«Talento Tech»

Desarrollo de Videojuegos

Unity 3D

Clase 09

Buenos Aires Ciudad

Clase N° 9 | Interacción con UI en 3D II

Temario:

- Ejemplo Barra de vida en enemigo
- ProBuilder

Objetivos de la clase

Implementar una barra de vida visual para un enemigo.

- Diseñar una barra de vida que se alinee siempre con la cámara para garantizar su visibilidad.
- Programar su comportamiento para que desaparezca al alejarse y evitar sobrecargar la pantalla con elementos innecesarios.

Explorar el uso básico de ProBuilder en Unity.

- Crear figuras básicas y personalizarlas para construir prototipos de niveles u objetos 3D.
- Aprender a editar vértices, bordes y caras para ajustar la geometría según las necesidades del proyecto.

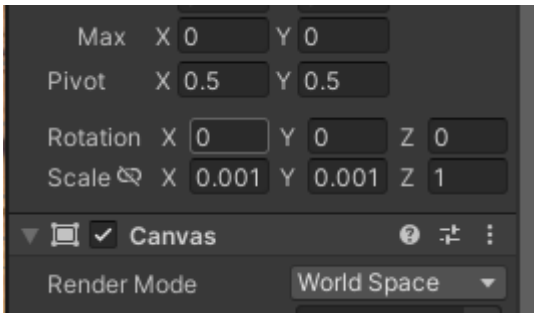
Ejemplo UI Ingame (barra de vida enemigo)

En la clase de hoy seguiremos trabajando con las capacidades de una UI ingame.
Como primer ejemplo crearemos una barra de vida sencilla que se ubique arriba de los enemigos.

Seteo de la barra

Al igual que la clase pasada crearemos una barra de vida dentro de un canvas en “World Space”, pero esta vez la acomodaremos como un Child del enemigo.

Le pondremos de escala 0.001 y lo posicionamos en 0,0,0, para que se nos haga más fácil acomodarlo por encima de la cabeza del enemigo.



Y ahora le acomodaremos nuestra barra de vida

Podríamos pensar que ya terminamos, pero no, si ponemos play y vamos por el costado del enemigo, veremos que la barra de vida no apunta a nosotros/as y por lo tanto, nos dificulta su visualización.

Script LookAt

Esto puede ser bastante molesto si vemos al enemigo desde perspectivas. Para corregirlo usaremos un comando sencillo

```
Transform target;
// Start is called before the first frame update
void Start()
{
    target = GameObject.Find("Player").transform;
}

// Update is called once per frame
void Update()
{
    transform.LookAt(target);
}
```

Vamos a guardar la referencia de nuestro Player en la variable “Target” y en el update ordenaremos que “vea” a nuestro target, haciendo que el objeto gire constantemente para apuntar al Player.

Ahora nos encontraremos con un posible problema. ¿Queremos que todas las barras del juego apunten a nuestro jugador y sean visibles constantemente? Esto supondría un consumo importante de recursos.
Para solucionarlo crearemos un Script de activación en nuestro enemigo.

Active según distancia

```
    GameObject bar;
    Transform player;
    [SerializeField]float MinDistance = 10f;

    void Start()
    {
        bar = transform.GetChild(2).gameObject;
        player = GameObject.Find("Player").transform;

    }


    void Update()
    {
        float d = Vector3.Distance(player.position,
transform.position);

        if (d <= MinDistance && !bar.gameObject.activeInHierarchy)
        {
            bar.SetActive(true);
        }else if (d > MinDistance)
        {
            bar.SetActive(false);
        }
    }
```

Explicación

Explicaremos este Script que es un poco más rebuscado:

Variables

```
    GameObject bar;
    Transform player;
    [SerializeField]float MinDistance = 10f;
```

`GameObject bar`:

- Representa la barra (u otro elemento del HUD/UI) que se activará o desactivará en función de la distancia.

`Transform player`:

- Referencia al transform del jugador (para obtener su posición en el espacio).

`[SerializeField] float MinDistance`:

- Una distancia mínima en la que el jugador debe estar para que la barra sea visible.
- Marcado con `[SerializeField]`, lo que permite ajustar este valor desde el Inspector de Unity sin necesidad de hacerlo público.

