****

Módulo Profesional 17: Programación de Videojuegos 2D **M08UF3 - Act1**

**Tower Defense**

**CICLO FORMATIVO DE GRADO SUPERIOR EN**

**Videojuegos y ocio digital**

**MODALIDAD PRESENCIAL**

**Jorge Luis Garcia Orbegoso**



**Introducción**



Realiza un juego de estilo Tower Defense con Unity.

|  |
| --- |
| Objetivos |
| Practicar el uso de Splines para definir caminos (con o sin alternativas).  Utilización de Scriptable Objects y extensiones del Editor de Unity.  Utilizar diferentes maneras de interacción con el dispositivo móvil.  Generar múltiples interfaces (móvil / Tablet / sobremesa) dentro del mismo proyecto.  Crear sistemas de oleadas.  Aprender a generar terrenos en Unity. |
| **Metodología de evaluación** |
| |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Ejercicio 1 | Ejercicio 2 | Ejercicio 3 | Ejercicio 4 | Ejercicio 5 | Ejercicio 6 | Ejercicio 7 | Ejercicio 8 | Extras | | 5% | 10% | 15% | 15% | 15% | 5% | 10% | 10% | 15% |   Todos los ejercicios no opcionales tienen que estar aprobados para poder aprobar la actividad. |
| **Entrega** |
| A través del campus con nombre M08UF3-ACT1-NombreApellidos.zip incluyendo enunciado, build, proyecto de Unity, APK y vídeo de la prueba en el dispositivo.  Importante:   * Si falta cualquier entregable la nota es 0 * Por cada día de retraso en la entrega se resta un punto |
| **Documentos de referencia** |
| * [Defense of the Ancients - Wikipedia](https://en.wikipedia.org/wiki/Defense_of_the_Ancients#:~:text=DotA%20has%20its%20roots%20in,its%20expansion%2C%20The%20Frozen%20Throne.) * [DoTA - Como jugar (Guía para Novatos) (youtube.com)](https://www.youtube.com/watch?v=VdjKdqEko2o) * [Kingdom Rush (3 Stars)【FULL GAME】| Longplay (youtube.com)](https://www.youtube.com/watch?v=dZ56Y2FYYU0) * [Unity - Scripting API: ScriptableObject (unity3d.com)](https://docs.unity3d.com/ScriptReference/ScriptableObject.html) * [Unity - Manual: Coroutines (unity3d.com)](https://docs.unity3d.com/Manual/Coroutines.html) * <https://assetstore.unity.com/> * [Terrain Sample Asset Pack | 3D Landscapes | Unity Asset Store](https://assetstore.unity.com/packages/3d/environments/landscapes/terrain-sample-asset-pack-145808) * [Poly Pizza: Free 3D models for everyone](https://poly.pizza/) * <https://www.mixamo.com/> |

**Situación**

Desde que tu último jefe se retiró con la frase “soy demasiado rico, me voy al Tibet a alimentarme espiritualmente” tu empresa ha trabajado en lo que en el argot de programación se conoce como el modo “pollo sin cabeza”.

Por suerte, el grupo de inversión multimillonario que compró la empresa por una cantidad indecente de la que no has visto ni un duro, ha contratado al mejor CEO que han podido encontrar.

En positivo: es un gran CEO – podríamos decir que es muy-muy CEO: es CEísimO. En negativo: en 2002 este también se retiró al Tibet y lleva 20 años alimentándose espiritualmente mientras los monjes se ponen de 5 Jotas a su costa.

En todo este tiempo, su único entretenimiento ha sido modear el único juego que se llevó al templo: el Warcraft III. Lo ha modificado de mil maneras, y ahora ha vuelto con la mejor idea del mundo: una modificación nunca vista que pone a los edificios, y no a las unidades en el centro de la acción.

La ha llamado “Defender con Torreones”.

Nadie en la empresa se atreve a explicarle que esto ya existe, que primero fue el DOTA y después se convirtió en un género – así que te toca programar tremendo Tower Defense.



**Descripción del Juego**



El juego está dirigido a móvil y tablet. Consiste en un escenario 3D con vista cenital con temática libre sobre el cual hay los siguientes elementos:

* Caminos
* Cimientos para Torres
* Uno o varios puntos de entrada
* Uno o varios puntos de salida
  + Los puntos de entrada y de salida están obligatoriamente unidos por caminos

Incluir imagen demostrativa del resultado esperado

**Controles**

El juego tiene que funcionar en dispositivos móviles, específicamente en Android. Se interactuará con el juego a través de diferentes gestos en la pantalla táctil.

**Tropas Enemigas**

La partida se desarrollará a lo largo de varias oleadas. Durante una oleada, se generarán Tropas Enemigas en los puntos de entrada, que decidirán de antemano el camino a recorrer para llegar a un punto de salida. Tanto los puntos de entrada como los puntos de salida se escogerán al azar.

**Torres**

El jugador dispondrá de las siguientes acciones:

* Construir una nueva torre sobre un cimiento
* Mejorar una torre existente
* Reparar una torre existente
* Activar los bufos de una torre
* Vender una torre

Habrá tres tipos de torres con tres mejoras posibles cada una

* Tipos:
  + Arqueros
  + Magos
  + Cañoneros
* Mejoras
  + Inicial
  + Mejorada
  + Superior

|  |
| --- |
| Como veremos más adelante, habrá que preparar un prefab TorreBase, con tres Variants por cada tipo que tendrán a su vez tres Variants por cada mejora. |

**Proyectiles**

El intercambio de daño entre Enemigos y Torres tendrá lugar a través de Proyectiles. Tanto los Enemigos como las Torres lanzarán Proyectiles a sus rivales en rango.

Los proyectiles podrán hacer daño directo, daño de área, e incluso generar sub-proyectiles.

|  |
| --- |
| Como veremos más adelante, habrá que preparar un prefab ProyectilBase, con variants para todo:   * Tres Variants por cada tipo de Torre que tendrán a su vez tres Variants por cada mejora. * Una variant por cada tipo de Enemigo |

**Mecánica General y Progreso del Juego**

* El juego está organizado en niveles
  + Cada nivel se juega de una sentada
    - No se puede salvar entre medias
* El objetivo principal en todos los niveles es sobrevivir
  + El jugador dispone de un indicador de resistencia
    - Este indicador de resistencia se reducirá cada vez que una Tropa Enemiga consiga alcanzar un punto de salida
    - Si el indicador de resistencia llega a cero, la partida termina con resultado de derrota
  + La partida termina con resultado de victoria cuando la última tropa (generable y generada) desaparezca del mapa
    - “Generada” para contar las tropas que ya hay en el mapa
    - “Generable” para contar las tropas que aún quedan por generar
* Durante un nivel, las Tropas Enemigas aparecerán en los puntos de entrada y recorrerán caminos al azar hasta los puntos de salida
  + La aparición de las tropas se organizará en oleadas
    - Habrá entre 3 y 5 oleadas por nivel, definidas por
      * Tiempo de aparición
      * Cantidad de enemigos
      * Tipo de enemigos
      * Frecuencia de aparición durante la oleada
* El jugador podrá disponer de manera estratégica las Torres
  + El jugador podrá consumir recursos para construir torres sobre los cimientos
  + Las torres lanzarán proyectiles a los enemigos, y recibirán proyectiles por parte de éstos, provocando muerte, pero también pudiendo llegar a destruirse
  + El jugador podrá realizar varias acciones sobre las torres:
    - Mejorarla
    - Repararla
    - Venderla

