Plants VS Zombies
Proyecto Unity

ÍNDICE

Introducción

Plants vs. Zombies es un icónico videojuego de estrategia en tiempo real desarrollado por PopCap Games, que mezcla humor y acción en un enfrentamiento entre plantas y zombis. Los jugadores deben defender su hogar del ataque de oleadas de zombis utilizando plantas con habilidades específicas. El desafío crece a medida que avanzan las oleadas, exigiendo una planificación estratégica para gestionar los recursos y posicionar las plantas adecuadamente.

En este proyecto, se ha desarrollado una versión simplificada del juego en Unity como parte de una colaboración entre los alumnos Jesús Moruno Muñozo y Adrián Ramos Caballero. La adaptación recrea las mecánicas principales del juego original, centrándose en ofrecer una experiencia básica pero funcional.

Esta versión incluye:

- **Plantas disponibles:**
  - **Girasol**: Genera recursos (sol) para colocar otras plantas.
  - **Lanzaguisantes**: Ataca a los zombies disparando guisantes que disminuyen su vida.

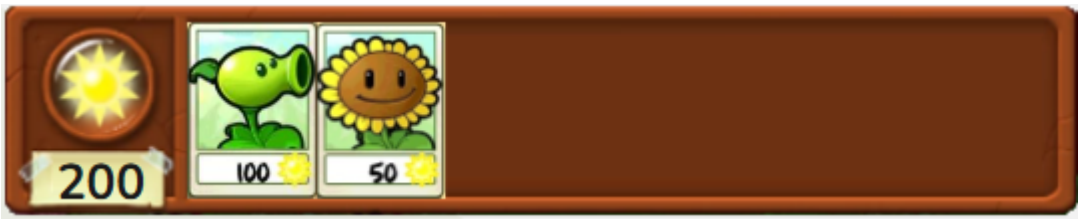
- **Zombies disponibles:**

- **Zombie Normal**: Avanza en línea recta y es fácil de derrotar.
- **Zombie Cono**: Más resistente gracias a su cono protector.

El proyecto se ha desarrollado como ejercicio práctico para explorar el diseño y desarrollo de videojuegos en Unity. Esta documentación se enfocará exclusivamente en las partes desarrolladas por Adrián Ramos Caballero, destacando los aspectos técnicos y creativos involucrados en su contribución.

A continuación, se detallan las partes del proyecto desarrolladas por el alumno Adrián Ramos Caballero. Estas comprenden la lógica, el funcionamiento y la creación de los zombis, abarcando la implementación de su comportamiento, características y mecánicas, así como su interacción con el entorno y las plantas. Además, incluyen la lógica, el funcionamiento y el diseño de la interfaz gráfica (GUI), que permite al jugador gestionar y controlar las acciones disponibles durante el juego, como la colocación de plantas y la recogida de soles.

Explicación de la GUI Principal



En Plants vs. Zombies, la GUI principal es la interfaz gráfica del usuario que sirve como punto de interacción entre el jugador y el juego. Esta interfaz permite gestionar las acciones necesarias para desarrollar la estrategia de defensa contra los zombis. En la versión simplificada del proyecto, la GUI principal ofrece las siguientes funcionalidades:

- **Seleccionar la planta a usar:** Permite al jugador elegir entre las plantas disponibles, como el Girasol o el Lanzaguisantes, según la estrategia deseada.

- **Aumentar o disminuir el número de soles:** Facilita la gestión de los recursos necesarios para colocar plantas en el campo de juego.

- **Instanciar plantas:** Permite colocar las plantas seleccionadas en las posiciones estratégicas del jardín para enfrentar a los zombis.

- **Recoger soles:** Permite recolectar los soles generados por los Girasoles o los que aparecen en el campo de juego, aumentando los recursos disponibles para el jugador.

Esta GUI ha sido diseñada para ser intuitiva y eficiente, asegurando que el jugador pueda realizar todas las acciones de manera fluida durante el desarrollo del juego.

Para implementar las cuatro acciones mencionadas anteriormente, la GUI principal está vinculada al script **GameManager.cs**, el cual gestiona la lógica central del juego. Este script contiene la siguiente estructura.