Método Start()

```
    void Start()
    {
        bar = transform.GetChild(2).gameObject;
        player = GameObject.Find("Player").transform;

    }
```

`transform.GetChild(2).gameObject`:

- Obtiene el tercer hijo (índice 2) del objeto al que está vinculado este script y lo asigna a la variable `bar`.
- Se asume que este hijo representa la barra que se mostrará/ocultará.

`GameObject.Find("Player").transform`:

- Busca en la jerarquía un objeto llamado `"Player"` y obtiene su `Transform`.
- Esto permite acceder a la posición del jugador en el mundo.

Método Update()

```
    void Update()
    {
        float d = Vector3.Distance(player.position,
transform.position);

        if (d <= MinDistance && !bar.gameObject.activeInHierarchy)
        {
            bar.SetActive(true);
        }else if (d > MinDistance)
        {
            bar.SetActive(false);
        }
    }
```

`Vector3.Distance(player.position, transform.position)`:

- Calcula la distancia entre el jugador y el objeto al que está asociado este script.
- **`player.position`**: Posición del jugador.
- **`transform.position`**: Posición del objeto actual.
- El resultado se guarda en la variable `d`.

Condición: `if (d <= MinDistance && !bar.gameObject.activeInHierarchy)`:

- Comprueba si la distancia entre el jugador y el objeto (`d`) es menor o igual a la distancia mínima (`MinDistance`).
- También verifica si la barra no está actualmente activa (`!bar.gameObject.activeInHierarchy`).

`bar.SetActive(true)`:

- Si ambas condiciones son verdaderas, activa la barra (`bar`), haciéndola visible en el juego.

Condición: `else if (d > MinDistance)`:

- Si la distancia es mayor que `MinDistance`, desactiva la barra.

`bar.SetActive(false)`:

- Desactiva la barra (`bar`), haciéndola invisible.

Finalmente tendremos una barra de vida sobre nuestro enemigo que solo aparecerá cuando nos acerquemos lo suficiente a él

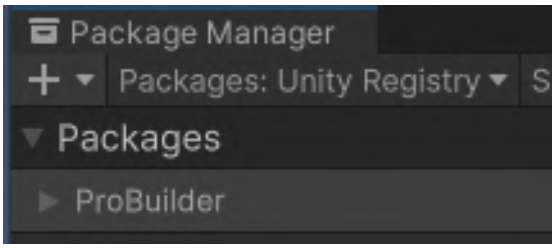
ProBuilder

¿Qué pasaría si deseamos crear una mejor pared o puerta o ambiente? ¿Tendremos que descargar todo desde la Asset Store o unir cada figura base?
Podemos usar la herramienta de “Pro Builder”.

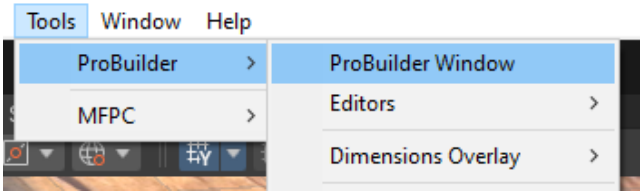
Es un package muy útil que nos puede ayudar a prototipar nuestros niveles y ahora lo usaremos para crear algunas partes sencillas de nuestro nivel, como una pared con el espacio para una puerta.

Descarga

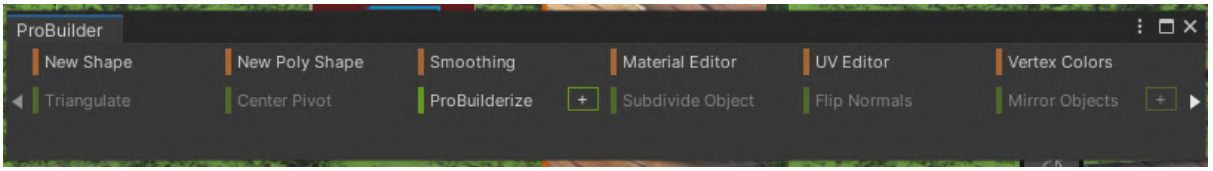
Primero iremos a Windows->Package Manager.
Cambiaremos el Registro para que sea de Unity y en el buscador de la derecha, pondremos “ProBuilder”



