Juego para disparar a enemigos.

1. **Importar FPS.**

* Ir a **Assets** **- import package – characters**.
* Importar **Unity Package**.
* Activa todo, importar **Assets** folder que ahora contiene **Standard Assets**, **characters**, **First Person Characters**.
* Agrega un objeto **plane** para evitar que el jugador caiga.
* Ir a **hierarchy**, clic derecho, crear **3d object**, agregar un **cube**. Renombrarlo como **floor**, centrar Position 0 0 0. Escala a x = 20 y = 0.1 z = 20.
* Crear un folder **Game**.
* Dentro de este crear un a folder **Materials**.
* Dentro de este crear un **material** para el **plane**.
* Colocar el **FPS character** **Prefab** en la escena.
* Arrastrar el **FPSController** en la **hierarchy**. Este contiene **audio** y **camera**.
* Borrar **main camera**.
* Guardar la escena y arrastrarla y soltarla al folder **Scenes**.

1. **Crea un GameController.**

* En la **hierarchy** crear un nuevo objeto vacío y renombrarlo como **GameController**. Ponerlo en position 0 0 0.
* Selecciona **directional light**, **floor** y **FPS** y arrástralos y suéltalos a **GameController**.
* Renombra **FPSController** como **Player**.

1. **Crea un arma.**

* En la **hierarchy**, dentro del **FPS**, agrega un **empty** object y renómbralo como **Gun**.
* Agrega un cube dentro de Gun.
* Retira el **box collider** de este cube. Cambia scales y positiondel **cube**.
* Agrega un segundo **cube** y realiza acciones similares para formar una **L** como un arma.
* Selecciona **cubes - Mesh renderer - cast shadows – off**. Asígnale un **material**.

1. **Elabora un script para el Player y la Bullet**.

* No cambies el script del **FPS** porque es solo para el movimiento y la física.
* Crea un folder **Scripts** dentro del folder **project**.
* Dentro de este crea un script **Player** y un script **Bullet**.
* Agrega el script del **Player** como un **component** del **Player** en el inspector.
* En la **hierarchy,** selecciona **Player**, arrastra y suelta el script **Player** en el **Player**.
* Abre el script **Player**.
* Aquí tenemos el **update method** el cual es llamado en cada escena.
* **GetMouseButton** retorna **true** a cada momento que el mouse es presionado. Esto se usa para las **machine guns** de modo que tu puedas disparar varias **bullets**.
* Para una pistola usa **GetMouseButtonDown** de modo que puedas disparar una **bullet** cada vez que presiones el **trigger**. **(1)**

1. **Instanciar una Bullet y retirarla**.

