Plan de Desarrollo - Tower Defense (5 Días)

Día 1: Configuración del Proyecto y Mapa (Tilemap)

- **Objetivo:** Tener el mapa base cargado con tilesets, la cámara configurada y un sistema básico de interacción con el ratón para seleccionar casillas.
- **Tareas:**
    1. **Configuración del Proyecto Unity:**
        - Crear un nuevo proyecto 2D en Unity.
        - Importar todos los assets (sprites de tilesets, torres, enemigos, UI, etc.) y organizarlos en carpetas lógicas (`Sprites/Tiles`, `Sprites/Towers`, `Prefabs/Enemies`, `Scripts`, etc.).
    2. **Creación del Mapa con Tilemap:**
        - Configurar un Tilemap. Es la forma más eficiente de gestionar mapas basados en tiles en Unity.
        - Crear un Tile Palette con tus tilesets para el camino, hierba, rocas, etc.
        - Diseñar el mapa base usando el Tilemap. Asegúrate de tener claro el camino por donde irán los enemigos.
    3. **Configuración de la Cámara:**
        - Ajustar la cámara principal para que muestre correctamente el mapa. Si es un juego top-down 2D, una cámara ortográfica es lo ideal.
    4. **Interacción con el Ratón (Highlight de Casillas):**
        - Crear un script para detectar la posición del ratón sobre el Tilemap.
        - Implementar un "highlight" visual (un sprite que sigue al ratón o un cambio de color) sobre la casilla actual del ratón. Esto te permitirá ver qué casilla vas a seleccionar.

Día 2: Sistema de Spawneo de Enemigos y Movimiento Básico

- **Objetivo:** Que los enemigos puedan aparecer y seguir un camino predefinido.
- **Tareas:**
    1. **Definición de Waypoints:**
        - Crear una serie de objetos vacíos (GameObjects) en tu escena que representen los puntos clave del camino que seguirán los enemigos (waypoints).
        - Crear un script `Path` o `WaypointManager` que almacene estos waypoints en una lista o array.
    2. **Clase Base de Enemigos:**
        - Crear una clase `Enemy` que contenga propiedades como `health`, `speed`, `goldValue` (oro que da al morir).
        - Crear un prefab para cada tipo de enemigo (si hay varios) basado en esta clase.
    3. **Movimiento de Enemigos:**
        - En el script `Enemy`, implementar la lógica para que el enemigo se mueva de un waypoint al siguiente. Puedes usar

`Vector3.MoveTowards` o `Transform.Translate` junto con `Time.deltaTime`.

4. **Spawneo de Enemigos:**
    - Crear un `EnemySpawner` que instancie prefabs de enemigos en el punto de inicio del camino.
    - Implementar una lógica de oleadas (waves) básica: spawneo de un número determinado de enemigos cada cierto tiempo.

Día 3: Sistema de Colocación de Torres y Disparo Básico

- **Objetivo:** Poder colocar torres en el mapa y que estas detecten y disparen a los enemigos.
- **Tareas:**
    1. **Clase Base de Torres (`Tower`):**
        - Crear una clase `Tower` con propiedades como `damage`, `fireRate`, `range`.
        - Crear prefabs para cada tipo de torre.
    2. **Sistema de Colocación de Torres:**
        - Integrar la interacción del ratón del Día 1. Cuando el jugador haga clic en una casilla válida (que no sea camino), debería poder instanciar una torre.
        - Necesitarás lógica para verificar si la casilla es válida para construir. Puedes usar tags en los tiles del Tilemap o una matriz lógica en un script `MapManager`.
        - Implementar la funcionalidad de seleccionar una torre del menú superior y luego colocarla.
    3. **Detección de Enemigos (`Targeting`):**
        - En el script `Tower`, implementar un sistema para detectar enemigos dentro de su `range`. Puedes usar `Physics2D.OverlapCircleAll` o `Trigger Colliders` en las torres.
        - Seleccionar un objetivo (por ejemplo, el enemigo más cercano o el que esté más avanzado en el camino).
    4. **Disparo Básico:**
        - En el script `Tower`, implementar la lógica para que la torre dispare a su objetivo a una `fireRate` determinada. Por ahora, un simple "rayo" o una reducción de vida instantánea puede ser suficiente, sin proyectiles visibles si el tiempo es muy ajustado.