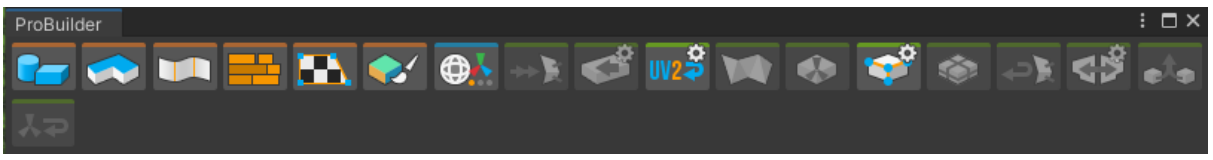
Lo descargamos e iremos a la solapa de arriba que dice “Tools” -> ProBuilder -> Probuilder Window



Apareciendo algo como esto



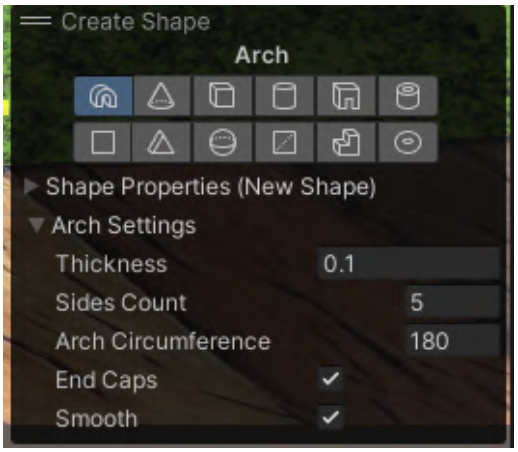
Que si no nos gusta como se ve, la derecha en los 3 puntitos podremos cambiarlo al “Icon Mode”:



Dentro de esta ventana, tendremos numerosas herramientas. Pero por ahora usaremos “New Shape” y “New Poly Shape”

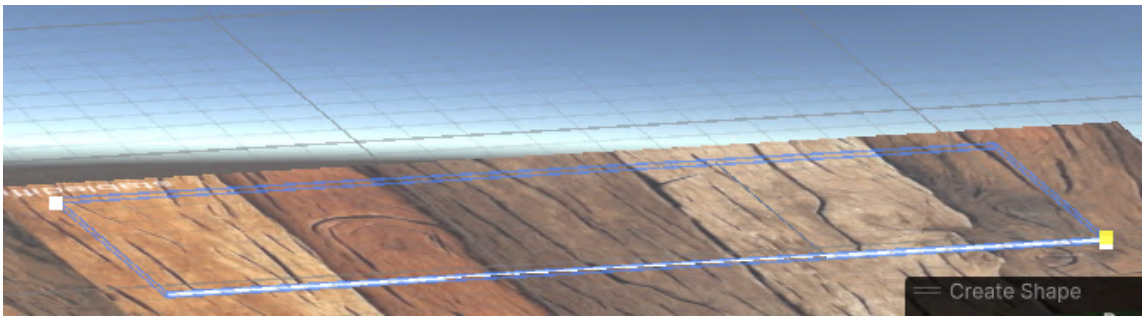
New Shape

La primera nos dará una serie de opciones para crear formas:

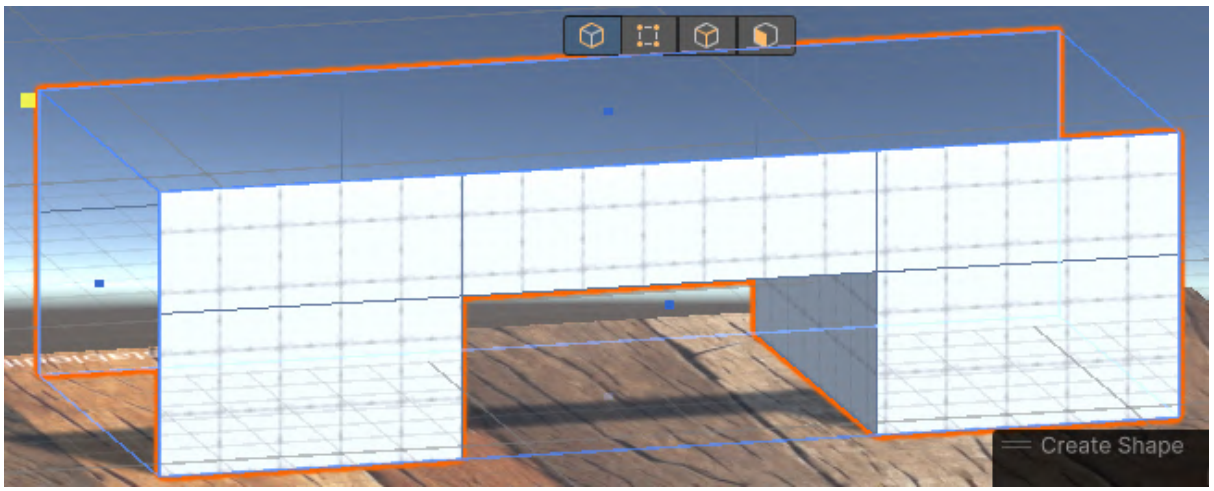


Cada forma tiene una variedad de propiedades que nos permitiran modificarla a gusto. Para hacer nuestra “puerta” iremos a la quinta opción “Door”, le pondremos a “Pendiment High” un valor de 2 y a “Side Width” un valor de 5.

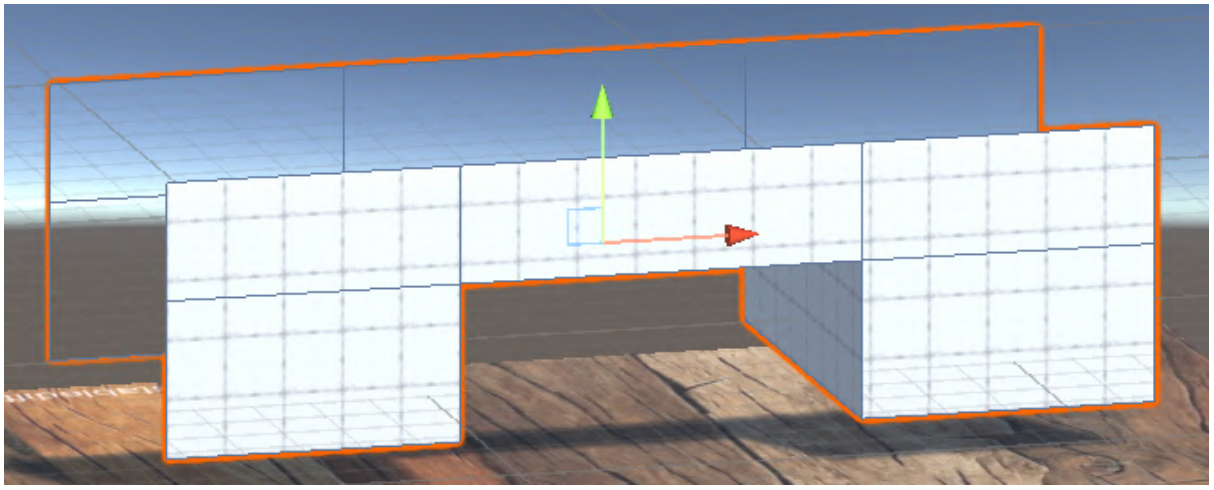
Seguiremos, haciendo click en cualquier suelo y manteniendo arrastraremos paraDesignar la superficie de nuestra puerta