**Ejercicio 0: Antes de Empezar**

Importa los siguientes paquetes de Unity que se utilizarán durante el desarrollo del juego:

* **InputSystem –** utiliza InputSystem para responder
* **Cinemachine –** en este caso no es el asset más importante, pero es conveniente acostumbrarse a utilizarlo
* **Splines –** esteasset nos permitirá dibujar una serie de curvas; en este juego servirán principalmente para definir el camino que han de seguir los enemigos

Instala también el siguiente paquete, que tiene una versión Free disponible en el la AssetStore de Unity:

* **DOTween –** este paquete permite crear animaciones sencillas por código, lo utilizaremos para crear el movimiento de cámara

**Importante:** Dado que el juego tiene que funcionar para móvil, y tiene que haber una gran cantidad de unidades, procura elegir assets 3D que tengan el poligonaje lo más bajo posible. Utiliza también técnicas de optimización, como:

* utilizar un único material por personaje
* o materiales muy livianos (solo color, sin textura)
* o compartir material entre modelos

**Ejercicio** **1: Preparación del Campo de Batalla**

En este primer ejercicio hay que realizar las siguientes tareas:

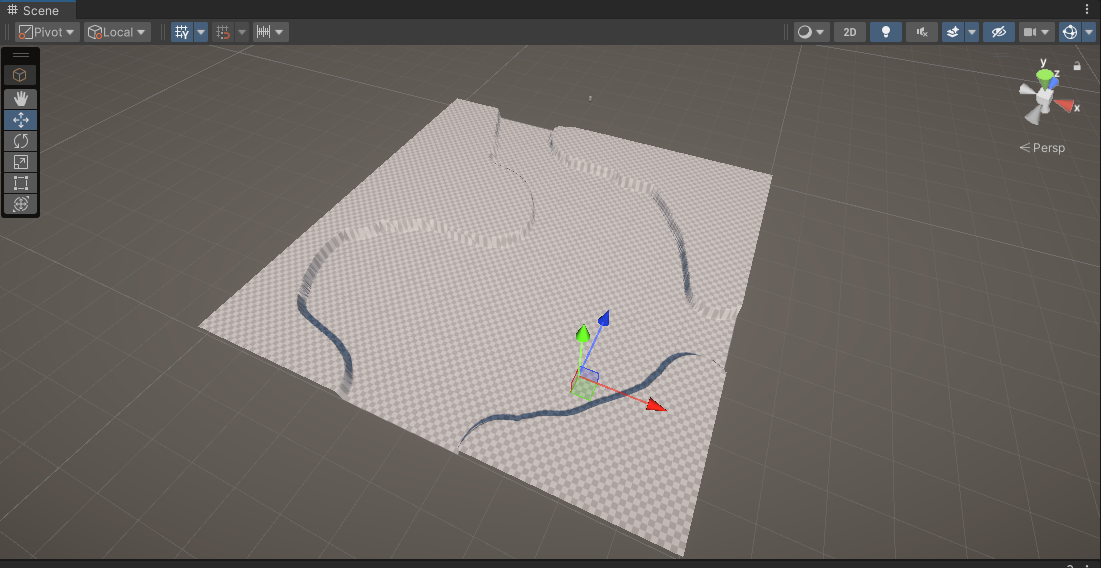
* Preparar un terreno con el Terrain de Unity
* Preparar y distribuir los Cimientos de las Torres
* Preparar los Caminos para las tropas enemigas

**Preparando el Terreno**

Prepara un terreno:

* Crea un terreno de Unity
* Reduce su tamaño a un cuadrado de 100x100m
* Genera una topología que
  + Tenga 3 puntos de entrada
  + Tenga 3 puntos de salida
  + Tenga caminos que unan los puntos de entrada y de salida
    - Dichos caminos tienen que ser sinuosos para dar la oportunidad a las torres de que ataquen a los enemigos que pasan por ellas

El resultado tiene que ser parecido al de la siguiente imagen:

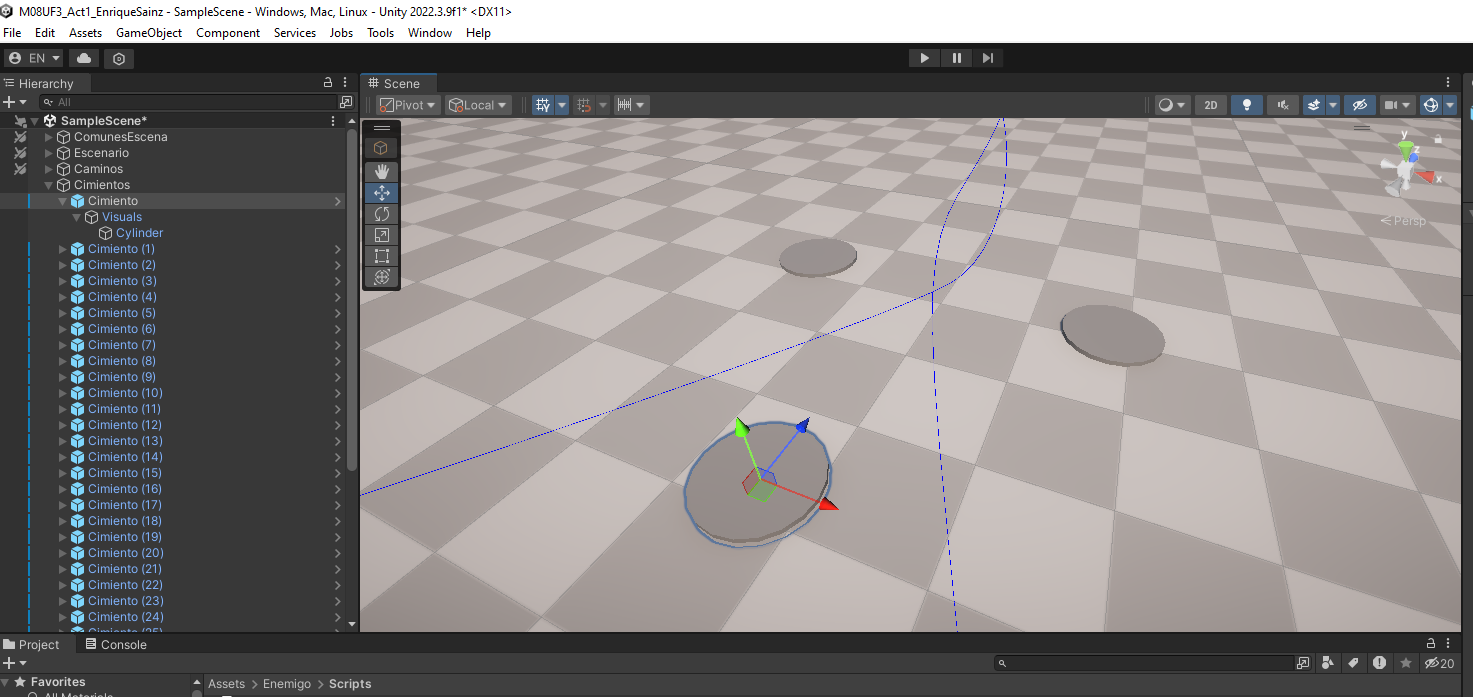


**Preparando los Cimientos**

Inicialmente, los cimientos sólo tienen que ser instancias de un prefabricado al que en ejercicios posteriores le añadiremos la lógica adecuada:

* Crea un GameObject vacío, llámalo Cimiento
* Crea un hijo vacío de Cimiento, llámalo Visuals
* Crea un Cilindro 3D, hijo de Visuals
  + Quítale la colisión
  + Aplánalo para que parezca una base sobre la que colocar otros objetos
* Convierte Cimiento en un prefabricado
* Instancia varios Cimientos en diferentes lugares del mapa

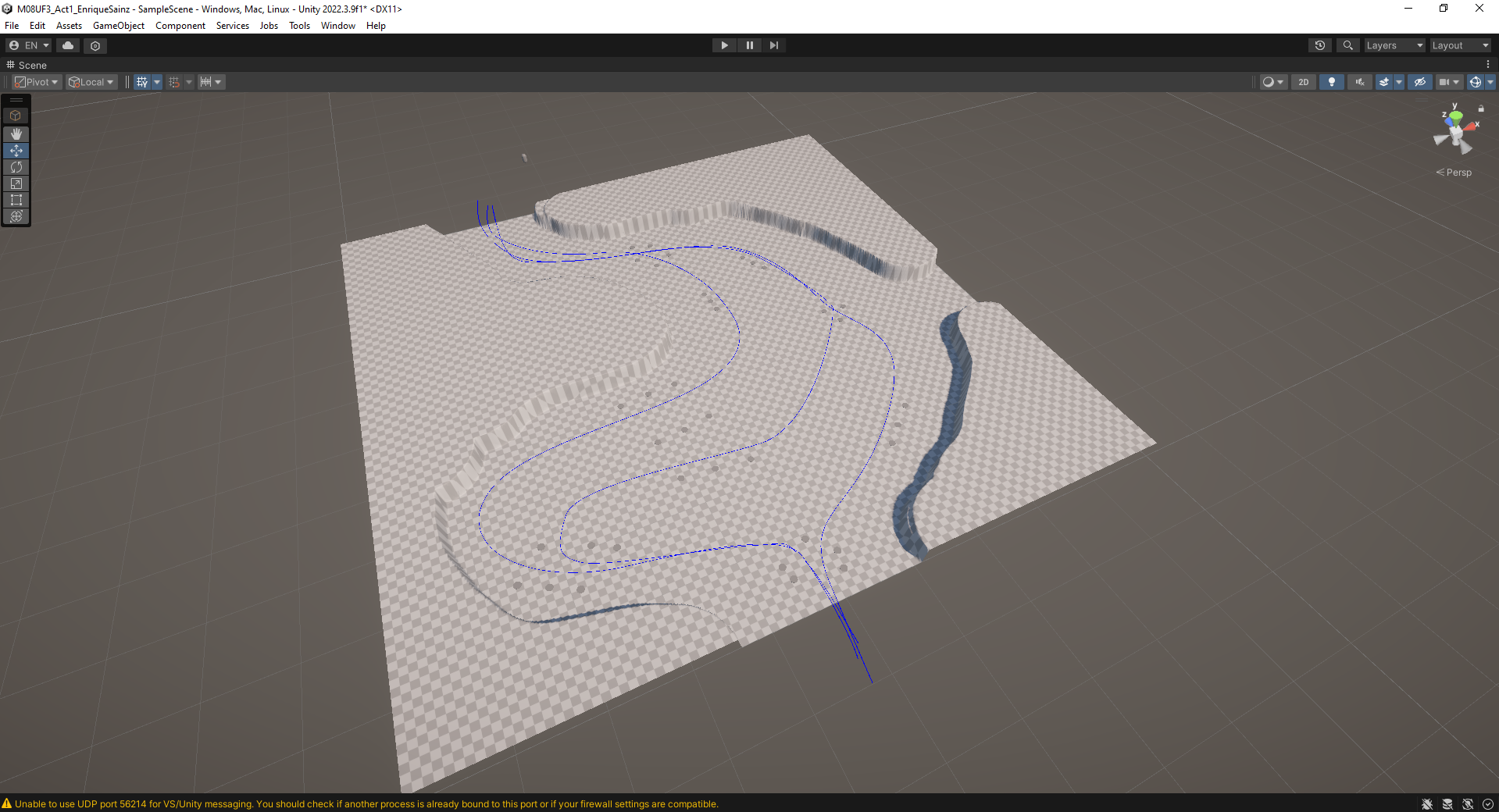
El resultado tiene que ser parecido al de la siguiente imagen:



**Preparando los Caminos**

Definiremos las rutas de los enemigos a través de una serie de Splines.

* Utiliza una Spline para definir caminos que conecten todos los puntos de entrada con todos los puntos de salida
  + Cada spline tiene que estar en un único GameObject



**Ejercicio 1:** Incluye a continuación un enlace a un vídeo en el que demuestres el funcionamiento del ejercicio realizado.

|  |
| --- |
|  |

**Ejercicio 2: Preparación de los Enemigos Básicos**

En este segundo ejercicio, tendrás que preparar los Enemigos Básicos.

**Resumen del Comportamiento**

Hay enemigos de tres tipos:

* Rápidos pero con poco aguante
* Lentos pero con mucho aguante
* Mediocres (en velocidad y en aguante)

Los enemigos se comportarán de la siguiente manera:

* Eligen uno de los caminos definidos completamente al azar
* Lo recorren hasta el final
  + Por el camino pueden recibir daño procedente de las torres
  + Si reciben suficiente daño, morirán

**Implementación**

* Prepara la Clase Enemigo
  + Implementa las siguientes propiedades modificables por Inspector:
    - Ruta (SplineContainer)
    - Velocidad (flotante, metros por segundo)
    - Vida (flotante)
  + Implementa el siguiente comportamiento:
    - Antes de empezar
      * Cachea de los puntos de ruta
        + Evaluar los puntos de una spline no es tan caro, pero que cientos de Enemigos lo hagan a cada fotograma sí que es caro
        + Por eso deberás crear una cache de puntos de ruta con una distancia fija entre ellos que se podrá establecer por enemigo
        + Guárdala como un array de Vector3, llámalo Vector3[] pathPoints
    - Una vez empezado
      * Movimiento
        + Al inicio, coloca al enemigo en pathPoints[0]
        + Establece un Vector3 nextPoint que, inicialmente, tendrá el valor de pathPoints[1]
        + En cada fotograma, avanza en línea recta hacia nextPoint