Atributos de la clase

- **List<Plantas> plantasAUsar:** Este atributo es una lista que almacena objetos de tipo Plantas. Se utiliza para definir las plantas que se pueden utilizar en el juego. Cada planta tiene propiedades como su costo y la representación gráfica (sprite) que se mostrará en la interfaz de usuario.

- **GameObject Deck:** Este atributo es un objeto de tipo GameObject que representa el contenedor donde se mostrarán las cartas de las plantas disponibles para el jugador. Generalmente, este objeto se utiliza para organizar y agrupar las cartas en la interfaz de usuario.

- **GameObject PrefabCarta:** Este atributo es un objeto de tipo GameObject que representa el prefab (plantilla) de la carta de planta. Este prefab se instanciará para crear las cartas que se mostrarán en el Deck. Contiene los componentes necesarios, como la imagen de la planta y el botón para seleccionarla.

- **Text TxtSoles:**Este atributo es un componente de tipo Text que se utiliza para mostrar la cantidad actual de Soles que tiene el jugador en la interfaz de usuario. Se actualiza cada vez que el jugador gana o gasta Soles.

- **int Soles:** Este atributo es una variable entera que almacena la cantidad actual de Soles que tiene el jugador. Inicialmente se establece en 999.

- **int PlantaAusar:** Este atributo es una variable entera que indica el índice de la planta que el jugador ha seleccionado para usar. Se actualiza cada vez que el jugador hace clic en una carta de planta en el Deck. Este índice se utiliza posteriormente para instanciar la planta correspondiente en el juego.

Método Start

```
void Start()
    {
        ActualizarSoles(0);
```

```
        for (int i = 0; i < plantasAUsar.Count; i++)
        {
            GameObject go = Instantiate(PrefabCarta) as
GameObject;
            go.transform.SetParent(Deck.transform);
            go.transform.position = Vector3.zero;
            go.transform.localScale = Vector3.one;

            Image img = go.GetComponent<Image>();
            img.sprite = plantasAUsar[i].cartaAsignada;

            Button bot = go.GetComponent<Button>();
            bot.onClick.RemoveAllListeners();
            int u = i;
            bot.onClick.AddListener(() => { PlantaAusar = u; });
        }
    }
```

El método Start en la clase GameManager se ejecuta automáticamente al iniciar el juego o al activar el objeto que contiene este script. Su propósito es inicializar ciertos aspectos del juego. Primero, llama al método ActualizarSoles(0), que establece el texto que muestra la cantidad de Soles, asegurando que se muestre el valor inicial (999) sin modificarlo. Luego, se utiliza un bucle for para iterar sobre cada planta en la lista plantasAUsar. En cada iteración, se instancia un nuevo objeto de carta a partir del prefab PrefabCarta, que representa visualmente la planta.

Este objeto se configura para ser hijo del contenedor Deck, lo que permite que todas las cartas se agrupen adecuadamente en la interfaz de usuario. Se establece su posición en el origen del espacio del mundo y se asegura que su escala sea la original, evitando deformaciones. A continuación, se obtiene el componente Image del objeto de carta y se le asigna el sprite correspondiente a la planta actual, lo que permite que la carta muestre la representación gráfica adecuada.

Además, se obtiene el componente Button del objeto de carta y se eliminan todos los listeners previos para evitar acciones duplicadas. Se utiliza una variable temporal para capturar el índice actual del bucle, lo que es necesario para que el listener sepa qué planta se ha seleccionado. Finalmente, se añade un listener al botón que, al hacer clic, asigna el índice de la planta seleccionada a la variable PlantaAusar, permitiendo al jugador elegir qué planta quiere usar. En resumen, el método Start prepara la interfaz del juego creando y configurando las cartas de las plantas y estableciendo la funcionalidad para la selección de plantas por parte del jugador

Método Update

```
void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            Ray r =
Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit2D hit =
Physics2D.Raycast(r.origin,r.direction);
            if (hit.collider != null)
            {
                if (hit.collider.CompareTag("Cuadricula"))
                {
                    Transform t = hit.collider.transform;
                    CrearPlanta(PlantaAusar, t);
                }
                else if (hit.collider.CompareTag("Sol"))
                {
                    ActualizarSoles(50);
                    Destroy(hit.collider.gameObject);
                }
            }
        }

    }
```

El método Update en la clase GameManager se ejecuta una vez por cada frame del juego y se utiliza para manejar la lógica que debe ser verificada o actualizada continuamente durante la ejecución del juego. En este contexto, el método Update se encarga de gestionar la entrada del jugador y de actualizar la interfaz de usuario en función de las acciones realizadas.

