**TRABAJO**

**UNITY**

**Nombre: Proyecto Granja**

**Integrantes:**

**Juarez Maria Florencia**

**Matías González Sandoval**

**Santiago Silva**

**Parte de Santiago**

Hizo los scripts de la cámara, el personaje que controla el jugador y configuró el ciclo día y noche:

Script: Movimiento3.cs

using UnityEngine;

public class Movimiento3 : MonoBehaviour

{

public float walkSpeed = 5f;

public float runSpeed = 10f;

public Animator animator;

public float jumpHeight = 3f;

public Transform groundCheck;

public float groundDistance = 0.1f;

public LayerMask groundMask;

bool isGrounded;

private CharacterController controller;

private Vector3 velocity;

public float gravity = -9.81f;

public Transform camara; // Asigna aquí tu cámara en el inspector

void Start()

{

controller = GetComponent<CharacterController>();

if (controller == null)

{

Debug.LogError("No se encontró CharacterController en el GameObject.");

}

}

void Update()

{

float verticalInput = Input.GetAxis("Vertical");

float horizontalInput = Input.GetAxis("Horizontal");

// Movimiento relativo a la cámara

Vector3 forward = camara.forward;

Vector3 right = camara.right;

forward.y = 0;

right.y = 0;

forward.Normalize();

right.Normalize();

Vector3 moveDirection = (forward \* verticalInput + right \* horizontalInput).normalized;

float speed = Input.GetKey(KeyCode.LeftShift) && verticalInput > 0 ? runSpeed : walkSpeed;

controller.Move(moveDirection \* speed \* Time.deltaTime);

// Rotar el personaje hacia la dirección de movimiento si se está moviendo

if (moveDirection.magnitude > 0.1f && verticalInput > 0)

{

transform.rotation = Quaternion.Slerp(transform.rotation, Quaternion.LookRotation(moveDirection), 0.15f);

}

// Animaciones

if (animator != null)

{

animator.SetFloat("VelX", horizontalInput);

animator.SetFloat("VelY", verticalInput);

animator.SetBool("IsSprinting", Input.GetKey(KeyCode.LeftShift) && verticalInput > 0);

}

// Comprobar si está en el suelo

isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance, groundMask);

if (isGrounded && velocity.y < 0)

velocity.y = -2f;

// Saltar

if (Input.GetKeyDown(KeyCode.Space) && isGrounded)

{

if (animator != null) animator.Play("Jump");

velocity.y = Mathf.Sqrt(jumpHeight \* -2f \* gravity);

}

// Aplicar gravedad

velocity.y += gravity \* Time.deltaTime;

controller.Move(velocity \* Time.deltaTime);

}

}

Script: Ciclodianoche.cs

using UnityEngine;

public class CicloDiaNoche : MonoBehaviour

{

[Range(0.0f, 24f)] public float Hora = 12;

public int Dia = 1; // <-- Agregado: contador de días

public Transform Sol;

public float DuracionDelDiaEnMinutos = 1;

private float SolX;

private void Update()

{

Hora += Time.deltaTime \* (24 / (60 \* DuracionDelDiaEnMinutos));

if (Hora >= 24)

{

Hora = 0;

Dia++; // <-- Agregado: suma un día cuando pasa de 24 horas

}

RotacionSol();

}

void RotacionSol()

{

SolX = 15 \* (Hora - 6); // Así a las 12, SolX = 90 (Sol arriba)

Sol.localEulerAngles = new Vector3(SolX, 0, 0);

if (Hora < 6 || Hora > 18)

{

Sol.GetComponent<Light>().intensity = 0;

}

else

{

Sol.GetComponent<Light>().intensity = 1;

}

}

}

Script: CamaraMouse.cs

using UnityEngine;

public class CamaraMouse : MonoBehaviour

{

public Transform objetivo; // El personaje a seguir

public float distancia = 5f; // Distancia detrás del personaje

public float altura = 2f; // Altura sobre el personaje

public float sensibilidadX = 2f;

public float sensibilidadY = 2f;

public float minY = -40f;

public float maxY = 80f;

public float suavizado = 5f; // Velocidad de retorno

private float rotacionY = 0f;

private float rotacionX = 10f;

private float rotacionXInicial = 10f;

void Start()