Asumimos que la ruta está bien hecha y es estática, y que no habrá obstáculos en el camino

* + - * + Cuando el enemigo esté cerca del nextPoint

Si se trataba del punto final, el enemigo desaparece

Si se trataba de un punto intermedio, se reasigna nextPoint al siguiente punto que tocaría

* Prepara un Enemigo
  + Crea un objeto EnemigoBase que tenga:
    - Visuals
      * Una capsula
  + En el objeto EnemigoBase, añade el la clase Enemigo
  + Asigna el enemigo a una de las rutas
  + Establece una velocidad de 4
  + Establece una vida de 2 puntos
  + Comprueba que la clase Enemigo es capaz de desplazar al EnemigoBase a lo largo de toda la ruta
* Prepara tres Prefabricados
  + Crea un prefabricado con el objeto EnemigoBase
  + Crea tres Variants:
    - EnemigoRapido
      * Velocidad 6
      * 1 punto de vida
      * Escala la cápsula a 0,75
    - EnemigoNormal
      * Es igual que el enemigo base
      * Pero haremos la Variant para poder cambiar el gráfico en el futuro
    - EnemigoLento
      * Velocidad 2
      * 20 puntos de vida
      * Escala la cápsula a 2

**Ejercicio 2:** Realiza un video mostrando los progresos hasta este punto.

|  |
| --- |
|  |

**Ejercicio 3A: Oleadas – Preparación del ScriptableObject**

Una oleada consiste en una cierta cantidad de enemigos que aparecerán más o menos agrupados y recorrerán los caminos.

Utilizaremos ScriptableObjects para definir las oleadas. Esto no es 100% necesario, ya que los prefabricados también valdrían, pero de este modo practicaremos el uso de ScriptableObjects.

**Preparación del ScriptableObject**

* Crea un nuevo C# Script con nombre DefinicionOleada
  + Sustituye la clase que aparece por defecto por una clase DefinicionOleada que derive de ScriptableObject
    - Ponle el atributo [CreateAssetMenu]
      * Esto permitirá crear assets de este tipo desde el Menú
  + Crea una clase BloqueEnemigos dentro de la clase DefinicionOleada
    - Ponle el atributo [System.Serializable]
      * Esto hará que C# reconozca la clase como serializable, lo que permitirá que se puedan guardar los valores a disco y editarlos desde el inspector
    - Añade las siguientes variables públicas:
      * TipoEnemigos – GameObject (servirá para referenciar Prefabricados)
      * Cantidad – entero
      * EnemigosPorSegundo – flotante
  + Crea un array de BloqueEnemigo en DefinicionOleada, llámalo bloques
  + El ScriptableObject está acabado
* Comprueba que todo funcione correctamente
  + En el editor de Unity, haz click derecho sobre la ventana de Proyecto
    - Elige Create 🡪 DefinicionOleada
  + Comprueba que se ha creado una nueva DefinicionOleada, y que puedes declarar en ésta tantos bloques como quieras

Incluir imagen demostrativa del resultado esperado

**Ejercicio 3A:** Realiza un video mostrando los progresos hasta este punto.

|  |
| --- |
|  |

**Ejercicio 3B: Oleadas – Generación de Enemigos**

Las oleadas tendrán lugar de manera secuencial:

**Tiempo de Espera 🡪 Primera Oleada 🡪 Tiempo de Espera 🡪 Segunda Oleada 🡪 etc…**

Con las oleadas definidas gracias al ScriptableObject, pasaremos ahora a definir tanto el objeto que gestionará las oleadas como la estructura de datos que definirá el guion del funcionamiento entre oleadas.

**Preparación del Guión de Oleadas**

* Crea un nuevo C# Script con nombre GestorOleadas
  + Crea una clase LineaGuion dentro de GestorOleadas
    - Añade el atributo [System.Serializable]
    - Añade los siguientes miembros públicos
      * Espera, de tipo flotante
      * oleada, de tipo DefinicionOleada
  + Crea una clase GuionOleadas dentro de GestorOleadas
    - Añade el atributo [System.Serializable]
    - Añade los siguientes miembros públicos
      * líneas, será un array de LineaGuion
  + En GestorOleadas, añade un campo serializable de tipo GuionOleadas
    - Comprueba que puedes añadir líneas de guion

Incluir imagen demostrativa del resultado esperado

**Ejecución de las Oleadas**

* Prepara una corrutina con nombre LeeGuion
  + Utiliza un bucle para recorrer todas las líneas de guion. Para cada línea…
    - … utiliza un WaitForSeconds() para realizar la espera de la siguiente oleada
    - … lanza la corrutina LanzaOleada una vez haya pasado ese tiempo
* Prepara una corrutina con nombre LanzaOleada
  + Tiene que recibir por parámetro la DefinicionOleada
  + Prepara un bucle para recorrer todos los BloquesDeEnemigos. Para cada bloque prepara un bucle que…
    - … espere el tiempo adecuado según los enemigosPorSegundo
    - … elija al azar un camino en el mapa
    - … instancie un enemigo con el prefabricado especificado
      * … en la posición inicial del camino elegido
    - … busque el componente Enemigo en éste
      * … asigne el camino elegido a dicho componente
* Ejecuta LeeGuion desde el Start()

**Comprobación del Funcionamiento**

* En la escena, añade un GestorOleadas
  + Prepara un guion de dos líneas con dos oleadas
    - Puedes crear dos assets de tipo DefinicionOleada
    - … o puedes crear uno solo y utilizarlo dos veces
* Comprueba que todas las oleadas tienen lugar en los tiempos que has especificado en la DefinicionOleada y en las LineaGuion

Incluir imagen demostrativa del resultado esperado

**Ejercicio 3B:** Realiza un video mostrando los progresos hasta este punto.

|  |
| --- |
|  |

**Ejercicio 4: Proyectiles**

Los ataques de este juego se harán utilizando proyectiles. Los proyectiles son objetos de baja velocidad definidos por:

* **Colisionador esférico –** todos los proyectiles tienen esta forma por eficiencia de cálculo
* **Pueden hacer daño por impacto directo –** opcionalmente, un proyectil puede hacer daño directo por impacto
* **Pueden hacer daño por área –** opcionalmente, un proyectil puede hacer daño de área (splash damage)
* **(Opcional) Pueden generar nuevos proyectiles –** opcionalmente, un proyectil generará proyectiles secundarios al impactar contra un enemigo o contra el suelo
  + Los proyectiles secundarios nunca generan nuevos proyectiles

Todos los proyectiles tienen un punto de partida y un punto de llegada, y hacen un movimiento más o menos parabólico.