* En el **GameController** object crea un **empty** object lllamado **Bullet**.
* Dentro de este crea un **empty** object llamado **Container** (un objecto intermedio para separar la parte **logic** de la parte **visual**).
* **Click derecho – create 3d object** – **Sphere**.
* Retira el **sphere collider**. Reescala la **sphere** reduciéndola. Crea un **material** para este.
* Cambia su **color**. Arrastra el script de la **Bullet** en el objeto **Bullet**.
* Agrega **component** - **sphere** **collider**. Cambia su radius para reducer su tamaño.
* Marca la casilla **Is** **Trigger** para usar la **Bullet** para **trigger overlaps**.
* Las **Bullets** van hacia los **Enemies** y los hiere o los mata.
* Crea un folder **Prefabs** en el folder **Game**.
* Seleciona la **Bullet** y arrastrla y suéltala al folder **Prefabs.**
* Borra la **Bullet** de la **hierarchy** porque ya está guardada.
* Abre el script **Player**. Aqui agrega **public** **GameObject bulletPrefab (2)**.
* En **Unity**, selecciona **Player** en **hierarchy**.
* Arrastra y suelta la prefab **Bullet** en el campo **Bullet Prefab** en **Player** inspector.
* Arrastra y suelta el **Player** dentro del folder **Prefab**.
* Para instanciar las **bullets (3)** necesitamos considerar las **bullets position** a una cierta **position y** en frente de la **camera** del **Player** y del **Player** mismo **(4)**.
* En el **Player Script** agrega una referencia a **Player** **Camera (5)**.
* En **Unity**, arrastra y suelta la **Camera** en el campo **Player Camera** – **apply**.
* Necesitamos mover la **Bullet** hacia adelante.
* Ve al script del **Player** y configuramos el **forward vector** de la **Bullet** para que sea el mismo que el **forward vector** de la **Player Camera (5)**. Este va a apuntar hacia donde esté mirando la **camera** **(6)**.
* Ve a **Bullet** scripts y haz una lógica para el movimiento de la **bullet**.
* Aqui definimos un **float speed (7)**.
* En el **update method** vamos a mover la bala **(8).**
* Las **bullets** salen para siempre, gastando la memoria.
* Vamos a agregar una **lifeDuration** al **Bullet script**, de modo que cada vez que agreguemos una **Bullet**, esta va a durar un tiempo **(9)**.
* Necesitamos un **lifeTimer** que va a ser un **internal clock** para la **Bullet** que va hacer una cuenta regresiva y cuando llegue a **0**, nos desaceremos de la **Bullet** **(10)**.
* Crear un **OnEnable method**, y hacer **lifeTimer = lifeDuration; (11)**.
* En el **update method** disminuimos el **Time.delaTime (12)** hasta que sea 0.
* Si esto sucede, la **Bullet** debe ser destruida **(13).**
* Cada vez que agreguemos una **bullet** a este juego (**instanciación**) le toma un tiempo a **Unity** conseguir ese elemento para ponerlo en el juego y hacerlo funcionar.
* Crear y eliminar elementos son operaciones costosas. Para evitarlas, necesitamos un **pooling** **manager** para mantener objetos.