Día 4: Interfaz de Usuario (UI), Sistema de Oro y Vida, Venta/Mejora de Torres

- **Objetivo:** Implementar los elementos básicos de la interfaz de usuario, la economía del juego y las funcionalidades de venta/mejora de torres.
- **Tareas:**
    1. **Interfaz de Usuario (Canvas):**
        - Crear un Canvas en el modo Screen Space - Camera.
        - Diseñar la UI para mostrar la salud (`Health`), el oro (`Gold`) y los botones del menú superior (colocar torre, vender, mejorar).

        - Conectar estos elementos UI a un script `UIManager` para actualizar los valores y manejar los eventos de los botones.
    2. **Sistema de Oro y Vida:**
        - Crear un script `GameManager` que gestione el `gold` actual del jugador y la `health` del jugador.
        - Cuando un enemigo es destruido, el `GameManager` debe añadir `gold`.
        - Cuando un enemigo llega al final del camino, el `GameManager` debe reducir la `health` del jugador.
        - Lógica de "Game Over" cuando la salud llega a 0.
    3. **Venta de Torres:**
        - Cuando el jugador selecciona una torre colocada, debe aparecer la opción de venderla.
        - Al vender, se debe destruir la torre y el `GameManager` debe devolver una parte de su costo en `gold`.
    4. **Mejora de Torres:**
        - Al seleccionar una torre, debe aparecer la opción de mejorarla.
        - Implementar un sistema de mejora básico: aumenta el `damage`, `fireRate` o `range` de la torre a cambio de `gold`.

Día 5: Pulido, Efectos Visuales y Sonoros Básicos, Equilibrio

- **Objetivo:** Darle un toque final al juego, añadir efectos y realizar un equilibrio inicial para que sea jugable.
- **Tareas:**
    1. **Efectos Visuales (VFX):**
        - Añadir partículas simples o animaciones para el disparo de las torres, la explosión de los enemigos al morir, o el "highlight" de la casilla del ratón. Unity tiene un sistema de partículas muy potente.
    2. **Efectos Sonoros (SFX):**
        - Importar sonidos básicos para el disparo de las torres, la muerte de los enemigos, la colocación de torres, etc.
        - Reproducir estos sonidos a través de Audio Sources en los GameObjects correspondientes.
    3. **Equilibrio Inicial:**
        - Ajustar los valores de `health`, `speed` de los enemigos.
        - Ajustar el `damage`, `fireRate`, `range` y los costos de las torres.
        - Ajustar las recompensas de `gold` por enemigo.
        - Ajustar la velocidad de spawneo de las oleadas. El objetivo es que sea desafiante pero no imposible.
    4. **Menú Principal / Pantalla de Fin de Juego (Opcional si hay tiempo):**
        - Un menú muy simple con un botón "Start".
        - Una pantalla de "Game Over" que muestre un mensaje y un botón para reiniciar.
    5. **Refinamiento General:**

        - Testear el juego exhaustivamente para encontrar bugs y asegurar que todo funciona como se espera.
        - Optimizar el rendimiento si hay algún cuello de botella (aunque para un juego simple 2D no debería haber muchos).

Consejos Adicionales para Ángel:

- **Itera Rápido:** No intentes hacer todo perfecto al principio. Implementa la funcionalidad básica y luego refínala.
- **Divide y Vencerás:** Cada tarea es un pequeño objetivo. Concédete pequeños éxitos.
- **Control de Versiones:** Si puedes, usa Git (o simplemente haz copias de seguridad de tu proyecto al final de cada día).
- **Unity Docs y Tutorials:** Si te atascas, la documentación de Unity y los tutoriales online son tus mejores amigos.
- **Prioriza:** Si ves que una tarea te lleva demasiado tiempo, haz la versión más sencilla posible o déjala para más adelante si no es crítica para el *core* del juego.
- **Reutilización:** Dado que eres estudiante, ya sabes la importancia de reutilizar código. Por ejemplo, la lógica de `Enemy` puede ser una clase base para diferentes tipos de enemigos con herencia.