**Prepara la interfaz IGolpeable**

* Esta Interfaz será la que el Proyectil utilice para transmitir el daño
* Dispone de una única función:
  + RecibeDanyo(float cantidad)

**Prepara la clase Proyectil**

* Añade las siguientes propiedades:
  + **danyoImpactoDirecto –** el daño que realiza a una unidad cuando esta recibe impacto directo
  + **radioDanyoArea –** indica la distancia desde el punto en el que se destruyó el proyectil dentro de la cuál aplicaremos el daño de área
  + **danyoAreaEnOrigen –** el daño de área que recibiría una unidad estando esta justo en el centro del área
  + **danyoAreaEnBorde –** el daño de área que recibiría una unidad estando justo al borde del área
  + **subProyectilesAGenerar –** la cantidad de subproyectiles a generar
  + **radioSubProyectiles –** distancia máxima a la que puede caer el subProyectil
  + **prefabSubProyectil –** prefabricado a utilizar como SubProyectil
  + **tagsAfectados[] –** se trata de un array de strings que indica qué tags se utilizan para detener al proyectil y para realizar el daño de area
* Añade las siguientes funciones:
  + **Init(puntoInicial, puntoFinal, alturaSalto) –** pone en marcha el proyectil
    - Utiliza DOTween para realizar el movimiento
* Implemente OnTriggerEnter
  + Comprueba si el objeto que colisiona tiene uno de los tags en tagsAfectados
    - En caso afirmativo:
      * Intenta obtener la interfaz IGolpeable
        + Si la obtienes, transmite el daño por una cantidad de danyoImpactoDirecto
      * Destruye el proyectil
* Si el proyectil acaba su recorrido, destruye el proyectil
* Si el proyectil se destruye por cualquier motivo, calcula el daño de área sobre otros objetos cercanos
  + Utiliza OverlapSphere() para obtener todos los colliders cercanos
  + Para todos los que tengan uno de los tags en tagsAfectados[], intenta obtener la interfaz IGolpeable
    - Si la obtienes, utiliza Mathf.Lerp() para calcular el daño que tiene que recibir según la distancia al centro del área
* Opcionalmente, genera los subProyectiles, decidiendo un punto de aterrizaje al azar dentro de la distancia
  + Si el Proyectil original impactó directamente a un collider, utiliza Physics.IgnoreCollision () para que los subProyectiles no impacten contra dicho collider nada más aparecer

**Prepara el Debugging para la clase proyectil**

* Añade las siguientes propiedades:
  + **debugTransformPuntoFinal –** es una referencia a un objeto que podemos utilizar para obtener un punto de destino
  + **debugTransformInit –** es el booleano que se utilizará en OnValidate para llamar al Init() del proyectil
    - Utiliza la posición actual como punto de inicio
    - Utiliza la posición de debugTransformPuntoFinal como punto final

**Prepara el Proyectil Base**

* Prepara el prefabricado ProyectilBase
  + Prepara una esfera de 0.1 de radio con un trigger collider
    - En el futuro modificaremos el radio del collider para que acierte más si fuera necesario
  + Prepara un material ProyectilBase, Unlit y Magenta, y asígnalo
  + Añade la clase Proyectil
  + Añade un componente Rigidbody en modo isKinematic
    - Esto es necesario para que los proyectiles puedan colisionar contra los enemigos
  + Conviértelo en un prefabricado

**Prepara las Variants para todos los Proyectiles del Juego**

* Utiliza la siguiente tabla para realizar todas las Variants

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Variant a Realizar** | Proyectil Enemigo | ProyectilEnemigoRapido | ProyectilEnemigoNormal | ProyectilEnemigoLento | FlechaBase | FlechaLv0 | FlechaLv1 | FlechaLv2 | FlechaSubProyectil | BombaBase | BombaLv0 | BombaLv1 | BombaLv2 | BombaSubProyectil | MagiaBase | MagiaLv0 | MagiaLv1 | MagiaLv2 | MagiaSubProyectil |
| **Variant Original** | ProyectilBase | Proyectil Enemigo | Proyectil Enemigo | Proyectil Enemigo | ProyectilBase | FlechaBase | FlechaBase | FlechaBase | FlechaBase | ProyectilBase | BombaBase | BombaBase | BombaBase | BombaBase | ProyectilBase | MagiaBase | MagiaBase | MagiaBase | MagiaBase |
| **Color** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Daño Impacto Directo** | -1 | 1 | 2 | 3 | -1 | 1 | 2 | 3 | 1 | -1 | 0 | 0 | 0 | 0 | -1 | 1 | 2 | 3 | 1 |
| **Radio Daño Area** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 3 | 4 | 1 | 0 | 0 | 2 | 10 | 30 |
| **Daño de Area en Origen** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0,5 | 1 | 2 | 0,25 | 0 | 0 | 0,25 | 10 | 3 |
| **Daño de Area en Borde** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0,25 | 0,5 | 1 | 0,25 | 0 | 0 | 0,25 | 10 | 3 |
| **SubProyectiles a Generar** | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 5 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 10 | 0 |
| **Radio SubProyectiles** | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 6 | 0 | 0 | 3 | 6 | 9 | 0 | 0 | 0 | 0 | 10 | 0 |
| **Prefab SubProyectil** | Null | Null | Null | Null | Null | Null | Null | FlechaSubProyectil | Null | Null | Null | BombaSubProyectil | BombaSubProyectil | Null | Null | Null | Null | MagiaSubProyectil  l | Null |
| **Tags Afectados** |  | Torre | Torre | Torre | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo |

**Arregla la Clase Enemigo**

* Arregla la clase Enemigo para que pueda recibir daño
  + Implementa la Interfaz IGolpeable en la clase Enemigo
    - Haz que el enemigo pierda la vida pasada por parámetro en RecibeDanyo()
    - Si la vida llega a 0, destruye el GameObject del enemigo

**Prepara una prueba**

* Crea varios Proyectiles cerca del inicio de una de las rutas
* Utiliza las variables de Debug para asegurar que el Proyectil cae en la ruta
* Utiliza algunos proyectiles para comprobar que el Proyectil se mueve
* Utiliza otros proyectiles para comprobar que hace daño por impacto directo
* Utiliza otros proyectiles para comprobar que hace daño de area

**Ejercicio 4:** Realiza un video mostrando los progresos hasta este punto.

|  |
| --- |
|  |

**Ejercicio 5: Torres: Acción**

En este ejercicio, prepararaemos las torres, y haremos que pueden atacar y ser atacadas.