Dentro de este método, se comprueba si el jugador ha presionado la tecla de espacio. Si es así, se verifica si hay suficientes Soles disponibles para realizar una acción, como instanciar una planta en el juego. Si el jugador tiene suficiente cantidad de Soles, se procede a instanciar la planta correspondiente en la posición deseada dentro del juego, lo que puede implicar la creación de un nuevo objeto de planta en la escena.

Además, después de instanciar la planta, se deducen los Soles correspondientes del total del jugador, y se actualiza el texto que muestra la cantidad de Soles en la interfaz de usuario para reflejar el nuevo total. Esto asegura que el jugador esté siempre informado sobre sus

recursos y que la interacción con el juego sea fluida y dinámica. En resumen, el método Update es fundamental para gestionar la lógica del juego en tiempo real, permitiendo al jugador interactuar con el entorno y asegurando que la interfaz se mantenga actualizada con la información relevante.

Método CrearPlanta

```
void CrearPlanta(int numero, Transform t)
    {
        if (plantasAUsar[numero].precioSoles > Soles)
            return;
        if (t.childCount != 0)
            return;

        GameObject g =
Instantiate(plantasAUsar[PlantaAusar].gameObject, t.position,
gameObject.transform.rotation) as GameObject;
        g.transform.SetParent(t);

        ActualizarSoles(-plantasAUsar[numero].precioSoles);
    }
```

El método CrearPlanta tiene como objetivo instanciar una nueva planta en el juego, siempre que se cumplan ciertas condiciones. Primero, verifica si el costo de la planta que se desea crear es mayor que la cantidad de Soles que el jugador tiene. Si el costo excede los Soles disponibles, el método se detiene y no se crea la planta, asegurando que el jugador no realice acciones que no puede permitirse. A continuación, se verifica si el transform donde se desea colocar la planta ya tiene un hijo; si es así, el método también se detiene, evitando que se coloquen múltiples plantas en el mismo espacio, lo cual podría romper la lógica del juego.

Si ambas condiciones se cumplen, se procede a instanciar la planta utilizando Instantiate, creando un nuevo objeto de planta basado en el prefab correspondiente. Este nuevo objeto se posiciona en la ubicación especificada y se le asigna la rotación del objeto que contiene el script. Una vez que la planta ha sido instanciada, se establece su padre como el transform correspondiente, organizando así la jerarquía de los objetos en la escena.

Finalmente, se actualiza la cantidad de Soles del jugador restando el costo de la planta, lo que refleja que se ha realizado una compra. En resumen, el método CrearPlanta es crucial en la mecánica de colocación de plantas dentro del juego, gestionando la creación de nuevas plantas y asegurando que se cumplan las condiciones necesarias, como la disponibilidad de recursos y el espacio para colocar la planta.

Método ActualizarSoles

```
public void ActualizarSoles(int Add)
    {
        Soles += Add;
        TxtSoles.text = Soles.ToString();
    }
```

El método ActualizarSoles es una función que se encarga de modificar la cantidad de Soles que el jugador tiene en el juego. Cuando se llama a este método, se pasa un valor numérico como parámetro, que puede ser positivo o negativo. Si el valor es positivo, se suma a la cantidad actual de Soles del jugador, lo que significa que el jugador ha ganado Soles. Por otro lado, si el valor es negativo, se resta de la cantidad actual de Soles del jugador, lo que significa que el jugador ha gastado Soles, por ejemplo, al comprar una planta.

Como ejemplo de funcionamiento, en el contexto del método CrearPlanta, se llama a ActualizarSoles con un valor negativo que coincide con el costo de la planta que se acaba de crear. Esto reduce la cantidad de Soles del jugador en la cantidad correspondiente al costo de la planta, reflejando que se ha realizado una compra. De esta manera, el método ActualizarSoles ayuda a mantener una cuenta precisa de los Soles del jugador, permitiendo que el juego pueda realizar cálculos y decisiones basadas en la cantidad de recursos disponibles.