{

Cursor.lockState = CursorLockMode.Locked;

Vector3 angulos = objetivo.eulerAngles;

rotacionY = angulos.y;

rotacionX = rotacionXInicial;

}

void Update()

{

if (Input.GetMouseButtonDown(1))

Cursor.lockState = CursorLockMode.Locked;

if (Input.GetMouseButtonUp(1))

Cursor.lockState = CursorLockMode.None;

}

void LateUpdate()

{

if (objetivo == null) return;

// Solo rota la cámara si el botón derecho está presionado

if (Input.GetMouseButton(1))

{

rotacionY += Input.GetAxis("Mouse X") \* sensibilidadX;

rotacionX -= Input.GetAxis("Mouse Y") \* sensibilidadY;

rotacionX = Mathf.Clamp(rotacionX, minY, maxY);

}

else

{

// Vuelve suavemente detrás del personaje y a la altura inicial

rotacionY = Mathf.LerpAngle(rotacionY, objetivo.eulerAngles.y, Time.deltaTime \* suavizado);

rotacionX = Mathf.Lerp(rotacionX, rotacionXInicial, Time.deltaTime \* suavizado);

}

Quaternion rotacion = Quaternion.Euler(rotacionX, rotacionY, 0);

Vector3 offset = rotacion \* new Vector3(0, 0, -distancia);

Vector3 posicionDeseada = objetivo.position + Vector3.up \* altura + offset;

transform.position = posicionDeseada;

transform.LookAt(objetivo.position + Vector3.up \* altura);

}

}

**Parte de Florencia**

Carpeta "Prefabs"

¿Qué es un prefab?

Es un objeto que puedes reutilizar muchas veces en la escena.

Guardas un GameObject como prefab para instanciarlo (crear clones) fácilmente por código.

Tip: Crea una carpeta llamada “Prefabs” en la ventana Project, y arrastra allí tus plantas para poder plantarlas luego por código.

2) Script de la planta

Script: PlantMultipleModels.cs

using UnityEngine;

public class PlantMultipleModels : MonoBehaviour

{

public GameObject[] growthModels; // Modelos: semilla, brote, madura

public int daysToGrow = 3; // Cuántos días tarda en crecer

private int currentDay = 0;

private void Start()

{

ActivateModel(0); // Empieza con la semilla

GameTime.Instance.OnNewDay += Grow; // Suscribe al evento de día nuevo

}

void Grow()

{

currentDay++; // Avanza un día

int stage = Mathf.Min(currentDay, growthModels.Length - 1); // Calcula en qué etapa está

ActivateModel(stage); // Muestra la etapa correcta

}

void ActivateModel(int index)

{

for (int i = 0; i < growthModels.Length; i++)

{

growthModels[i].SetActive(i == index); // Activa solo el modelo correcto

}

}

public bool IsFullyGrown() => currentDay >= daysToGrow; // ¿Ya creció?

public void Harvest()

{

if (IsFullyGrown()) Destroy(gameObject); // Si está lista, destruye (se cosechó)

}

private void OnDestroy()

{

GameTime.Instance.OnNewDay -= Grow; // Se desuscribe al destruirse

}

}

¿Por qué se programa aparte?

Cada planta debe manejar sus propios estados de crecimiento.

Este script se asigna al prefab de la planta.

3) Sistema de tiempo

Script: GameTime.cs

using UnityEngine;

using System; // Permite usar eventos y otros elementos como Action

using TMPro;

public class GameTime : MonoBehaviour

{

public void TriggerNewDay()

{

OnNewDay?.Invoke(); // Llama a todos los que estén suscritos al evento OnNewDay

}

public static GameTime Instance; // Variable estática para que otros scripts puedan acceder a esta clase

public int currentDay = 1; // Día actual en el juego (entero)

public float secondsPerDay = 10f; // Cuántos segundos reales equivalen a un "día" en el juego (número decimal)

private float timer; // Variable interna para contar el tiempo que pasa

public event Action OnNewDay; // Evento que se ejecuta cuando pasa un nuevo día

public CicloDiaNoche cicloDiaNoche; // Asigna tu CicloDiaNoche en el inspector

public TMP\_Text ui\_text; // Asigna tu TMP\_Text en el inspector

private void Awake() // Función que se llama automáticamente al iniciar el script