**Ejercicio 5A: La Torre Base**

**Empieza la Torre Base**

* Prepara el Prefabricado
  + Crea un GameObject vacío TorreBase
    - Crea un GameObject Visuals dentro de TorreBase
      * Resetea su transformada
    - Dentro de los Visuals
      * Crea un cubo de (1x2x1)
      * Ponle al cubo una posición local de (0, 1, 0)
      * Prepara un Material TorreBase, Lit, de color Magenta
        + Aplícalo al cubo
    - Crea un GameObject vacío origenDisparo dentro de torre base
      * Colócalo en posición (0, 2.5, 0)
  + Conviértelo el GameObject Torrebase en el Prefab TorreBase

**Prepara el Componente Torre**

* Prepara la clase Torre
  + Esta clase deriva de MonoBehavior e implementa IGolpeable
  + Añade las siguientes variables
    - **Resistencia** – es un flotante que indica un valor inicial de resistencia
  + Implementa IGolpeable para que vaya disminuyendo el valor de resistencia
    - Si llega a cero, se destruye el GameObject que contiene el componente Torre

**Prepara el Componente Disparador**

* Prepara el Componente Disparador
  + Esta clase deriva de MonoBehavior
  + Añade las siguientes variables
    - **radioDeteccion** – es la distancia de visión que tiene la torre
    - **radioPerdidaObjetivo** – es la distancia a la que la torre deja de detectar un objetivo que esté vivo
    - **tagsObjetivos** – es un array de strings que contiene los tags que utilizar para distinguir a los objetivos a adquirir
    - **prefabProyectil** – es una referencia al prefabricado que se instanciará cada vez que
    - **tiempoEntreProyectiles** – es el tiempo entre disparos de la
    - **origenProyectil** – es una referencia a una transform que utilizaremos como punto de origen para los disparos que salgan de la torre
    - **alturaSaltoProyectil** – es la altura (en metros)que se pasará al proyectil para que la cumpla a la mitad de su recorrido hacia el objetivo
* Adquisición de objetivo
  + Prepara una variable interna **objetivo** de tipo Transform
  + Si no tienes objetivo seleccionado (es decir, si es diferente de null)…
    - Utiliza OverlapSphere con radio radioDeteccion periódicamente para detectar y adquirir objetivos potenciales
      * Alternativamente, puedes gestionar eventos de TriggerEnter/Exit, para saber qué Enemigos están más cerca
    - Utiliza el tag en tagsObjetivos para determinar cuales son los enemigos que están al alcance
    - Selecciona el más cercano
  + Si ya tienes objetivo…
    - Comprueba que aún esté dentro de radioPerdidaObjetivo
      * Si no lo está, pon objetivo a null
    - Si el objetivo muere, al destruir su GameObject el objetivo se pondrá a null por sí solo
* Disparo con cadencia
  + Prepara la variable tiempoParaSiguienteProyectil
  + Inicializala a cero
  + En el Update()
    - Reduce el tiempoParaSiguienteProyectil en Time.deltaTime
      * Asegúrate de que no baje de cero
    - Si no tienes objetivo seleccionado (es decir, si es diferente de null)…
      * Si tiempoParaSiguienteProyectil es cero o menor que cero…
        + Realiza el disparo

Instancia prefabProyectil

Obtén su Componente Proyectil

Llama a la función Init con los parámetros adecuados:

**puntoInicial** 🡪 la posición de la transform origenProyectil

**puntoFinal** 🡪 la posición del objetivo

**alturaSalto** 🡪 el valor de alturaSaltoProyectil

* + - * + Establece en tiempoParaSigueinteProyectil el valor de tiempoEntreProyectiles

**Completa la Torre Base**

* Añade el Componente Torre al Prefabricado
  + Pon un valor inicial de 10f a la resistencia
* Añade el Componente Disparador al prefabricado
  + Utiliza los valores iniciales que encontrarás en la tabla del Ejercicio 5B

**Comprueba el funcionamiento de la Torre Base**

* Instancia una TorreBase en cualquier parte del mapa
* Cerca de ésta, coloca un EnemigoBase

**Ejercicio 5B: Las Torres Especializadas**

Utiliza la siguiente tabla para crear las Variants de las torres:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Variant a Realizar** | TorreBase | TorreFlechaBase | TorreFlechaLv0 | TorreFlechaLv1 | TorreFlechaLv2 | TorreBombaBase | TorreBombaLv0 | TorreBombaLv1 | TorreBombaLv2 | TorreMagiaBase | TorreMagiaLv0 | TorreMagiaLv1 | TorreMagiaLv2 |
| **Variant Original** |  | TorreBase | TorreFlechaBase | TorreFlechaBase | TorreFlechaBase | TorreBase | TorreBombaBase | TorreBombaBase | TorreBombaBase | TorreBase | TorreMagiaBase | TorreMagiaBase | TorreMagiaBase |
| **Color** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Resistencia** | 10 | 10 | 10 | 20 | 30 | 30 | 30 | 40 | 50 | 5 | 5 | 10 | 15 |
| **radioDeteccion** | 5 | 5 | 5 | 5 | 5 | 10 | 10 | 10 | 10 | 20 | 20 | 20 | 20 |
| **radioPerdidaObjetivo** | 5 | 5 | 5 | 5 | 5 | 15 | 15 | 15 | 15 | 30 | 30 | 30 | 30 |
| **tagsObjetivos** | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo | Enemigo |
| **prefabProyectil** | ProyectilBase | FlechaBase | FlechaLv0 | FlechaLv1 | FlechaLv2 | BombaBase | BombaLv0 | BombaLv1 | BombaLv2 | MagiaBase | MagiaLv0 | MagiaLv1 | MagiaLv2 |
| **tiempoEntreProyectiles** | 0.5 | 0.5 | 0.5 | 0.4 | 0.3 | 2 | 2 | 2 | 1.5 | 1 | 1 | 1 | 1 |
| **alturaSaltoProyectil** | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 0.5 | 0.5 | 0.5 | 0.5 |

**Ejercicio 5:** Realiza un video mostrando los progresos hasta este punto.

|  |
| --- |
|  |

**Ejercicio 6: Completa los Enemigos**

Modifica los enemigos para que hagan daño

**Prepara el Disparador**

* Añade el componente Disparador al prefabricado EnemigoBase
* Modifíca todos los Prefabs añadiendo los datos en esta tabla a sus disparadores:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Radio Deteccion** | **Radio Perdida Objetivo** | **Tags Objetivos** | **Prefab Proyectil** | **Tiempo Entre Proyectiles** | **Altura Salto Proyectil** |
| **EnemigoBase** | 5 |  | Torre | Proyectil Enemigo | 1 |  |
| **EnemigoRapido** | 7 | 7 | Torre | ProyectilEnemigoRapido | 1.5 | 0.3 |
| **EnemigoNormal** | 5 | 6 | Torre | ProyectilEnemigoNormal | 3 | 0.5 |
| **EnemigoLento** | 3 | 5 | Torre | ProyectilEnemigoLento | 6 | 1.5 |

**Comprueba el Funcionamiento**

* Prepara un escenario con una única ruta
* Prepara una DefinicionOleada que lance 10 enemigos de cada tipo, en orden de más débil a más fuerte
* Prepara un manager de oleadas en cuyo guión se lance 10 veces dicha DefinicionOleada
* Instancia varias torres de cada tipo a lo largo de la ruta (no hace falta poner los cimientos)
* Comprueba que los enemigos también atacan a las torres, y que éstas reciben el daño

**Ejercicio 6:** Realiza un video mostrando los progresos hasta este punto.

|  |
| --- |
|  |

**Ejercicio 7: Torres: Gestión**

En este ejercicio se preparará:

* El funcionamiento de los cimientos para la construcción de nuevas torres
* La mejora, reparación y venta de las torres

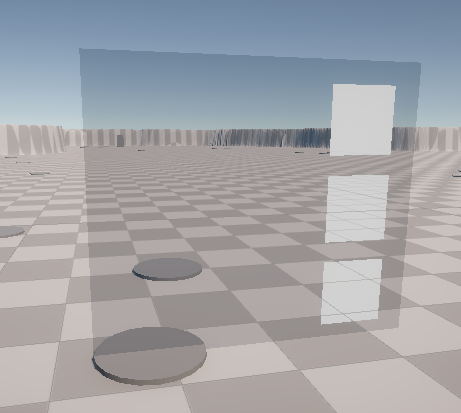
**Antes de Empezar…**

Para facilitar la jugabilidad, utilizaremos una cámara fija que proporcione visión sobre todo el mapa.

