**Conceptos Unity**

1. Hoja de sprites

Al agregar una hoja de sprites en la parte de edición se puede indicarle de cuantos pixeles está conformada cada sub-imagen; las imágenes también se pueden recortar de modo automático en la herramienta de “Sprite Editor” además es posible agregar márgenes y un padding al rededor. Si la imagen es borrosa se debe cambiar el “Filter Mode” a point en lugar de bilinear.

1. Project Settings
   1. Input

GetAxisRaw permite obtener el valor que el input por defecto cambia según presionemos las flechas

Todo esto definido en input

1. Animator

Controla los clips de animaciones con máquinas de estado.

* animator.SetFloat ("moveX",Input.GetAxisRaw("Horizontal"));

Según las variables se producen transiciones ya que desde el código se puede cambiar la máquina de estado a ejecutarse

* “Fixed duration” marcado indica que no sea en bucle el movimiento, es necesario desmarcarlo en la mayoría de ocasiones.
* “Transition duration” es el tiempo que se demora en pasar de una animacion a otra, de preferencia para el 2d en movimiento ese valor debe ser 0
  1. Blend Trees

Es una forma organizacional de estados

1. Colisiones

Para que los objetos colisionen no basta con las “box colider 2D” es necesario “Rigidbody 2D”

* void OnTriggerEntered2D(Collider2D other){}
* void OnCollisionEntered2D(Collision2D other){}

También hay la versión Exit, Enter y la persistida

1. Capas

“Sorting layers” indica que objeto irá encima de otro.

Podemos apoyarnos en el vector z para esto pero no es muy aconsejable.

1. Instanciación de un objeto especifico a partir de uno genérico (prefab)

* var clone = (GameObject) Instantiate (danio\_flotante, col.gameObject.transform.position, Quaternion.Euler (Vector3.zero));

clone.GetComponent<NumerosFlotantes> ().numero = damage;

El código de arriba clona el prefab para permitir la modificación de sus atributos

1. UI interfaces y Canvas
   1. Texto flotante

Primero se lo aparta del canvas agregándole la opción “world space” de esa manera no se ata al UI. El texto se lo crea en una posición y se aumenta su eje y.

* 1. Barra de vida

Con un slider se puede crear una barra de vida, eliminando las partes innecesarias

1. Sistema de partículas
   1. Sangre

Se usó la instanciación esférica, de color rojo, con tamaño reducido y creación de cuadros

1. NPC
   1. Movimiento

Para limitar el movimiento de un npc en 2d lo encerramos en un box collider que solo detectará el mismo. También se puede usar la mecánica del 3D donde se usa EmptyObjects como target de una ruta (mirar el script del oso)

1. Composición (POO) en MonoBehaviour

Para realizar composición en lugar de “new” debemos agregar componentes

* PlayerStats estadisticas = gameObject.AddComponent <PlayerStats>();

1. Triggers

Los objetos vacíos pueden servir para activar cosas en el juego, esto es importante.

1. Audio

Para agregar un sonido rápidamente se puede usar un prefab que cree algo y arrastrando el clip de sonido al mismo sonara al ser instanciado.

Otra manera es crear un controlador (con varios audios sources) con la música ya cargada. Agregamos esos controladores a nuestra clase donde habrá sonido y simplemente con la función .Play podemos reproducir el efecto cuando deba suceder

1. Escenas
   1. Cambio de escenas

Primero se cargan en BuildSettings todas las escenas a utilizar

Luego en código dentro de un objeto trigger se importa using UnityEngine.SceneManagement; y se usa SceneManager.LoadScene (level2load); siendo level2load el nombre de la escena

1. Persistencia de datos

DontDestroyOnLoad (transform.gameObject);

1. Comunicación entre scripts

* col.gameObject.SendMessageUpwards ("applyDamage", 10);

Podemos usar una función de otro script para realizar una acción, adicionalmente podemos indicarle si es necesario o no un recibidor.

1. Desactivación en lugar de destrucción

Con esto evitamos errores y simplemente lo hacemos no jugable

col.gameObject.SetActive (false);

1. Transform

Para cambiar la posición local o rotación usamos

* this.transform.localPosition = new Vector3 (0.004f,-0.292f,0f);
* this.transform.eulerAngles = new Vector3(0f, 180f, -179.13f);

Importante en el rotation no es necesario usar Quaterniones

1. Animator

Animator animator = GetComponent<Animator> ();

animator.Play ("saltar",-1,0f);

**Funciones en Unity**

1. Lerp

Vector3.Lerp (transform.position, target\_position, move\_speed\*Time.deltaTime);

Lerp interpola los valores de inicio a fin de manera lenta y continua.

Usado en una cámara no va tan pegada al objetivo, va lentamente.

1. Velocity en RigidBody

* rb.velocity = new Vector2(axis.x\*velocidad\_movimiento,rb.velocity.y);

Modifica la velocidad del personaje y lo hace caminar sin detención en una dirección

1. Clamp

Para anclar el valor de una variable como de la cámara usamos clamp

Mecánica BOUNDS

cam = GetComponent<Camera> ();

alto\_vision\_mitad = cam.orthographicSize;

ancho\_vision\_mitad = alto\_vision\_mitad \* Screen.width/Screen.height;

float clampedX = Mathf.Clamp (transform.position.x, min\_bound.x + ancho\_vision\_mitad, max\_bound.x - ancho\_vision\_mitad);

float clampedY = Mathf.Clamp (transform.position.y, min\_bound.y + alto\_vision\_mitad, max\_bound.y - alto\_vision\_mitad);

transform.position = new Vector3 (clampedX, clampedY, transform.position.z);

clamp no retorna un valor menor al mínimo o mayor al máximo

1. Translate

translate -> le avisa al motor que queremos cambiar las posiciones x,y,z de transform

transform.Translate (new Vector3 (Input.GetAxisRaw("Horizontal")\*velocidad\_movimiento\*Time.deltaTime,0,0));