{

if (Instance == null) Instance = this; // Guarda una única instancia accesible desde cualquier script

}

void Update() // Se llama automáticamente una vez por frame

{

timer += Time.deltaTime; // Aumenta el temporizador según el tiempo real que pasó

if (cicloDiaNoche == null || ui\_text == null) return;

int hora = Mathf.FloorToInt(cicloDiaNoche.Hora);

int minutos = Mathf.FloorToInt((cicloDiaNoche.Hora - hora) \* 60f);

ui\_text.text = "Día: " + cicloDiaNoche.Dia + "\nHora: " + hora.ToString("00") + ":" + minutos.ToString("00");

if (timer >= secondsPerDay) // Si pasó el tiempo necesario para un día...

{

timer = 0f; // Reinicia el temporizador

currentDay++; // Avanza un día

OnNewDay?.Invoke(); // Llama a todos los que estén suscritos al evento OnNewDay

}

}

}

¿Por qué se programa aparte?

Esto es como un "reloj del juego", que avisa a todos los sistemas cuando pasa un día.

Las plantas escuchan ese evento para crecer.

4) Parcelas de tierra

Script: Plot.cs

using UnityEngine;

public class Plot : MonoBehaviour

{

public GameObject currentPlant; // La planta que tiene sembrada

public Transform plantPoint; // Dónde colocar la planta

public Inventario inventario;

public void PlantSeed(GameObject plantPrefab)

{

if (currentPlant == null)

{

currentPlant = Instantiate(plantPrefab, plantPoint.position, Quaternion.identity);

}

}

public void TryHarvest()

{

if (currentPlant != null)

{

var plant = currentPlant.GetComponent<PlantMultipleModels>();

if (plant != null && plant.IsFullyGrown())

{

plant.Harvest();

currentPlant = null;

inventario.addItem("tomate", 1); // Añadir un tomate al inventario

inventario.addItem("semilla\_tomate", 2); // Añadir dos semillas de tomate al inventario

}

}

}

}

¿Por qué se programa aparte?

Cada parcela es independiente y tiene su propia lógica para plantar o cosechar.

Al separar esto, podés tener muchas parcelas que funcionan igual.

TIP: En cada parcela de la escena, crea un hijo vacío y llámalo plantPoint. Ese es el punto exacto donde se planta la semilla.

5) Control del jugador  
Script: PlayerPlanting.cs

using UnityEngine;

public class PlayerPlanting : MonoBehaviour

{

public GameObject plantPrefab; // Prefab de la planta que vas a plantar

public Inventario inventario;

private Plot selectedPlot;

void Update()

{

if (Input.GetMouseButtonDown(0)) // Clic izquierdo

{

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

if (Physics.Raycast(ray, out RaycastHit hit))

{

Plot plot = hit.collider.GetComponent<Plot>();

if (plot != null && inventario.tipo.Contains("semilla\_tomate"))

{

selectedPlot = plot;

plot.PlantSeed(plantPrefab);

inventario.removeItem("semilla\_tomate");

}

}

}

if (Input.GetKeyDown(KeyCode.E) && selectedPlot != null)

{

selectedPlot.TryHarvest(); // Cosecha si está lista

}

}

}

¿Por qué se programa aparte?

Esto es lo que el jugador realmente controla.

Hace el clic para plantar y aprieta E para cosechar.

TIP: Este script puede ir en un objeto llamado “Player” o en la cámara principal.

Estructura del GameObject en la jerarquía:

1. En Unity, crea un Empty GameObject en la jerarquía y nómbralo, por ejemplo, PlantaTomate.

2. Dentro de ese GameObject, crea tres hijos que serán tus modelos:

EtapaSemilla (por ejemplo, un modelo pequeño o sprite de semilla)

EtapaBrote (un modelo un poco más grande, con hojitas)

EtapaMadura (modelo final, con frutas o flores)

PlantaTomate (con script "PlantMultipleModels")

├── EtapaSemilla (SetActive = true)

├── EtapaBrote (SetActive = false)

└── EtapaMadura (SetActive = false)

Scene

├── TimeManager (con GameTime.cs)