1. **Object Pooling Manager.**

* Hacer esto nos permite mantener una **Bullet** y precargar una cierta cantidad de ellas.
* Hacemos estos objetos **inactive** y los escondemos del juego.
* Cada vez que generamos una **Bullet,** en lugar de instanciarlas vamos a obtenerlas del **pull of objects.**
* Ya tenenos una **Bullet** disponible, de manera que va a ser **activated**.
* Este proceso es conocido como **object pooling** y te ahorra mucha potencia de proceso y memoria.
* Crea un **empty object** y ponlo fuera del **GameController**.
* Renómbralo como **ObjectPoolingManager**.
* Posiciónalo en **0 0 0**.
* Hazle un script y llámalo **ObjectPoolingManager**.
* Vamos a poner una referencia. Arrastra y suelta el script **ObjectPoolingManager** en **ObjectPoolingManager**.
* En el script **ObjectPoolingManager** habrá una referencia a la instancia del **ObjectPoolingManager (14).**
* Hacemos un **pattern** para definir una **property (15).**
* Vamos a retornar **null**.
* En **ObjectPoolingManager** **script** en lugar de usar **Start method** podemos usar **Awake** **method (16)**.
* Nos deshacemos del **update method** y usamos un nuevo **GetBullet method** **(17)**.
* En el **Player** **script,** en el **update method**, dentro del bucle **if**, escribimos **ObjectPoolingManager.instance.GetBullet();** de modo que el jugador pueda conseguir acceso a **instance** en el **ObjectPoolingManager** **script** sin tener una referencia directa a éste **(18)**.
* En el **Player** script en vez de instaciar a la **Bullet**, vamos a obtener una **Bullet** del **ObjectPoolingManager**.
* El **ObjectPoolingManager** va a tener una referencia a la única instancia de ese **pooling manager** disponible en la escena.
* Necesitamos que el **ObjectPoolingManager** sea capaz de acceder al **Bullet** **prefab**. En **ObjectPoolingManager** escribe **public GameObject bulletPrefab**; **(19)**.
* En **Unity**, selecciona **ObjectPoolingManager** enla **hierarchy**, arrastra y suelta el **Bullet** **prefab** en el **Bullet Prefab field** del **ObjectPoolingManager** script.
* Necesitamos **precargar** un cierto número de **bullets**. En el **ObjectPoolingManager** script escribimos **public int bulletAmount = 20**; **(20).**
* En el **Awake method,** instanciamos **20 bullets (21).** Ejecutamos un **loop 20** times **(22).**
* Almacenamos lareferencia **(reference)** para la **Bullet (23).** Pasamos **transform** como un parámetro (**parameter)** porque va a referenciar al **ObjectPoolingManager (24)**.
* Las 20 **bullets** van a ser **desactivadas (disable) (25)**.
* En **Unity**, arrastre y suelte el **Bullet prefab** en el campo **Bullet Prefab** en el **ObjectPoolingManager script** en el inspector.
* Para **GetBullet method**, necesitamos definir una **List of bullets (26)** y en el **Awake method**, antes de que instanciemos las **bullets**, vamos a cargar una genérica **List of bullets** e incluir su **capacity** o **bullet amount, 20** en este caso **(27)** por lo que ya asignamos previamente esa cantidad de espacios en nuestra memoria para cargar las **bullets**.
* Hemos almacenado las referencias a las **bullets** antes de que intentemos usar el **GetBullet (28)**.
* Vamos a sentar las bases para hacer que funcione el **object pooling**. Necesitamos ir a **ObjectPoolingManager** luego tenemos que hacer que el método **GetBullet** funcione.
* Necesitamos iterar sobre todas las instancias **prefab** que hemos cargado y necesitamos verificar si alguna de ellas está **inactive** y si es así, esa será la **bullet** que vamos a usar **(29)**.
* Pero quizá todas las **bullets** están siendo usadas así que vamos a iterar sobre todas ellas no solo para que estén disponibles porque ya están todas en la escena. Esto significa que deberíamos instanciar otra **bullet** así que tomamos el código del bucle **for** para hacer una nueva **prefab bullet (30)**.
* Configuramos su **parent** para que sea el **ObjectPoolingManager (32)** y lo almacenamos en la **bullet list** **(33)** y retornamos la **bullet** agregada **(34)**.
* En el **Player** script, obtendremos la **bullet** del **ObjectPoolingManager.** Copiaremos **ObjectPoolingManager.Instance.GetBullet();** y lo pegaremos en lugar de **Instantiate (bulletPRefab)**; **(35)**. Con este cambio, la bullet es generada usando **ObjectPoolingManager.**
* Retire la referencia para el **bullet Prefab** porque el **Player** no necesita más saber cuál es el **prefab** específico **(36)**. En el **Bullet** script, escribe **gameObject.setActive = true;** en lugar de llamar a **Destroy(gameObject);** **(37)**.
* El **lifeTimer** se establece en el **Start method** al que solo se llama cuando el objeto es instanciado, pero como este va a ser lanzado, necesitamos cambiar este **Start** a **OnEnable (38)**. Cada vez que hacemos clic, la duración es devuelta a su estado original en el **lifeTime**.
* Si inicializamos la **Bullet Amount** a 2 en el inspector del **ObjectPoolingManager** y si hacemos clic en la **bullet** 1 se instancia, si hacemos clic dos veces, ambas están activas. Estamos reutilizando las **bullets** retirándolas del **ObjectPoolingManager** y ya no las instanciamos más.
* Si hacemos clic varias veces muy rápidamente, nuestras **bullets** son agregadas al **ObjectPoolingManager** y este va a usar todas las **bullets** disponibles, pero ellas van a ser reusadas. No más asignación de memoria, no más una gran cantidad de procesamiento involucrado en agregar estas **bullets**. El juego va a ser muy eficiente.

1. **Queremos que el player tenga una cantidad limitada de munición.**

* Abre el **Player** script. Ponemos **public int initialAmmo = 12; (39)**.
* Ponemos una **private** variable que almacenará el **ammo (40)**.
* En el método **Start** hacemos **ammo = initialAmmo;** **(41)**.
* En el método **Update**, antes que generemos **bullets**, tenemos que verificar si tenemos suficiente **ammo** **(42)** después de presionar el botón del mouse para que el **Player** dispare.
* Por tanto, **ammo--;** disminuye en uno la cantidad de munición y lo almacena en esa variable **(43)**.

1. **Organiza, agrupa variables para leerlas en el editor**.