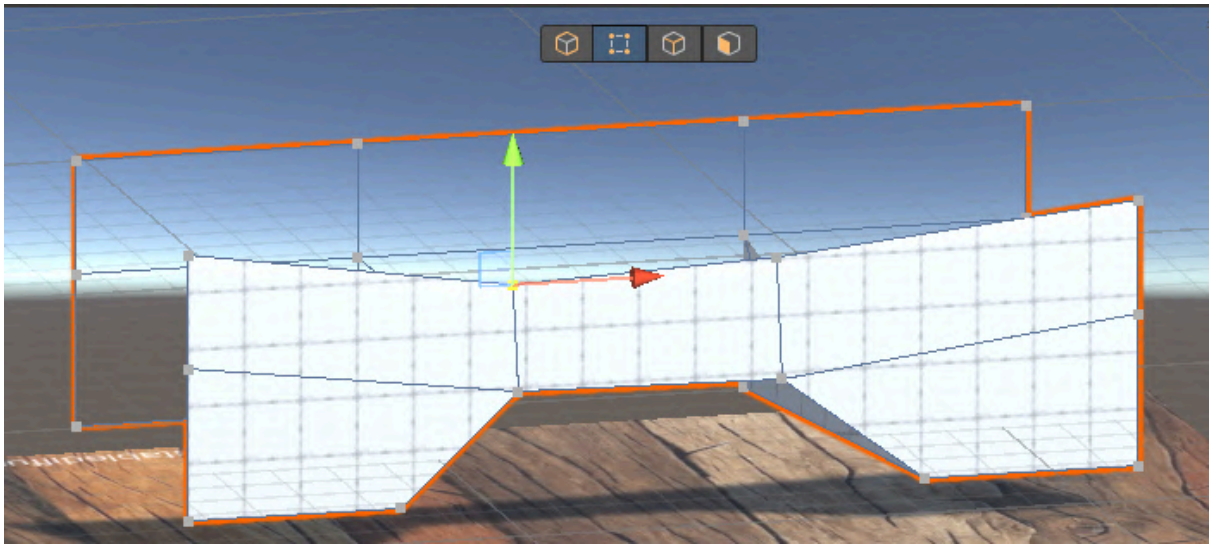
Cuando ya tengamos el valor deseado, soltaremos el click y levantaremos para ver cómo emerge nuestra estructura. Al hacer click otra vez esta se creará definitivamente.
Notaran que parte de la imagen no se “**Renderiza**” y eso es debido a que, se supone que este elemento se utiliza para interiores o exteriores donde no podemos ver por encima o sus costados. Lo que hace Probuilder es trabajar con las “**normales**” del objeto. Las normales son las partes visibles/renderizadas de mi figura. Porque si TODO el objeto se renderizada, este **gastaría grandes recursos** en trabajar en algo que muy posiblemente NUNCA utilizaremos.



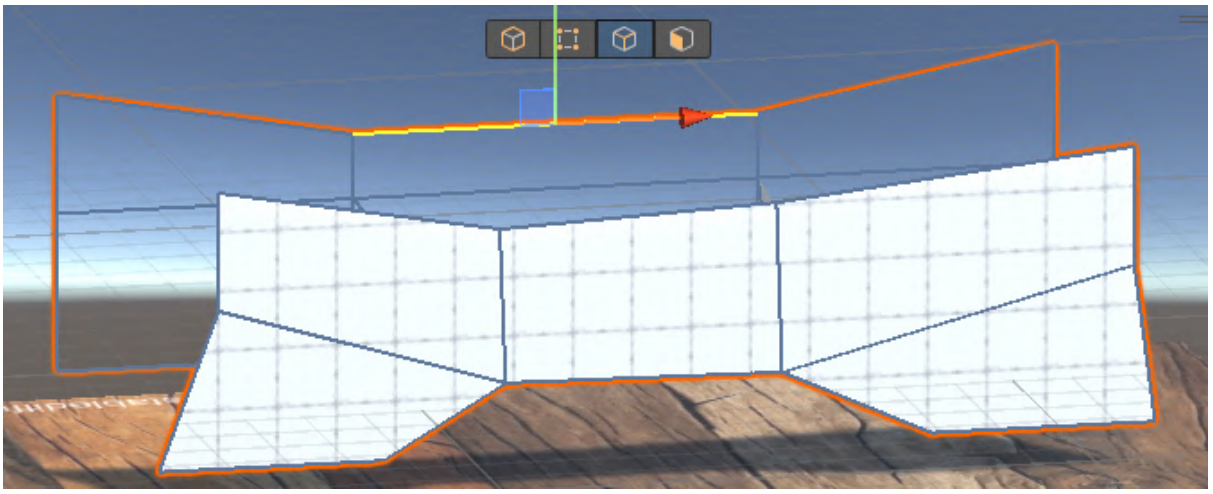
Veremos como en la parte superior nos aparece un nuevo set de herramientas. Estas nos permitirán modificar la figura desde diferentes puntos.
La primera nos permite manejar la figura como un todo.