├── Player (con PlayerPlanting.cs)

├── Parcela1 (con BoxCollider + Plot.cs)

│ └── plantPoint (posiciona la planta aquí)

├── Parcela2

│ └── plantPoint

└── Prefabs

└── PlantaTomate (con PlantMultipleModels.cs)

├── EtapaSemilla

├── EtapaBrote

└── EtapaMadura

**Parte de Matías**

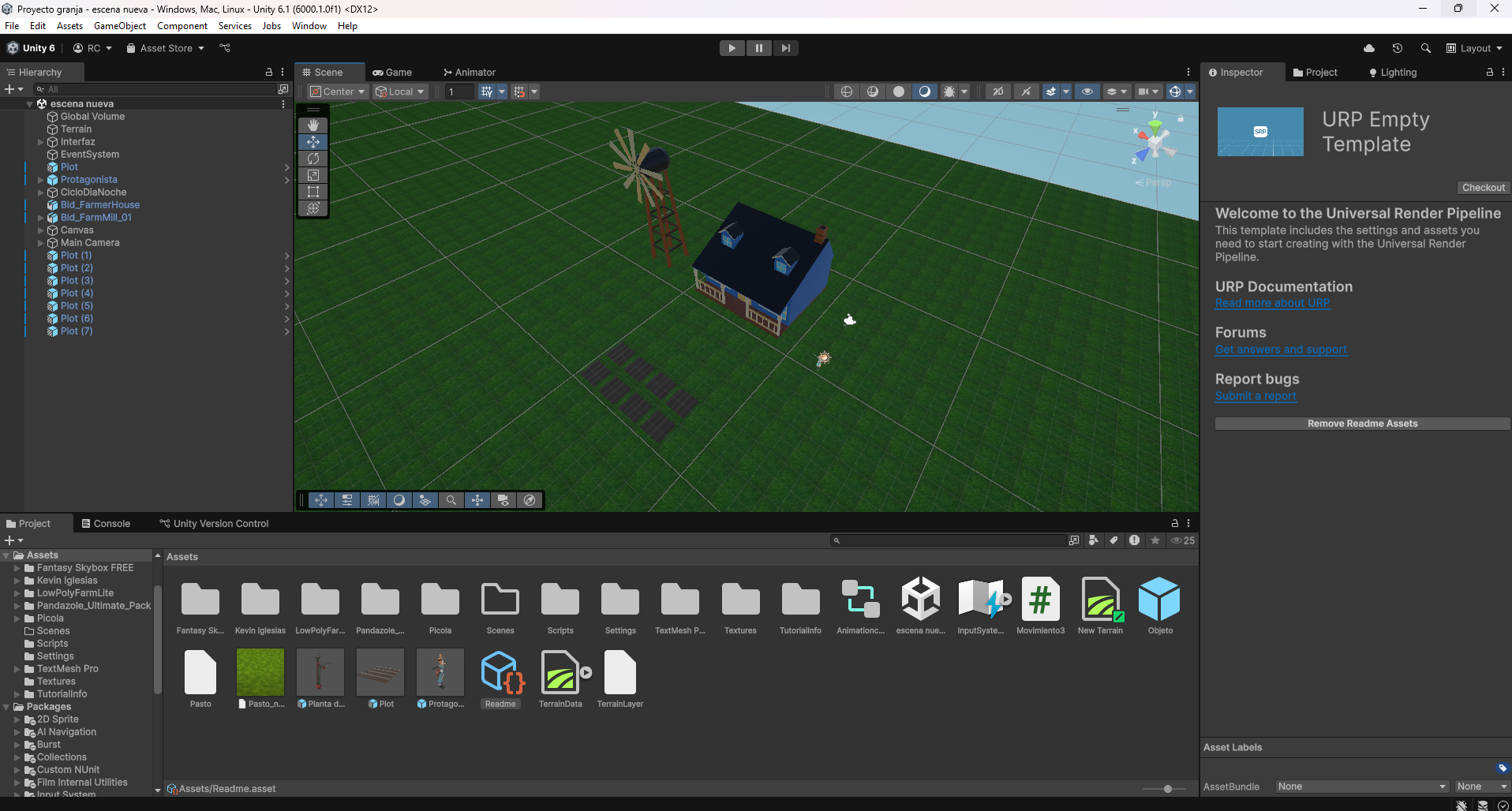
Se dedicó a la creación del inventario y la interfaz que ve el jugador. Al ser este mi primer proyecto programando en Unity y en 3D aprendí muchísimo, por sobre todas las cosas entendí como funciona tanto C# como la interfaz y las funcionalidades que brinda Unity al lenguaje de programación en forma de librerías. Aprendí a usar tanto listas como arrays y recorrerlas por medio de bucles/ciclos for, crear funciones Void, tanto públicas como privadas, y los condicionales (IF), herramientas esenciales para programar.

