos de manera aleatoria y también se ocupa de no crear aberturas en el mapa.

Esto presupone ser un apoyo al momento de ser implementado, puede editarse tanto la parte visual como la parte de codigo para acoplarse a las necesidad de los usuarios.

También se cuenta con un menú de pausa con las opciones de reiniciar el la herramienta y generar un nuevo mapa y también de salir por completo de la herramienta misma.

Detalles de los codigos.

-RoomTemplates-

```
public GameObject[] bottomRooms;
public GameObject[] topRooms;
public GameObject[] rightRooms;
public GameObject[] leftRooms;

public GameObject closedRoom;

public List<GameObject> rooms;

System.Random rnd = new System.Random();

private int roomCount = 0;
public Text roomCountText;


public void IncrementRoomCount()
{
    roomCount++;
    UpdateRoomCount();
}

private void UpdateRoomCount()
{
    roomCountText.text = "Cuartos Generados: " + roomCount.ToString();
```

Funcion:
El código mostrado anteriormente forma parte de un sistema de generación procedural de habitaciones para una herramienta de videojuegos. su función principal de es contar, enlistar y mediante otro script generar habitaciones y pasillos interconectados.

Para lograr esto, el código utiliza listas de objetos para guardar los diferentes tipos de habitaciones que hemos agregado y una habitación totalmente cerrada. También se utiliza la clase System.Random para elegir aleatoriamente entre las habitaciones que se genraran.

-PlayerMovement-

```
public CharacterController controller;
public float speed = 10f;
private Vector3 playerPosition;
private Quaternion playerRotation;
public float X, Y;


void Update()
{

    float x = Input.GetAxis("Horizontal");
    float z = Input.GetAxis("Vertical");


    Vector3 move = transform.right * x + transform.forward * z;
    controller.Move(move * speed * Time.deltaTime);
}
```

Funcion:

El codigo permite el movimiento de nuestro personaje dentro de la herramienta usando teclas de dirección (W, A, S, D).

Defininimos una variable de tipo CharacterController para controlar el movimiento. También definimos su velocidad inicial para y las variables playerPosition y playerRotation para guardar la posición y rotación del personaje.

El código permite que el jugador controle el movimiento del personaje.

-RoomSpawner-

```
public int openSide;

private RoomTemplates templates;
private int rand;
private bool spawned = false;

private void Start()
{
    templates = GameObject.FindGameObjectWithTag("Rooms").GetComponent<RoomTemplates>();
    Invoke(nameof(Spawn), 0.1f);
}

private void Spawn()
{
    if (spawned) return;

    switch (openSide)
    {
        case 1:
            rand = Random.Range(0, templates.bottomRooms.Length);
            Instantiate(templates.bottomRooms[rand], transform.position, templates.bottomRooms[rand].transform.rotation);
            break;

        case 2:
            rand = Random.Range(0, templates.topRooms.Length);
            Instantiate(templates.topRooms[rand], transform.position, templates.topRooms[rand].transform.rotation);
            break;

        case 3:
            rand = Random.Range(0, templates.leftRooms.Length);
            Instantiate(templates.leftRooms[rand], transform.position, templates.leftRooms[rand].transform.rotation);
            break;
        case 4:
            rand = Random.Range(0, templates.rightRooms.Length);
            Instantiate(templates.rightRooms[rand], transform.position, templates.rightRooms[rand].transform.rotation);
            break;
    }

    templates.IncrementRoomCount();
    spawned = true;
}

private void OnTriggerEnter(Collider other)
{
    if (!other.CompareTag("SpawnPoint")) return;

    var otherSpawner = other.GetComponent<RoomSpawner>();
    if (otherSpawner != null && !otherSpawner.spawned && !spawned)
    {
        Instantiate(templates.closedRoom, transform.position, Quaternion.identity);
        Destroy(gameObject);
    }

    spawned = true;
}
```

Funcion:

Este codigo es el complemento de “RoomTemplate” se encarga de la generacion en orden de los cuartos con base a lo mostrado en su codigo complemento. La función actual es controlar en qué posición aparece una habitación.