Funcionamiento de los zombies

La implementación de los zombis en esta versión simplificada de Plants vs. Zombies se centra en reproducir su comportamiento básico, permitiendo su interacción con las plantas y el entorno. Los zombis están diseñados como enemigos controlados por el juego, con movimientos y acciones predeterminadas que contribuyen a la experiencia estratégica del jugador.

Cada zombi cuenta con atributos y métodos que definen su funcionamiento. Estos incluyen su capacidad para avanzar en línea recta, interactuar con plantas (como comerlas si están en su camino) y recibir daño al ser golpeados por proyectiles, como los guisantes.

El código de los zombis se estructura de manera modular, utilizando scripts para encapsular las características individuales de cada tipo de zombi. En este proyecto, se implementan dos tipos:

- **Zombi Normal:** Representa el nivel básico, con atributos estándar de velocidad y salud.

- **Zombi Cono:** Similar al zombi normal, pero con una mayor resistencia gracias a su casco de cono.

Esta implementación busca ofrecer un comportamiento realista y funcional, asegurando que los zombis presenten un desafío equilibrado dentro del juego.

Atributos de la clase

- **public int vida:** Este atributo representa la cantidad de vida que tiene el zombie. En este caso, comienza con un valor de 4. Cuando el zombie recibe daño, este valor se reduce. Si la vida llega a 0 o menos, el zombie es destruido.

- **public float velocidad:** Este atributo determina la velocidad a la que el zombie se mueve. El valor se puede establecer en el editor de Unity o en otro lugar en el código, y afecta cómo de rápido el zombie se desplaza hacia la izquierda en la escena.

- **public LayerMask layerPlanta:** Este atributo es un LayerMask, que se utiliza para especificar qué capas de colisiones se deben considerar al realizar raycasts. En este caso, se utiliza para detectar si hay alguna planta (o un objeto que representa una planta) en la dirección hacia la que el zombie está mirando (izquierda).

- **public float cadencia:** Este atributo indica la cadencia o el tiempo que debe transcurrir entre los ataques del zombie. En este caso, se establece en 1 segundo. Esto significa que el zombie puede atacar a la planta una vez cada segundo si está lo suficientemente cerca.

- **float cadAux:** Este es un atributo auxiliar que se utiliza para llevar un registro del tiempo que ha pasado desde el último ataque del zombie. Se incrementa en cada actualización (Update) y se reinicia a 0 cuando se realiza un ataque o cuando no hay una planta frente al zombie.

Estos atributos trabajan juntos para definir cómo se comporta el zombie en el juego, incluyendo su movimiento, su capacidad para atacar y su resistencia a los daños.

Método Update

```
void Update()
    {
        Debug.DrawRay(transform.position, Vector3.left* .5f);
        RaycastHit2D hit = Physics2D.Raycast(transform.position,
Vector3.left, .5f, layerPlanta);
```

```
        if (hit.collider != null)
        {
            cadAux += Time.deltaTime;
            if (cadAux >= cadencia)
            {
                cadAux = 0;
                hit.collider.SendMessage("Morder");
            }
        }
        else
        {
            cadAux = 0;
            transform.position -= Vector3.right * velocidad *
Time.deltaTime;
        }
    }
```

El método Update es llamado una vez por cada frame del juego y tiene como propósito controlar el comportamiento del zombie, específicamente su movimiento y su capacidad para atacar a las plantas. En primer lugar, dibuja un rayo en la vista de escena de Unity para propósitos de depuración, extendiéndose 0.5 unidades hacia la izquierda desde la posición actual del zombie, lo que ayuda a visualizar la dirección en la que se está realizando la detección de colisiones. A continuación, se realiza un raycast hacia la izquierda desde la posición del zombie, con una longitud de 0.5 unidades, verificando si hay un collider en la capa especificada por layerPlanta. Si se detecta un collider, se almacena la información en hit.

Si el raycast detecta un collider (una planta), se incrementa el temporizador cadAux con el tiempo transcurrido desde el último frame. Luego, se comprueba si ha pasado el tiempo suficiente para realizar un ataque, es decir, si cadAux es mayor o igual a cadencia. Si es así, se reinicia el temporizador a 0 y se envía un mensaje al collider detectado para que ejecute su método Morder, lo que representa un ataque del zombie.