* Vamos a usar un atributo en el **Player** script. Le damos un nombre a la etiqueta “**Visuals**” que va a aparecer en el editor **(43)**.
* Haz lo mismo con la etiqueta “**Gameplay**” **(44)**.
* En **Unity**, puedes ver en la variable **initialAmmo** una vez que se inicia el juego, éste está inicializado a 12 y a medida que hacemos clic en **ammo** va disminuyendo hasta **0**.
* Y si intentamos hacer clic de nuevo, nos quedamos sin **ammo**.

1. **Inteface de Usario.**

* En el inspector, agrega **text Ammo:0**.
* Aumenta el tamaño de la fuente.
* Para hacer que ese valor sea leído por el **Player**, haz un script para controlar nuestro **canvas**.
* Haremos esto en el **GameController** script.
* Deberíamos mostrar cuántas **bullets** tiene el **Player**, por lo que necesitamos hacer una **User Interface**, un área para dar información al **Player**. Vaya a **hierarchy**, haga clic derecho, elija **UI – Canvas**, que es un área para colocar diferentes **User Interface**: texto, botones, grupo de botones, paneles.
* Haga doble clic en **Canvas**: elija **UI** y luego **text**.
* Hay una cámara separada que está renderizando todo esto.
* Seleccione la herramienta **resizing tool**, haga clic en el texto y muévalo hacia la parte superior izquierda.
* Puedes aumentar su tamaño. Elija el color y el estilo de fuente.
* En el inspector, agrega **text Ammo:0**.
* Aumente el tamaño de la fuente.
* Para hacer que ese valor se lea para el **Player**, cree un script para controlar nuestro lienzo. Haremos esto en el script de **GameController**.

1. **GameController script**.

* Este va a controlar varios aspectos del juego: **cuánto tiempo el Player** está vivo, verificar si el **Player está muerto** y enviar un mensaje si el ha muerto, contar el **ammo**.
* Ve al **Scripts** folder -- **Game** folder – crea un **GameController** script.
* Arrastra el **GameController** script en el **GameController**.
* Teniendo el **GameController** seleccionado, haz doble clic en el **GameController** script.
* Este objeto ha sido seleccionado.
* Abre el **GameController** **script**. Indicamos donde está el **Player**, así que escribe **public Player player;** **(45).**
* En **Unity**, arrastra y suelta el **Player** en la variable **Player** del **GameController** script en el inspector.
* Agregamos un cabecero “**Game” (46)**.

1. **Ammo system.**

* En Unity, arrastra el **Text** en el **ammoText**.
* En **Player** script para acceder a la variable **ammo** crea una **property**.
* Le damos al **Player** alguna chance para recuperar **ammo**.
* Hacemos un **box** que podemos recolectar. Crea an **empty object** en el **GameController** y llámalo **ammocrate**.
* Dentro de este crea un empty object llamado **Container**.
* Dentro de este crea un object llamado **Cube**.
* Retira su **box collider**.
* Reescálalo, proporcionale un **material** y **color**.
* En el **Game** folder crea un script llamado **ammocrate**.
* En **hierarchy** -- **ammocrate** arrastra y suelta el **ammocrate** script.
* En **hierarchy** arrastra el **container** y suéltalo en el **container** script.
* Agrega el componente **box collider** al **ammocrate**.
* Cambia el tamaño del **box collider**.
* Agrega el componente **rigid body** para aplicar la física al **ammocrate**.
* Agrega un **particle system** en el **ammocrate**.
* Necesitamos que si el **player** colisiona con el **ammocrate** entonces los recolecte.
* Tenemos **trigger collisions** y **physics collisions**.
* Ve al **Player** script.
* En el **ammocrate** script, vuelve al **Player** script, abre el **Prefabs** folder y arrastra y suelta el **ammocrate** dentro de él.

1. **Health system**.

* En el **Player** script asi como el **ammo,** podemos tener los **health** points.
* Agrega un **text** a **canvas** con **Health: 0**.
* En **GameController** necesitamos poner una referencia al **health text**.
* Arrastra el **text health** y suéltalo en el **GameController** inspector.

1. **Crea un enemy**.