El valor de "openSide" indica el lado de la habitación que debe ser abierta (abajo, arriba, izquierda o derecha). La función "Start" inicia la variable "templates" y llama a la función "Spawn" para generar una habitación de manera aleatoria.

La función "Spawn" elige una habitación al azar de una lista de habitaciones correspondiente al valor de "openSide" la instancia en la posición actual del objeto.

La función "OnTriggerEnter" se encarga de destruir el objeto actual y crear una habitación cerrada en su lugar si se detecta una colisión con otro objeto, en este caso si una habitación se genera encima de otra.

-AddRoomToList-

```
private RoomTemplates templates;


void Start()
{
    templates = GameObject.FindGameObjectWithTag("Rooms").GetComponent<RoomTemplates>();
    templates.rooms.Add(this.gameObject);
}
```

Funcion:
El código es un complemento para la generacion de las habitaciones, este mismo agrega la habitación actual a una lista de habitaciones en el objeto "RoomTemplates" para que pueda ser utilizada en la generación de nuevas habitaciones.

-cameraLook-

```
public float mouseSensitivity = 60f;
public Transform playerBody;
float xRotation = 0f;


void Update()
{


    float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity * Time.deltaTime;
    float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity * Time.deltaTime;

    xRotation -= mouseY;
    xRotation = Mathf.Clamp(xRotation, -90f, 90);

    transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
    playerBody.Rotate(Vector3.up * mouseX);
}
```

Funcion:

El código permite que el jugador pueda mover la cámara en cualquier dirección en función del movimiento del ratón. La sensibilidad del ratón se puede ajustar a través de la variable "mouseSensitivity". Además, el código también realiza una limitación en la rotación vertical de la cámara para evitar que la vista del jugador se invierta. también actualiza la rotación del cuerpo del jugador en función del movimiento horizontal del ratón.

Variables Usadas:

- En el primer código, las variables utilizadas son:
    - bottomRooms: una matriz de GameObjects que contiene las habitaciones disponibles para ser colocadas en la parte inferior de la pantalla.
    - topRooms: una matriz de GameObjects que contiene las habitaciones disponibles para ser colocadas en la parte superior de la pantalla.
    - rightRooms: una matriz de GameObjects que contiene las habitaciones disponibles para ser colocadas en la parte derecha de la pantalla.
    - leftRooms: una matriz de GameObjects que contiene las habitaciones disponibles para ser colocadas en la parte izquierda de la pantalla.
    - closedRoom: un GameObject que se usa para cerrar una habitación si no se puede colocar ninguna habitación adyacente.
    - rooms: una lista de GameObjects que contiene todas las habitaciones generadas en el juego.
    - rnd: una instancia de la clase System.Random utilizada para generar números aleatorios.
    - roomCount: un contador que realiza un seguimiento del número de habitaciones generadas.
- En el segundo código, las variables utilizadas son:
    - controller: un objeto CharacterController que controla el movimiento del personaje.
    - speed: una variable que determina la velocidad de movimiento del personaje.
    - playerPosition: una variable que almacena la posición actual del personaje.
    - playerRotation: una variable que almacena la rotación actual del personaje.
    - X, Y: variables utilizadas para almacenar la entrada del usuario (movimiento del joystick o del teclado).
    - x: una variable que almacena la entrada horizontal del usuario.
    - z: una variable que almacena la entrada vertical del usuario.
    - move: un Vector3 que representa el movimiento del personaje en la dirección horizontal y vertical.

- En el tercer código, las variables utilizadas son:
    - openSide: un entero que indica en qué dirección se puede colocar la siguiente habitación.
    - templates: un objeto RoomTemplates que contiene matrices de habitaciones para ser generadas.
    - rand: un entero aleatorio utilizado para seleccionar una habitación de la matriz correspondiente.
    - spawned: un booleano que indica si la habitación ya ha sido generada.
    - other: un Collider que representa la colisión entre la habitación actual y otra habitación.
    - otherSpawner: una instancia de la clase RoomSpawner utilizada para determinar si la habitación adyacente ya ha sido generada.