Por otro lado, si no se detecta ninguna planta, se reinicia el temporizador cadAux a 0 y el zombie se mueve hacia la izquierda en función de su velocidad, restando de su posición actual un valor que es el producto de velocidad y Time.deltaTime, asegurando así que el movimiento sea suave y esté basado en el tiempo. En resumen, el método Update gestiona tanto el movimiento del zombie como su capacidad para atacar a las plantas, utilizando raycasting para detectar colisiones y un temporizador para controlar la cadencia de los ataques, lo que permite que el zombie se comporte de manera realista y dinámica en el entorno del juego.

Método OnTriggerEnter2D

```
void OnTriggerEnter2D(Collider2D col)
    {
        if (col.CompareTag("Guisante"))
        {
            vida--;
            Destroy(col.gameObject);

            if (vida <= 0)
                Destroy(gameObject);

        }
        if (col.CompareTag("FailState"))
        {
            Destroy(gameObject);
            print("Has perdido");
        }

    }
```

El método OnTriggerEnter2D es un evento que se activa cuando un collider marcado como "trigger" entra en contacto con el collider del objeto que contiene este script. Su propósito es gestionar las interacciones del zombie con otros objetos en el juego, específicamente aquellos que tienen etiquetas específicas. En este método, se verifica si el objeto que ha entrado en contacto tiene la etiqueta "Guisante". Si es así, se decrementa la vida del zombie en una unidad y se destruye el objeto "Guisante" que ha colisionado. A continuación, se comprueba si la vida del zombie ha llegado a 0 o menos; de ser así, el zombie se destruye, eliminando así su presencia en el juego.

Además, el método también verifica si el objeto que ha colisionado tiene la etiqueta "FailState". Si se detecta un objeto con esta etiqueta, el zombie se destruye inmediatamente y se imprime un mensaje en la consola que indica que el jugador ha perdido. Este método es esencial para el manejo de colisiones en el juego, ya que permite al zombie interactuar con otros elementos, como los guisantes que representan un ataque o un daño, y responder adecuadamente a situaciones que resultan en un estado de fallo. En resumen, OnTriggerEnter2D es fundamental para definir la lógica de colisiones y las consecuencias de dichas interacciones en el comportamiento del zombie dentro del entorno del juego.

Instanciador de zombies

El script InstanciadorDeZombies.cs es una herramienta diseñada para instanciar objetos de tipo "Zombie" en Unity en momentos específicos, utilizando un arreglo de tiempos definido por el usuario. Este script contiene dos variables públicas: un arreglo de enteros llamado tiempos, que permite al usuario establecer diferentes intervalos de tiempo en segundos para la generación de los zombies, y una variable de tipo GameObject llamada Zombie, que permite asignar el prefab del zombie que se desea instanciar desde el editor de Unity.

Al iniciar el juego, el método Start() se ejecuta, donde se recorre cada elemento del arreglo tiempos mediante un bucle. Para cada tiempo especificado, se utiliza el método Invoke para llamar a la función InstanciarZombie después del retraso correspondiente. Esta función es responsable de crear una instancia del objeto Zombie en la posición del objeto que contiene este script y con la rotación del prefab de zombie.

En resumen, el script permite generar zombies de manera controlada en intervalos regulares, lo que es especialmente útil en juegos donde los jugadores deben enfrentar oleadas de enemigos, como en juegos de supervivencia o de acción.

```
public class InstanciadorDeZombies : MonoBehaviour {


    public int[] tiempos;

    public GameObject Zombie;

    void Start()
    {
        for (int i = 0; i < tiempos.Length; i++)
        {
            Invoke("InstanciarZombie", tiempos[i]);
        }
    }
    void InstanciarZombie()
    {
        Instantiate(Zombie, transform.position,
Zombie.transform.rotation);
    }
}
```

Apariencia en Unity

En Unity, para simular que los zombis llegan caminando al jardín sin aparecer de manera abrupta, se utilizan diez objetos invisibles colocados fuera de la vista de la cámara. Estos objetos actúan como instanciadores y contienen el script encargado de generar los zombis. Al hacerlo, los zombis parecen emerger de un punto distante, lo que mejora la inmersión del jugador al no revelar el momento exacto de su aparición.