La segunda nos dejara cambiar de lugar sus vertices

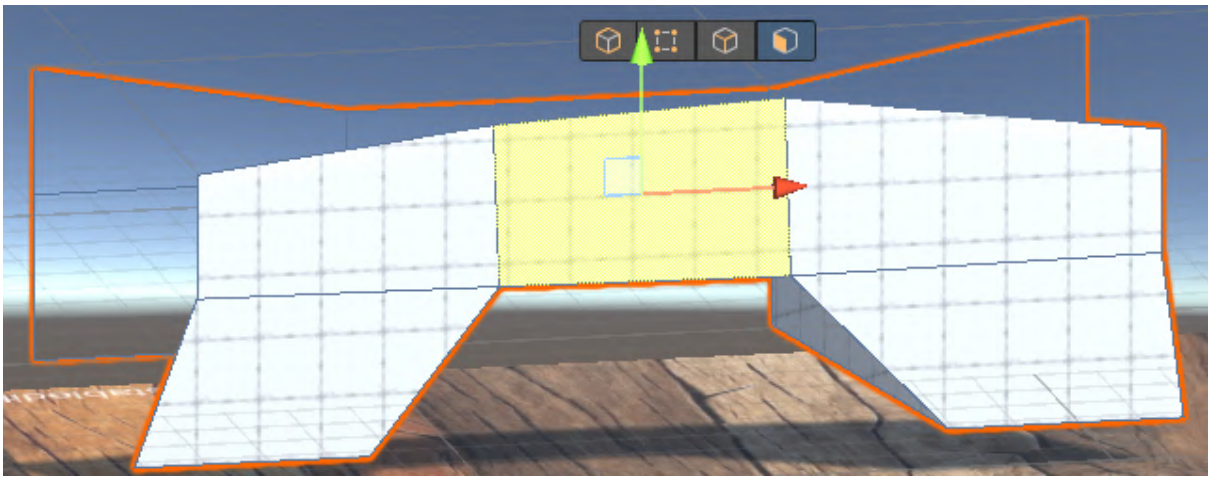


La tercera, mover sus “bordes”



Y la ultima, la cuarta, nos dejara modificar sus “caras”

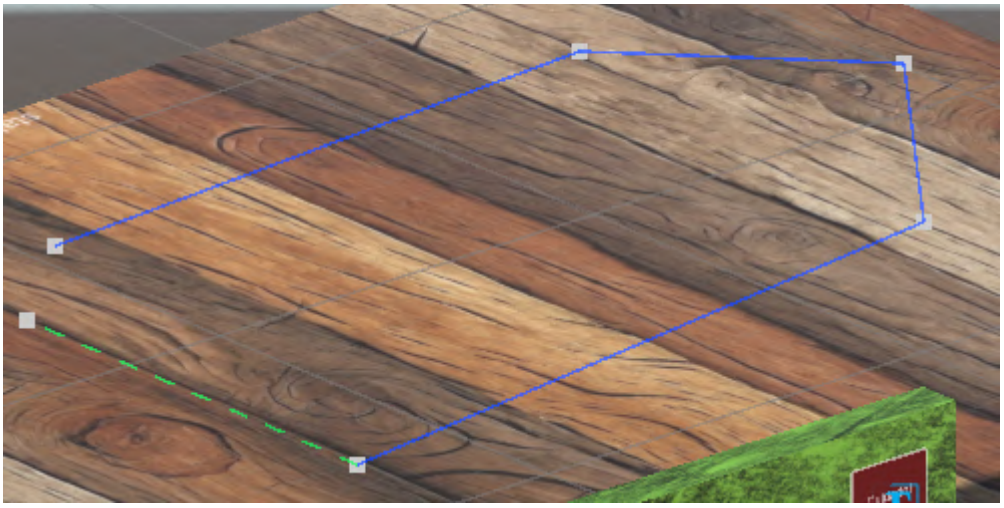
Con esto podremos modificar la figura a gusto.