* En el **GameController** crea un empty object.
* Renómbralo como **Spike**.
* Crea un empty object dentro de **Spike** y renómbralo como **Container**.
* Crea un **Cube** dentro de **Container**.
* Retira el **box collider** de este **Cube**.
* El **Spike** necesita tener un collider alrededor de él.
* En el **Script** folder en el **Game** folder crea un script y renómbralo como **Enemy**.
* En el **Enemy** script arrastra el **Enemy** y suéltalo en **Spike.**
* Arrastra y suelta el **Spike** en el **Prefab** folder y bórralo de la **hierarchy**.

1. **Sistema para Knockback Effect**.

* Cuando el **Player** colisiona con el **Enemy**, él va a recibir una fuerza para hacerlo impulsarlo hacia atrás.
* En el **Player** script implementa un **knockback effect**.
* Crea un script llamado **ForceReceiver** y ábrelo.
* El **ForceReceiver** script puede solo agregarse en **CharacterController**.
* Nuestro **Player** contiene el **CharacterController** que viene de nuestro **FPS package**.
* Tenemos el **Player** script y el **ForceReceiver** script.
* Ve al **Player** script.
* En el inspector seleccione **Player** y presione aplicar.

1. **Lógica de Disparo**.

* Crea un empty object en el **GameController**.
* Renómbralo como **ShootingEnemy**.
* Crea un **Container** object dentro de él.
* Dentro de este **Container** crea un **capsule** object.
* Retira el **capsule collider**.
* Aplica un **material** y **color**.
* Crea un script para ese **ShootingEnemy**.
* En el **Scripts** folder en el **Game** folder, abre el **ShootingEnemy** script.
* Cambia el extend **monobehavior** por extend **Enemy** porque ese es un enemigo también.
* Necesitamos que el **ShootingEnemy** le dispare balas al **Player**.
* **ShootingEnemy** necesita referenciar a las **Bullets**. Este también necesita tener una referencia de dónde está el **Player**, de modo que los **Enemies** puedan dispararle al **Player**.
* Arrastra y suelta el **ShootingEnemy** script en el **ShootingEnemy** inspector.
* Vuelve al **ShootingEnemy** script, la referencia a las **Bullet** **prefabs** está en nuestro **pooling manager** el cual está en el **Scripts** folder -- **Util** folder.
* Abre el **ObjectPoolingManager** para buscar una **Bullet**.
* En el **ShootingEnemy** script necesitamos una lógica para acceder al **Player** y para crear una **Bullet** y para dispararla al **Player**.
* Vamos a obtener la referencia al **Player** y para el **Player** script.
* Hacemos una lógica para los **ShootingEnemies** para que disparen al **Player**.
* Tenemos que crear una **Bullet**.
* Utiliza la lógica que hicimos para el **Player**: genera el **Bullet** object, establece su **position** y establece su **forward vector** en el **ShootingEnemy** script.

1. **Distancia de Disparo.**

* Vamos a hacer la lógica para asegurar que los **ShootingEnemies** no van a realizar los disparos si ellos no están lo suficientemente cerca al **player** (a la **shooting distance)**.
* Por ahora, el **Player** no es golpeado por la **Bullet,** las **bullets** no están destruyendo **Enemies** y los **Enemies** no están destruyendo al **Player**.
* Necesitamos una lógica en la **Bullet** que nos permita saber su **origen**. Si el **Player** dispara una **Bullet** no queremos que la **Bullet** hiera al **Player**. Si el **Enemy** dispara una **Bullet** no queremos que la **Bullet** hiera a un **Enemy**.
* En la **Bullet**, cada vez que el **Player** genera una **Bullet** necesitamos decirle que la **Bullet** va a ser generada por el **Player**.
* Ve al **ShootingEnemy** script.
* Cada vez que obtengamos una referencia a la **Bullet** en el **Player** script, tenemos dos cosas que pueden herir al **Player**: El **Enemy** y la **Bullet**.
* Ve al **Bullet** script. Ve al **Player** script.
* En Unity, en el **ShootingEnemy** -- **add component** -- **capsule collider**. Ajusta la altura a 2.
* Ahora cuando colisionemos con el **Enemy** disminuirá nuestra **health**.
* Selecciona **Player** en **hierarchy**, el **Player** tiene un **capsule collider**. Necesitamos otro collider para las **trigger collisions**.
* **Add component** -- **box collider**. Cambia la **y** y marca **is trigger**.
* Ve a **Player** script. Cambia **onControllerColliderHit** for **OnTriggerEnter.**
* Cambia **ControllerColliderHit Collision** por **Collider otherCollider** y **hit.collider** por **otherCollider**.
* Ve a **Unity**, aplica los cambios en el **Player**.