De manera **totalmente opcional**, puedes preparar una cámara controlable con las siguientes acciones:

* Panning 🡪 Arrastrar con dos dedos
* Zoom 🡪 Alejar o acercar dos dedos sobre la pantalla
* Rotación 🡪 Modificar la dirección entre dos dedos (rotar uno respecto al otro / o rotar ambos)

**Ejercicio 7A: Interacción con los Cimientos**

* **Interacción general con elementos 3D**
  + En la cámara principal, añade un componente PhysicsRaycaster
    - Este componente permitirá que los objetos 3D en la escena también reciban notificaciones de interacción (click, drag, etc.)
    - Asegúrate de añadir un EventSystem en la escena
  + En el cimiento, añade el componente EventTrigger
    - Este componente nos permitirá añadir funciones con las que responder a las notificaciones de interacción
* **Prepara la clase Cimiento**
  + Prepara una clase Cimiento
    - Añade las siguientes variables
      * torreConstruida – esta variable es de tipo Torre, y hace referencia a la torre que se pueda construir en futuros apartados
    - Añade las siguientes funciones
      * ConstruyeTorreArqueros() – Prepara un Debug.Log(), de momento
      * ConstruyeTorreMagica() – Prepara un Debug.Log(), de momento
      * ConstruyeTorreCañones() – Prepara un Debug.Log(), de momento
      * HayTorreConstruida() – devuelve cierto si ya hay una torre construida
* **Preparación de las clases para interactuar con elementos 3D**
  + Prepara la clase **MenuDesplegable**
    - Esta clase representa a un menú oculto que, al mantener pulsado un instante un objeto 3D, se visualiza y permite seleccionar una opción
    - La clase MenuDesplegable dispondrá de varias funcionalidades públicas que después mapearemos a eventos de TriggerEvent (del Cimiento o del Canvas)
    - Las funcionalidades a realizar son las siguientes
      * **EstableceEvento()** – recibe un UnityEvent y lo guarda en una variable privada llamada eventoParaLlamarAlOcultar
      * **Mostrar() –** Muestra el Canvas (activa el GameObject)
      * **Ocultar()** – Oculta el Canvas (desactiva el GameObject) y, si es diferente de null, invoca eventoParaLlamarAlOcultar; después lo pone a null
  + Prepara la clase **BotonAlSoltar**
    - Esta clase añade funcionalidad a una imagen de modo que ejecute una funcionalidad cuando se suelta el botón de ratón sobre esta
    - Dispone de algunas funcionalidades que mapearemos a eventos de TriggerEvent()
    - También dispone de un UnityEvent llamado “accion” que permitirá asignar el resultado a ejecutar
    - Las funcionalidades son las siguientes
      * **Awake()** – busca el MenuDesplegable que contiene al botón
        + Utiliza GetComponentInParent()
      * **EntraPuntero()** – Llama a MenuDesplegable.EstableceEvento() pasando accion como parámetro
      * **SalePuntero()** – Llama a MenuDesplegable.EstableceEvento() pasando null como parámetro
* **Preparación del Menú de los Cimientos**
  + Prepara el menú desplegable de los cimientos
    - Crea un nuevo Canvas de Mundo
    - Colócalo como hijo de los cimientos
    - Añade un panel semitransparente que cubra la torre, y que sobresalga por su derecha, como en esta imagen  
        
      
    - Añade tres imágenes cuadradas, a modo de iconos, para seleccionar el tipo de torre a crear
      * Añade el componente TriggerEvent a las tres imágenes para poder responder a las notificaciones de interacción
    - Añade el componente MenuDesplegable al Canvas
      * Mapea las llamadas de TriggerEvent en el Cimiento a las funciones de MenuDesplegable
        + **PointerDown** 🡪 Mostrar()
        + **PointerUp** 🡪 Ocultar()
    - Añade el componente BotonAlSoltar a las imágenes/iconos
      * **Consejo:** aprovecha para hacer un único prefábricado, y así poder reutilizar el botón; también se aplicarán automáticamente las mejoras que vayas realizando sobre el botón
      * Mapea las llamadas de TriggerEvent a las funciones de BotonAlSoltar
        + **PointerEnter** 🡪 EntraPuntero()
        + **PointerExit** 🡪 SalePuntero()
* **Comprobación del funcionamiento**
  + Comprueba que el funcionamiento de los cimientos es el siguiente:
    - Al pulsar sobre ellos (sin soltar) se abre el menú correspondiente
    - Podemos mover el cursor (o el dedo) sobre el menú
    - Si soltamos sobre alguno de los icónos, se llama a la “accion”
      * Llama a las funciones ConstruyeXXXX() correspondientes
    - Si soltamos sobre cualquier otro sitio, se cierra el menú
    - El menú no se cierra mientras se mantiene pulsado el botón del ratón (o se mantiene pulsado el dedo contra la pantalla)
  + Aplica los cambios realizados al prefabricado de los cimientos, y asegúrate de que todos los cimientos del mapa funcionan bien

**Ejercicio 7B: Preparación de la Gestión de la TorreBase**

* **Preparación del Menú de la TorreBase**
  + Prepara el menú desplegable de la TorreBase; este menú es común para todos los tipos de torre
    - Instancia una TorreBase
    - Crea un nuevo Canvas de Mundo
    - Colócalo como hijo de los TorreBase
    - Añade un panel semitransparente que cubra la torre, y que sobresalga por su derecha, como en esta imagen  
        
      Incluir Imagen
    - Añade tres imágenes cuadradas, a modo de iconos, para seleccionar la acción a realizar
    - Añade el componente TriggerEvent a las tres imágenes para poder responder a las notificaciones de interacción
    - Añade el componente MenuDesplegable al Canvas
      * Mapea las llamadas de TriggerEvent en el Cimiento a las funciones de MenuDesplegable
        + XXXXX 🡪 Mostrar()
        + XXXXX 🡪 Ocultar()
    - Añade el componente BotonAlSoltar a las imágenes/iconos
      * **Consejo:** reutiliza el prefabricado de BotonAlSoltar que preparaste para el cimiento
      * Mapea las llamadas de TriggerEvent a las funciones de BotonAlSoltar
        + XXXXX 🡪 EntraPuntero()
        + XXXXX 🡪 SalePuntero()
        + XXXXX 🡪 OnReleased()