Arriba a la izquierda se puede ver la primera versión de cómo se veía el reloj que muestra la hora y a la derecha se muestra el inventario vacío.





Así se ve el reloj en un estado más avanzado de desarrollo.



El script que organiza el inventario y su interfaz es el siguiente:

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

using System.Collections.Generic;

using TMPro;

public class Inventario : MonoBehaviour

{

// Declaro el listado que contiene el tipo de los objetos que posee el jugador.

public List<string> tipo = new List<string>();

// Este otro listado contiene las cantidades de cada objeto que posee el jugador

public List<int> cantidad = new List<int>();

// Acá declaro un array que contiene las ranuras del inventario como interfaz

public GameObject[] slots;

public Sprite icon1; // Icono de semilla\_tomate

public Sprite icon2; // Icono de tomate

public Sprite empty\_icon; // Icono de la ranura de inventario vacía

private bool inventoryOpen; // Variable booleana que me dice sí el inventario está abierto o no

void Start()

{

hideInventory();

}

void hideInventory() // Función que esconde el inventario al jugador

{

inventoryOpen = false;

for (int i = 0; i < slots.Length; i++)

{

slots[i].SetActive(false);

}

}

void showInventory() // Función que muestra el inventario al jugador

{

inventoryOpen = true;

for (int i = 0; i < slots.Length; i++)

{

slots[i].SetActive(true);

}

if (tipo.Count > 0) // Sí el inventario tiene objetos, dibujarlos en la interfaz

{

for (int i = 0; i < tipo.Count; i++)

{

if (tipo[i] == "semilla\_tomate")

{

slots[i].GetComponent<Image>().sprite = icon1; // Asignamos un icono al objeto

}

if (tipo[i] == "tomate")

{

slots[i].GetComponent<Image>().sprite = icon2; // Asignamos un icono al objeto

}

slots[i].transform.Find("Cantidad").GetComponent<TMP\_Text>().text = cantidad[i].ToString(); // Mostramos la cantidad que tenemos de dicho objeto

}

}

else // Sí el inventario ya no tiene objetos, hacemos que la interfaz del inventario no muestre ningún objeto "fantasma"

{

for (int i = 0; i < slots.Length; i++)

{

slots[i].GetComponent<Image>().sprite = empty\_icon; // Icono de la ranura vacia

slots[i].transform.Find("Cantidad").GetComponent<TMP\_Text>().text = ""; // Ya no mostramos la cantidad de objetos porque no los hay

}

}

}

public void addItem(string item, int quantity) // Función que se encarga de añadir un objeto al inventario

{

if (tipo.Contains(item))

{

cantidad[tipo.IndexOf(item)] += quantity;

}

else

{

tipo.Add(item);

cantidad.Add(quantity);

}

}

public void removeItem(string item, int quantity = 1) // Función que se encarga de remover un objeto del inventario

{

int index\_item = tipo.IndexOf(item); // Agarramos el índice del objeto que vamos a remover

if (cantidad[index\_item] > quantity)

{

cantidad[index\_item] -= quantity; // Removemos la cantidad del objeto

}

else

{

// Removemos el objeto del inventario

tipo.RemoveAt(index\_item);

cantidad.RemoveAt(index\_item);

}

}

void Update() // Función que se activa cada frame

{

if (Input.GetKeyDown(KeyCode.I)) // Al tocar la tecla I

{

if (inventoryOpen) // Sí el inventario está abierto

{

hideInventory();

}

else // Sí el inventario NO está abierto

{

showInventory();

}

}

}

}