1. **Lógica para colisionar con el enemy.**

* **ShootingEnemy** contiene un **capsule collider**.
* **Is Trigger** está marcado como **false.**
* **El** **Bullet** prefab contiene un **sphere collider**.
* **is Trigger** está marcado como **true**.
* La **collision** entre la **Bullet** y el **Enemy** no sucederá porque al menos uno de estos elementos necesita tener un **Rigid Body**.
* Hacemos eso en el **ShootingEnemy**, seleccionandolo -- **add component – RigidBody**.
* Ve al **Scripts** folder -- **Game** folder – Abre el **Enemy** script.
* Haz una lógica similar para el **Player**. Vuelve al **Enemy**.

1. **Enemy camina hacia el Player.**

* Hay mecanismos de **Inteligencia Artificial** incorporados para hacer fácil el trabajo con la **navigation** y **navmesh** (**navigation match**).
* Necesitamos decirle a **Unity** cuál es el **área** en la cual nuestro **Enemy** va a ser capaz de moverse.
* Necesitamos que el **Enemy** camine en el suelo. Necesitamos activar el **floor** como un **navigation area**.
* Ve a **Unity -- window – navigation - object -- check navigation static - bake –** radio y altura del **Agente**. **Agents** son los elementos que navegan aquí, en este caso los **Enemies**.
* Podemos definir el ángulo para pasar las pendientes, **bake** precompila para regenerar la información que nuestro **Enemy** va a usar para la **navigation**.
* Necesitamos decirle a **Unity** que nuestro **Enemy** va a ser un **navigational agent**. Selecciona **ShootingEnemy** -- add **component** – agrega el **nav mesh agent**.
* Ajusta la **base offset**.
* Necesitamos decirle a través del script hacia donde se tienen que mover los **Enemy**.
* Vamos a hacer que se muevan alrededor hacia el **Player**.
* Necesitamos ir al **ShootingEnemy** script.
* Almacenar una referencia para el **navmesh**.
* Puedes poner alguna distancia entre el **Enemy**.
* El **Enemy** se detuvo porque estamos lejos de él. Si nos acercamos lo suficiente, va disparar y se va a mover hacia el **Player**.
* El destino del **Update** **method** solo se establece si el **Enemy** dispara. Si no lo hace entonces no irá hacia nosotros.
* Entonces crea otra distancia en el **ShootingEnemy** script.
* **Chasing distance** solo se establece si el agente está dentro de la **chase distance**.
* Necesitamos otro temporizador (**timer**).

1. **Asegúrate de que no vamos a perder el ShootingEnemy**.

* Arrastra y suelta al **Enemy** como un **prefab**.
* Cada vez que matamos al **Enemy** (**health = 0**) necesitamos l **ShootingEnemy** para que este caiga a tierra.
* En el **Enemy** script, está la lógica para llamar al **OnKill** **method** si la **health** es menor o igual que 0.
* En el **ShootingEnemy** script, necesitamos que el **Enemy** se caiga una vez que sea matado.
* Agrega más **Enemies**. En el **GameController** – crea un empty object – renómbralo como **EnemyContainer**. Arrastra el **shooting** **Enemy** dentro del **EnemyContainer**. Haz Clic derecho – duplica varias veces.
* Necesitamos crear una lógica cuando matamos a todos los **Enemies** de modo que pasemos de nivel.
* En el **GameController** -- **canvas** -- **2d**. necesitamos una referencia al **EnemyContainer**.
* Necesitamos contar cuántos **Enemies** tenemos.
* En el **GameController** script, ve a **Unity** en el **canvas** crea el **Enemy text**.
* Ve al **GameController**. Arrastra el **Enemy** **text** y suéltalo en el inspector.
* Selecciona **EnemyContainer** y arrastra y suéltalo en el inspector.
* Necesitamos actualizar a los **Enemies**.
* Ve al **Enemy** script.
* Vuelve al **GameController** para contar y ver cuántos **Enemies** fueron muertos.
* Necesitamos verificar si tenemos 0 **Enemies** vivos. Esto significa que hemos completado el nivel.
* Escribe una lógica para mostrar el mensaje **Ganaste**.
* Agrega un **text** en **canvas** posiciónalo en 0 en x y 0 en y.
* Aumenta su lado.
* En el **GameController** necesitamos una referencia a aquel **infoText**.
* Voy a abrir el **GameController** script.
* Al inicio del juego, necesito esconder el **infoText**. Arrastro el **infoText** en el **infoText** inspector.

1. **Lógica para cuando el Player muere**.

* Para hacer que el **Player** pierda necesitamos incrementar el daño que las **bullets** están ocasionando.
* Selecciona el **Bullet prefab**.
* Aumenta el **damage** a 20.
* En el **GameController** script verifica si el **Player** ha sido muerto.
* En el **Enemy** script ncesitamos hacer que el **Enemy** pare de moverse o pare de disparar si ellos notan que el e **Player** ha sido destruido.
* Ve al **GameController** – selecciona el **Enemy Container**.
* Elige uno de los **Enemies**.
* Una vez que ellos caen podríamos desactivar el **RigidBody**.
* Si se para, no necesitamos aplicarle más la física. Si matamos un **Enemy** y este cae al suelo. Si caminamos sobre este podemos todavía ser heridos lo que no queremos.
* Cada vez que somos heridos por el **Enemy** podemos verificar si ellos están muertos.
* De manera similar verifica para la **Bullet.**
* Ve al **Player** script. Si un **Enemy** ha sido muerto este no podrá herir más al **Player**.
* Si el **Player** es destruido en el **ShootingEnemy** script necesitamos definir dónde verificar si el **Player** ha sido muert**o**.
* Una vez que el **Player** está muerto no necesitamos ser capaces de disparar **Bullets** porque estamos muertos, y no seríamos capaces de movernos.
* Si desactivo el **CharacterController** no me puedo mover.
* Si quitamos el **First-Person Controller** dejaríamos de mirar alrededor.
* Ve al **Player** script y repasa la lógica donde disparamos **Bullets**.

1. **Haz un Menu**.

* **File -- new scene -- file -- save scene**. Nómbralo como **Menu**.
* Mueve este archivo **Menu** al **Scenes** folder.
* En el **Menu hierarchy** agrega **canvas**.
* Elije **2d** – agrega un **text**.
* Renómbralo como **FPS text**.
* Selecciona **camera** y cambia el **color** si deseas.
* En **canvas** -- agrega un **button**.
* Cambia el **text** a **play**.
* Si hacemos clic al **play button** deberíamos ir a la escena **Level1**.
* En el **Scripts** folder -- crea un **Menu** folder.
* Dentro del **Menu** folder crea un **MenuController** script.
* En la **hierarchy** crea un **empty object** y posiciónalo en **0 0 0**.
* Renómbralo como **menu controller**.
* Pon la **main camara** y la **directional light** dentro del **menu controller**.
* Arrastra y suelta el **MenuController** script dentro del **MenuController** inspector.
* Entra en el **MenuController** script.
* Ve a **Unity** – selecciona el botón en las **canvas**.
* En el inspector hay una sección **On CLick()**. Aqui Podemos agregar diferentes **callbacks** o **methods**. Estos pueden ser llamados cuando se hace clic en estos **button**.
* Haz Clic en el **plus sign**.
* Arrastra y suelta el **MenuController** aqui.
* Selecciona la **callback** -- **MenuController** – **OnPlay**. **Files** - **build settings** -- agrega **open scenes** -- agrega **Menu** scene.
* Guardar e ir a **Level1**. Y haces lo mismo. **Window -- lighting -- scanned -- disable auto – build**.
* Si ganamos o perdemos el juego volvemos al **Menu**.
* Ve a **GameController** – agrega el **scene manager namespace**.

1. **Crea un HealthCrate**.

* Duplica **ammocrate** en el **Prefabs** folder **ctrl + d**.
* Renómbralo como **healthcrate**.
* Arrastra y sueltálo en la escena.
* Crea materiales para aquel.
* Crea el material and cambia su color y asignaselo.
* En el **scripts folder** -- **Game** folder -- crea **healthcrate** script.
* Ábrelo – es similar al **ammocrate** script. Solo cambia **ammo** por **health**.
* En **Unity** arrastra y suelta el **healthcrate** script en el **healcrate** inspector.
* Borra el **ammocrate** script para este.
* Pon el **Container** del **healthcrate** en el **Container** del **healthcrate** script en el inspector.
* Haz clic en aplicar los cambios. Ve al **Player** script.

1. **Incluye Assets**.

* Un archivo de audio para **disparar el arma** y otro para cuando el **Enemy** muere.
* Agrega un **character** model para el **Enemy**.
* Agrega **building**.
* Agrega **weapons**.
* Ir a **Bullet** **prefab,** arrastra y suéltalo a la **hierarchy**.
* Agrega **component**, **audio source**.
* Ve al **Player** en la **hierarchy**, haz click en el **First Person Character**.
* Hay un **audio listener**, este sería usado para escuchar sonidos.
* A cada momento tendríamos solo un **audio listener** activo al mismo tiempo.
* Éste usualmente viene con la **camera**.
* En la **Bullet** **hierarchy,** chequea la **audio option**: **Play On Awake**.
* Este se va a reproducir tan pronto como aparezca.
* Abre el **Audio** folder, arrastra y suelta el **shoot** **audio file** en el **AudioClip** del inspector.
* Presiona aplicar.
* Bórralo de la **hierarchy**.

1. **Reproduce el audio cuando el Enemy muere**.

* Selecciona uno de los **Enemies** en nuestra escena.
* En **hierarchy** dentro del **ShootingEnemy**, crea un nuevo **game object**.
* Nómbralo **DeathSound** el el inspector.
* Agrega **component,** agrega **audio source**, arrastra y suelta el **dying sound** en el **Audio Clip**.
* El **ShootingEnemy** necesita una referencia a esa **audio source**.
* Abre el **Enemy** script.
* En **hierarchy**, en **ShootingEnemy**, arrastra y suelta el **DeathSound** en el **DeathSound** del inspector.
* Aplica para que los otros **prefabs** obtengan ese sonido.
* Desactiva **Play on Awake** en el inspector del **ShootingEnemy**.
* El audio se va a reproducir cuando el **Enemy** muera.

1. **Cambia al Enemy**.

* Selecciona un **Enemy**.
* Dentro de **Container** en lugar de **capsule**, ve a **Character** **models**.
* Selecciona un **Character** **model**
* Arrastra y suéltalo dentro de **Container**.
* Ajusta la escala y posición.
* Borra la **capsule**.
* Elige el **ShootingEnemy** y aplica.

1. **Agrega las Weapon, la Bullet y los edificios**.

* En la **hierarchy** ve a **Gun,** ve a **weapons** y elige un modelo.
* Arrastra y suelta dentro de **Gun** en la **hierarchy**.
* Ajusta la escala y la posición.
* Borra los modelos previos.
* Selecciona el **Bullet prefab**, arrastra y sueltalo en el **GameController**.
* Abre la **Bullet**, abre el **Container** y reemplaza aquella **Sphere** con el modelo **Bullet**.
* Arrastra y suéltalo en el **Container**.
* Escálalo y rótalo.
* Arrastra y suelta los **building models**.
* Si no queremos que los **Enemies** accedan a aquellos modelos, ve a **navigation**.
* En la **hierarchy** y expande los **building** **models**.
* Marca los **mesh elements**.
* Marca la opción **Navigation static** en el **navigation** object.
* En el **bake** menu presiona **bake** again.
* Asegúrate que el **floor** tenga chequeada la opción **navigation static**.
* Presiona **bake**, las áreas blancas nos muestran dónde el **Enemy** no puede navegar.
* Queremos colisionar con edificios para evitar pasarlos.
* Selecciona **mesh** en el inspector.
* Agregar **component**, agregar **collider**, **mesh collider**.
* Selecciiona los otros y aplica **mesh collider** también.