**Ejercicio 7C: Mejora, Reparación y Venta de las Torres**

* **Implementa la Mejora de la Torre**
  + En el MenuDesplegable de la Torre, responde al botón de Mejora con los siguientes pasos:
    - Destruye la torre actual
    - Instancia en su lugar el prefabricado de la torreMejorada
    - **¡Ojo! Tendrás que comprobar si hay torreMejorada; si no la hay, no hagas nada**
* **Implementa la Reparación de la Torre**
  + En el MenuDesplegable de la Torre, responde al botón de Mejora con los siguientes pasos:
    - Recupera una cantidad de puntos de vida
* **Implementa la Venta de la Torre**
  + En el MenuDesplegable de la Torre, responde al botón de Venta con los siguientes pasos:
    - Destruye la torre actual

**Ejercicio 7:** Realiza un video mostrando los progresos hasta este punto.

|  |
| --- |
|  |

**Ejercicio 8: Condiciones de Final de Partida**

Implementa las condiciones de Final de Partida. Para ello tendrás que implementar la resistencia del mapa (la “vida” que le queda).

**Prepara la clase GameLogic**

* Crea un componente GameLogic
  + Conviertelo un “singleton”
  + Añade las siguientes variables públicas:
    - **Resistencia** – es la resistencia total, que irá disminuyendo por cada enemigo que alcance el final de su recorrido
  + Añade las siguientes variables privadas:
    - **cantidadDeEnemigos –** es una cuenta de la cantidad de enemigos que se han ido registrando
    - **ultimoEnemigoCreado –** es un booleano que guarda si ya se ha creado el último enemigo; si el último enemigo ya fue creado, la cantidad de enemigos llega a cero, y la resistencia es aún positiva, el resultado es de victoria
  + Prepara las siguientes funciones:
    - **NotificaEnemigoCreado() –** sirve para que los Enemigos notifiquen que han sido creados
      * Incrementa en 1 cantidadDeEnemigos
    - **NotificaEnemigoDestruido() –** sirve para que los Enemigos notifiquen que han sido destruidos
      * Decremente en 1 cantidadDeEnemigos
    - **NotificaEnemigoLlegaAlFinal() –** sirve para que los Enemigos notifiquen si han
      * Pon ultimoEnemigoCreado a true
    - **NotificaUltimoEnemigoCreado() –** sirve para que el GestorOleadas notifique cuando se ha instanciado el último enemigo de la última oleada
      * Pon ultimoEnemigoCreado a true
  + En el Update(), si aún no se ha cumplido ninguna condición de final, comprueba:
    - Si la resistencia llega a 0, entonces se cumple la condición de derrota
      * Prepara una cutscene con una timeline para que muestre un canvas con un texto de Derrota
    - Si la cantidad de enemigos llega a cero y el último enemigo ya fue creado
      * Prepara una cutscene con una timeline para que muestre un canvas con un texto de Victoria
    - Si se ha cumplido cualquiera de las dos condiciones
      * A los cinco segundos, recarga la escena
        + Puedes hacerlo con

DOVirtual.DelayedCall()

Invoke()

**Crea el objeto GameLogic**

* Crea un GameObject GameLogic
* Añade el Componente GameLogic
  + Establece Resistencia a 100

**Modifica los Enemigos**

* Haz que la Clase Enemigo
  + Llame a NotificaEnemigoCreado() en el Start()
  + Llame a NotificaEnemigoDestruido() en OnDestroy()
  + Llame a NotificaEnemigoLlegaAlFinal() en el OnComplete() del DOPath()
    - Esto puede ocurrir o no, depende de si el enemigo muere por el camino

**Modifica el GestorOleadas**

* Haz que la clase GestorOleadas llame a NotificaUltimoEnemigoCreado() **cinco segundos después de que** haya terminado de instanciar el último Enemigo en la ultima DefinicionOleada del Guion
  + Estos cinco segundos son importantes para que no haya un lapso en el que el último enemigo haya llamado a NotificaEnemigoCreado()

**Comprueba que todo funcione**

A estas alturas, el juego debería estar funcional, y poder ser jugado una y otra vez.

* Pruebas extremas
  + Prepara un guión con una sola oleada de un solo enemigo
  + Pon a la GameLogic un valor de resistencia de 0,00001
  + Prueba la derrota
    - Comprueba que, al no haber torre, el enemigo llega rápidamente al final y ocurre la derrota
  + Prueba la victoria
    - Ahora, añade una o varias torres para asegurar que se mata al enemigo
    - Comprueba que, en cuanto muere el enemigo, ocurre la victoria
* Prueba normal
  + En el escenario que hemos trabajado desde el principio, simplemente juega y comprueba que no ocurren victorias ni derrotas antes de tiempo

**Ejercicio 8:** Realiza un video mostrando los progresos hasta este punto.

|  |
| --- |
|  |

**Extra (1/3 cuenta como actitud): Características Opcionales**

Prepara por lo menos dos de las siguientes características opcionales:

* Implementa la Economía
* Añade HUD
* Añade un sistema de navegación que permita acceder a la partida desde un menú principal, pausar durante la partida, modificar opciones, etc.
* Añade música, sonidos y fanfarrias
  + Mínimo una pieza musical de batalla
  + Mínimo dos fanfarrias (victoria y derrota)
  + Mínimo sonidos de disparo de torre, disparo enemigo, muerte de enemigo, y destrucción de torreta
* Añade una magia con tiempo de recarga alto que permita lanzar un obús sobre una zona de la pantalla, realizando gran cantidad de daño de área a los enemigos
  + Intercambiable por una tormenta que vaya haciendo daño por segundo
  + O por cualquier otra fantasía que te parezca interesante
* Mejora Gráfica 1: pon material al terreno – puedes utilizar el Terrain Sample Asset Pack, está gratuito en la AssetStore
* Mejora Gráfica 2: pon gráficos y animaciones a los enemigos – puedes descargar modelos de PolyPizza y animarlos en Mixamo
  + ¡Vigila con el poligonaje! Ten en cuenta que se van a juntar gran cantidad de personajes en pantalla, y
* Mejora Gráfica 3: pon gráficos a las torres y a sus mejoras
  + Seguro que encuentras algo en PolyPizza
* Mejora Gráfica 4: pon efectos a los ataques y al feedback visual
  + Cuando mueren los enemigos
  + Cuando se construyen las torres
  + Cuando se reparan las torres
  + Cuando se mejoran las torres
  + Cuando se destruyen las torres
* Mejora Gráfica 5: pon un gráfico interesante a los caminos utilizando el SplinePathGenerator
  + Si tienes terminado este ejercicio, no te costará utilizarlo para sacar unos visuales más atractivos

**Extras:** Realiza un video mostrando los progresos hasta este punto y explica brevemente de qué manera has realizado las tareas.

|  |
| --- |
|  |