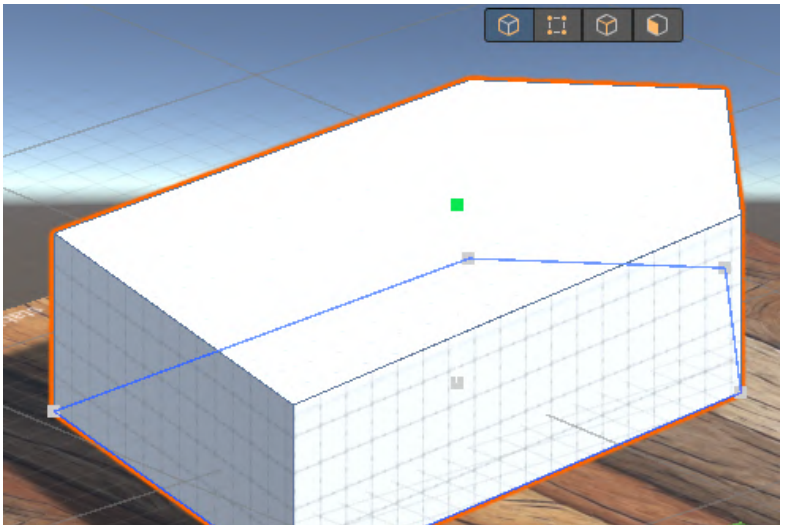


Cada figura que elijamos tendrá sus formas y propiedades pero TODAS tendrán estas herramientas para modificar. Ahora seguiremos con la siguiente opción.

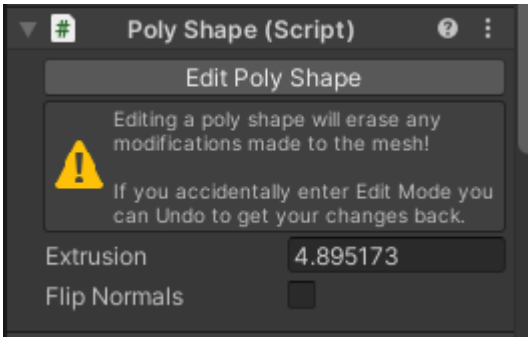
New Poly Shape

Esta opción, nos dejará crear una figura “a mano” realizando el dibujo sobre una superficie y luego extruyendolo para darle una altura:





De la misma manera que la anterior, podremos utilizar las herramientas de vértices, borde y cara para modificarla.
Si la figura ya fue creada y desean seguir modificando, pueden seleccionarla ir al Script “Poli Shape (Script)” y hacer click en “Edit Poly Shape”



Esto nos habilitará nuevamente para modificar las propiedades/variables de nuestra figura.

Vida y Maqueta:

Después de implementar el panel dinámico de datos para el jugador, Talento Lab da un giro en el enfoque del proyecto: *"Nuestro mundo está empezando a cobrar vida, pero no podemos olvidar que los enemigos también forman parte de esta experiencia inmersiva."*

En una nueva reunión, el cliente destaca la importancia de crear enemigos que no solo desafíen al jugador, sino que también proporcionen información visual clara sobre su estado durante los enfrentamientos. Para ello, solicitan implementar una **barra de vida** en los enemigos que permita a los jugadores medir su progreso durante un combate.

Además, el equipo de TalentoLab recibirá un encargo relacionado con la construcción del mundo: es momento de empezar a crear estructuras más complejas e interactivas utilizando **ProBuilder**, una herramienta que permitirá modelar y personalizar los entornos directamente dentro de Unity.

Ejercicios prácticos:

Elizabeth te envió un mensaje: “Ahora que Nexus comienza a tomar forma, necesitamos que los entornos reflejen la profundidad y la complejidad del mundo que estamos creando. Antes de construir un nivel completo, queremos que realicen una maqueta utilizando **ProBuilder**. Esto nos permitirá iterar rápidamente sobre ideas de diseño antes de pasar a las versiones finales.”

El cliente resalta la importancia de este ejercicio como un paso inicial para definir la arquitectura y la estética del universo de Nexus. El objetivo es crear una maqueta funcional que incluya elementos básicos del diseño del nivel, como plataformas, pasillos, rampas y zonas de interacción. Esta maqueta será la base para recibir feedback y realizar ajustes antes de agregar texturas, materiales y detalles finales.

El equipo decidió que sería un buen trabajo para que pongas en práctica tus ideas de diseño de Niveles

---

Materiales y recursos adicionales.

ProBuilder
https://docs.unity3d.com/Packages/com.unity.probuilder@6.0/manual/index.html

---

Preguntas para reflexionar.

1. ¿Es necesario que los enemigos tengan barras de vida?
2. ¿Qué otra forma podemos crear para mostrar el daño hecho o recibido?
3. ¿De qué sirve maquetar un nivel?

Próximos pasos.

En la próxima clase empezaremos a trabajar con el guardado de información y la data persistente a través de escenas, utilizando un método distinto al que conocíamos.

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad