**Documentación**

Equipo 11

SODVI – FI – UNAM

Integrantes:

Monroy Salazar Diego Gustavo

Ventura Ricárdez Jeremy

Muriel González Diego

Contenido

[Bob Controller 4](#_Toc142594485)

[BobController.cs 4](#_Toc142594486)

[Pick Up Coins 6](#_Toc142594487)

[PickupCoins.cs 6](#_Toc142594488)

[Time Manager 8](#_Toc142594489)

[TimeManager.cs 8](#_Toc142594490)

[Skin Manager 10](#_Toc142594491)

[SkinManager.cs 10](#_Toc142594492)

[Powerups Manager 11](#_Toc142594493)

[PowerupsManager.cs 11](#_Toc142594494)

[Powerups 14](#_Toc142594495)

[Powerups.cs 14](#_Toc142594496)

[Shop Manager 16](#_Toc142594497)

[ShopManager.cs 16](#_Toc142594498)

[Shop Item SO 24](#_Toc142594499)

[ShopItemSO.cs 24](#_Toc142594500)

[Shop Template 27](#_Toc142594501)

[ShopTemplate.cs 27](#_Toc142594502)

[Player Controller 29](#_Toc142594503)

[PlayerController.cs 29](#_Toc142594504)

[Score Manager 36](#_Toc142594505)

[ScoreManager.cs 36](#_Toc142594506)

[Object Pooler 39](#_Toc142594507)

[ObjectPooler.cs 39](#_Toc142594508)

[Ground Generator 41](#_Toc142594509)

[GroundGenerator.cs 41](#_Toc142594510)

[Death Menu Manager 43](#_Toc142594511)

[DeathMenuManager.cs 43](#_Toc142594512)

[Ground Destroyer 46](#_Toc142594513)

[GroundDestroyer.cs 46](#_Toc142594514)

[Check Ground 47](#_Toc142594515)

[CheckGround.cs 47](#_Toc142594516)

[Move Background 48](#_Toc142594517)

[MoveBackground.cs 48](#_Toc142594518)

[Coin Generator 50](#_Toc142594519)

[CoinGenerator.cs 50](#_Toc142594520)

[Pause Menu 52](#_Toc142594521)

[PauseManu,cs 52](#_Toc142594522)

[Death Menu 55](#_Toc142594523)

[DeathMenu.cs 55](#_Toc142594524)

[Game Manager 56](#_Toc142594525)

[GameManager.cs 56](#_Toc142594526)

[Character Database 59](#_Toc142594527)

[CharacterDatabase.cs 59](#_Toc142594528)

[Character 60](#_Toc142594529)

[Character.cs 60](#_Toc142594530)

[Camera Controller 61](#_Toc142594531)

[CameraController.cs 61](#_Toc142594532)

[Main Menu 63](#_Toc142594533)

[MainMenu,cs 63](#_Toc142594534)

# Bob Controller

Este script se encarga de administrar la posición de Bob.

## BobController.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class BobController : MonoBehaviour  {      // The public variables can be seen and modified thru the UI      // Group of variables that set the move speed of "the bob", setted in the UI      public float moveSpeed;      public float speedLimit;      public float speedMultiplier;      public float speedIncreaseDistance;      private float speedDistanceCounter;      // The rigidbody of the bob, used for movement and physics      private Rigidbody2D bob;      // A Game Manager reference      public GameManager theGameManager;      // A Player Controller reference      public PlayerController thePlayerController;      // A reference to the Sprite of bob      public SpriteRenderer bobSprite;      // Start is called before the first frame update      void Start()      {          // Get the bob rigidbody          bob = GetComponent<Rigidbody2D>();          // Set the inital move speed for Bob          moveSpeed = theGameManager.startSpeed;          // Set the inital Sprite for Bob          bobSprite = GetComponent<SpriteRenderer>();      }      // Update is called once per frame      void Update()      {          // Change the Bob move speed to match the player speed, so he is always catching up          moveSpeed = thePlayerController.moveSpeed;          // Aply a force in the "x" axis of Bob while maintaining it´s velocity in the "y" axis          bob.velocity = new Vector2(moveSpeed, bob.velocity.y);      }  } |

# Pick Up Coins

Este script se encarga de administrar lo que pasa cuando el jugador recoge una moneda.

## PickupCoins.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class PickupCoins : MonoBehaviour  {      // Amount of coin currency to add to the player      public int coinsToGive;      // A reference to the Score Manager      private ScoreManager theScoreManager;      // A reference to the audio source      [SerializeField] AudioSource coinSFX;      // Start is called before the first frame update      void Start()      {          // Here we set the Score Manager using FindObjectOfType, in this way Unity handle the search of the desired object          // so we dont have to do it manually using the UI          theScoreManager = FindObjectOfType<ScoreManager>();      }      // Update is called once per frame      void Update()      {      }      // Buit in function in Unity that checks when another object with a 2d collider enters in our trigger zone      void OnTriggerEnter2D(Collider2D other)      {          if (other.gameObject.name == "Player")          {              theScoreManager.AddCoins(coinsToGive);              gameObject.GetComponent<SpriteRenderer>().enabled = false;              coinSFX.Play();          }      }  } |

# Time Manager

Este script se encarga de administrar el power up de SlowMo.

## TimeManager.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class TimeManager : MonoBehaviour  {      // Group of variables that set the slow down factor properties for time managment      public float slowdownFactor = 0.05f;      public float slowdownLength = 2f;      public float slowdownLengthCounter;      public bool paused;      void Start()      {          // Initializing the folowing variables          slowdownLengthCounter = slowdownLength;          paused = false;      }      void Update()      {          // If the game isn't paused          if (!paused)          {              // Change the time scale values so we create a slow down efect, after som time the time scale return to 1 (normal)              Time.timeScale += (1f / slowdownLength) \* Time.unscaledDeltaTime;              Time.fixedDeltaTime += (0.01f / slowdownLength) \* Time.unscaledDeltaTime;              slowdownLengthCounter -= (0.01f / slowdownLength);              // Clamp the time scale of the game to 1 or 0 so it doesn't go any further              Time.timeScale = Mathf.Clamp(Time.timeScale, 0f, 1f);              Time.fixedDeltaTime = Mathf.Clamp(Time.fixedDeltaTime, 0f, 0.01f);          }          if (paused)          {              Time.fixedDeltaTime = 1;          }      }      // Function that sets the variables for the Slowmotion efect      public void DoSlowmo()      {          Time.timeScale = slowdownFactor;          Time.fixedDeltaTime = Time.timeScale \* .02f;      }  } |

# Skin Manager

Este script se encarga de administrar la base de datos de skins.

## SkinManager.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class SkinManager : MonoBehaviour  {      // A reference to the Character Database, here we will store our Skins      public CharacterDatabase skinsDB;      // Start is called before the first frame update      void Start()      {      }      // Update is called once per frame      void Update()      {      }  } |

# Powerups Manager

Este script se encarga de los eventos de los powerups.

## PowerupsManager.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class PowerupsManager : MonoBehaviour  {      // Variables for the time managmet      private bool slowMo;      public float slowMoFactor;      // Variables to controll the powerups      private bool invencible;      private bool powerupActive;      public float powerupActiveDuration;      private float powerupDurationCounter;      // Variables to set the game normal and current speed      private float gameNormalSpeed;      private float gameSpeed;      // A reference to the Player controller      private PlayerController thePlayerController;      // A reference to the Time Manager      private TimeManager theTimeManager;      // A reference to the Game Manager      private GameManager theGameManager;      // Start is called before the first frame update      void Start()      {          // Here we set thePlayerController, theTimeManager and theGameManager using FindObjectOfType, in this way          // Unity handle the search of the desired object so we dont have to do it manually using the UI          thePlayerController = FindObjectOfType<PlayerController>();          theTimeManager = FindObjectOfType<TimeManager>();          theGameManager = FindObjectOfType<GameManager>();          gameNormalSpeed = 1;          Time.timeScale = gameNormalSpeed;      }      // Update is called once per frame      void Update()      {          // If any powerup is active          if (powerupActive)          {              // Substract to the powerupDurationCounter so it ends in powerupDurationCounter time              powerupDurationCounter -= Time.unscaledDeltaTime;              // If the powerupReset of the theGameManager is true              if (theGameManager.powerupReset)              {                  // Reset the power up related varaibles                  powerupDurationCounter = 0;                  theGameManager.powerupReset = false;              }              // If the slowMo is true and our time scale is 1 (normal)              if (slowMo && Time.timeScale >= 1)              {                  // Make a slowmo                  theTimeManager.DoSlowmo();              }              // If invencible is true              if (invencible)              {                  // Set invencible as true in thePlayerController                  thePlayerController.invencibleActive = true;              }              // Start decreasing the powerupDurationCounter so the power up ends              powerupDurationCounter -= Time.deltaTime;              // If the powerupDurationCounter is less or equal 0.09 seconds              if (powerupDurationCounter <= 0.09)              {                  // Reset the powerup variables because the powe up time ended                  gameSpeed = gameNormalSpeed;                  thePlayerController.invencibleActive = false;                  powerupActive = false;              }          }      }      // Function that activates a power up when called      public void ActivatePowerup(bool slowMoRecived, float slowMoFactorRecived, bool invencibleRecived, float durationRecived)      {          // Set the power up related variables so we now have a power up          slowMo = slowMoRecived;          slowMoFactor = slowMoFactorRecived;          invencible = invencibleRecived;          powerupDurationCounter = durationRecived;          gameSpeed = gameNormalSpeed;          powerupActive = true;      }  } |

# Powerups

Este script se encarga de decirle a cada objeto que sea un power up que hacer.

## Powerups.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class Powerups : MonoBehaviour  {      // Variables for the time managmet      public bool slowMo;      public float slowMoFactor;      // Variables to controll the powerups      public bool invencible;      public float powerupDuration;      private bool powerupActive;      // A reference to the Power ups Manager      public PowerupsManager thePowerupsManager;      // Start is called before the first frame update      void Start()      {          // Here we set thePowerupsManager using FindObjectOfType, in this way Unity handle the search of the desired object          // so we dont have to do it manually using the UI          thePowerupsManager = FindObjectOfType<PowerupsManager>();      }      // Update is called once per frame      void Update()      {      }      // Buit in function of Unity that checks when another object with a 2d collider enters in our trigger zone      void OnTriggerEnter2D(Collider2D other)      {          // If the Player collides with the power up item          if (other.name == "Player")          {              // Start the power up              thePowerupsManager.ActivatePowerup(slowMo, slowMoFactor, invencible, powerupDuration);          }          // Deactivate the power up item so it disapears          gameObject.SetActive(false);      }  } |

# Shop Manager

Este script se encarga de administrar la tienda, sea cargándola o administrando los eventos de equipado y comprado de skins.

## ShopManager.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  using TMPro;  using System;  public class ShopManager : MonoBehaviour  {      // Variable that count the amount of coins the player have      public int coins;      // A reference to the coins text in the UI      public TMP\_Text coinsUI;      // A array of our Scriptable Object Shop Items      public ShopItemSO[] shopItemsSO;      // A array of our Empty Game Object Shop Panels      public GameObject[] shopPanelsGO;      // A array of our empty Shop Panels      public ShopTemplate[] shopPanels;      // A reference to the Skins Data Base      public CharacterDatabase skinsDB;      // A variable that holds the name of the player skin      public string playerSkin;      // Array of all the purchase buttons at the store      public Button[] myPurchaseBtns;      // Array of all the equip buttons at the store      public Button[] myEquipBtns;      // The name of the Menu scene      public string Menu;      // Start is called before the first frame update      void Start()      {          // Activate "shopItemsSO.Length" Game Objects Shop Panels          for (int i = 0; i < shopItemsSO.Length; i++)          {              shopPanelsGO[i].SetActive(true);          }          // Get the player prefs          getPlayerPrefs();          // Update the coins UI text to display the amount of coins the player has          coinsUI.text = coins.ToString();          // Load the panels of the shop          LoadPanels();          // If it's the first time the game is launched, the player wont have a skin equiped, so we equip the default one          if (!PlayerPrefs.HasKey("PlayerEquipedSkin"))          {              // Set the Default skin at the players prefs              PlayerPrefs.SetString("PlayerEquipedSkin", "Originalli");              PlayerPrefs.SetString("SkinsBought", "Originalli");              // Equip the default skin              EquipSkin(0);          }          // Check the purcheasable skins          CheckPurcheseable();          // Check the equipable skins          CheckEquipable();          // Check the purcheasable skins          CheckPurcheseable();      }      // Update is called once per frame      void Update()      {      }      // Function that checks the purcheasable skins      public void CheckPurcheseable()      {          // Get the skins bought          string[] skinsBought = GetSkinsBought();          // For each skin bougth, deactive the buy button so the player cant buy it again          for (int i = 0; i < skinsBought.Length; i++)          {              // For each shop item              for (int j = 0; j < shopItemsSO.Length; j++)              {                  // If the player have enogh coins to buy the item wich must not be equipable, and the items name is not a bught skin                  if (coins >= shopItemsSO[i].baseCost && !shopItemsSO[j].equipable && j != 0 && shopItemsSO[j].title != skinsBought[i])                  {                      // Activate the purchase button of the item                      myPurchaseBtns[j].gameObject.SetActive(true);                      myPurchaseBtns[j].interactable = true;                  } // if not but the shop item is equipable or it's the default skin or the item name is equal to a bought skin                  else if (shopItemsSO[j].equipable || j == 0 || shopItemsSO[j].title == skinsBought[i])//                  {                      // Deactivate the purchase button of the item                      myPurchaseBtns[j].interactable = false;                      myPurchaseBtns[j].gameObject.SetActive(false);                  } // If not but the player don't have enought coins to buy it                  else if (coins < shopItemsSO[i].baseCost)                  {                      // Deactivate the purchase button of the item                      myPurchaseBtns[j].gameObject.SetActive(true);                      myPurchaseBtns[j].interactable = false;                  }              }          }      }      // Function that updates the equipability of each skin      public void CheckEquipable()      {          // Get the skins bought          string[] skinsBought = GetSkinsBought();          //For each skin bought, set the button interactable to true, else false          for (int i = 0; i < skinsBought.Length; i++)          {              for (int j = 0; j < shopItemsSO.Length; j++)              {                  // If the shop panel title is equal to our skin                  if (shopItemsSO[j].title == skinsBought[i])                  {                      // Activate the equip button                      myEquipBtns[j].gameObject.SetActive(true);                      myEquipBtns[j].interactable = true;                      shopItemsSO[j].equipable = true;                  }// If not and the Item shuld not be equipable                  else if (!shopItemsSO[j].equipable)                  {                      //  Deactivate the equip button                      myEquipBtns[j].gameObject.SetActive(false);                  }              }          }      }      // Function that return a list of strings with all the bought skins      public string[] GetSkinsBought()      {          // Get the skins bought from the player prefs          string skinsBoughtTmp = PlayerPrefs.GetString("SkinsBought");          // Create list of skins splitting the skinsBoughtTmp string by the "."          string[] skinsBought = skinsBoughtTmp.Split(".");          // For each skin bought, print its name in the debug log console          // for (int i = 0; i < skinsBought.Length; i++)          // {          //     Debug.Log(skinsBought[i].ToString());          // }          // Return the list of skins bought          return skinsBought;      }      // Function that gets the item number to buy and buy it      public void PurchaseItem(int btnNo)      {          // If the player have an equal or grater amount of coins than the base cost of the item to buy          if (coins >= shopItemsSO[btnNo].baseCost)          {              // Thake the base cost of coins away for the player              coins -= shopItemsSO[btnNo].baseCost;              // Update the coins UI text displayed              coinsUI.text = coins.ToString();              // Update the player prefs coins              PlayerPrefs.SetInt("Coins", coins);              // Get the character "btnNo" from the charactersDB              Character characterPurchased = skinsDB.GetCharacter(btnNo);              // Get the name of the purchased skin              string characterName = characterPurchased.characterName;              // Save the bought skin              saveBoughtSkin(characterName);              // Check the purcheseable skins              CheckPurcheseable();              // Check the equipable skins              CheckEquipable();              // Check the purcheseable skins              CheckPurcheseable();          }      }      // Function that equips the "btnNo" skin to the player, the skins are saved in our skinsDB      public void EquipSkin(int btnNo)      {          // Check if we can equip the "btnNo" skin to the player          CheckEquipable();          // Get the character "i" from the charactersDB          Character characterPurchased = skinsDB.GetCharacter(btnNo);          // Get the name of the skin          string characterName = characterPurchased.characterName;          //If the skin is unlocked, equip it (if the equip button is interactable)          PlayerPrefs.SetString("PlayerEquipedSkin", characterName);          myEquipBtns[btnNo].interactable = false;      }      // Function that Get the Player Prefs      public void getPlayerPrefs()      {          // Get the amount of coins the player have          coins = PlayerPrefs.GetInt("Coins");          // Get the player selected skin          playerSkin = PlayerPrefs.GetString("PlayerEquipedSkin");      }      // Function that gets a skin name and save it in the players pref      public void saveBoughtSkin(string skinName)      {          // Get the skins bought          string[] skinsBought = GetSkinsBought();          // If the "skinName" is saved in the "skinsBought" array, set skinAlredyBought to true          bool skinAlredyBought = Array.Exists(skinsBought, element => element == skinName);          //If the skin is alredy bought, dont add it to the bought skins, otherwise add it          if (!skinAlredyBought)          {              Array.Resize(ref skinsBought, skinsBought.Length + 1);              skinsBought[skinsBought.Length - 1] = skinName;          }          // Temporal string to save later on the skins bought          string skinsBoughtPrefsString = "";          // For each skin bought, add it to the "skinsBoughtPrefsString" string, using a "." as spacer          for (int i = 0; i < skinsBought.Length; i++)          {              // Get rid of the '.' for the last skin              if (i == skinsBought.Length - 1)              {                  skinsBoughtPrefsString += skinsBought[i];              } // Append the "i" skin to the "skinsBoughtPrefsString" string              else              {                  skinsBoughtPrefsString += skinsBought[i] + '.';              }          }          // Set the sking bought player prefs          PlayerPrefs.SetString("SkinsBought", skinsBoughtPrefsString);      }      // Function that Pupulates "shopItemsSO.Length" number of panels in the store, so they      // show the "shopItemsSO.Length" itens that are in sale      public void LoadPanels()      {          // For each Scriptable Object Item          for (int i = 0; i < shopItemsSO.Length; i++)          {              // Update the title and equipable bool of the item              shopPanels[i].titleTxt.text = shopItemsSO[i].title;              shopItemsSO[i].equipable = false;              //Get the character i from the charactersDB              Character characterSkin = skinsDB.GetCharacter(i);              // Create the Game Object for the skin that will populate the Item              GameObject skinGO = new GameObject(characterSkin.characterName, typeof(SpriteRenderer));              // Create a Sprite Renderer for our skin Game Object              SpriteRenderer skinSpriteRenderer = skinGO.GetComponent<SpriteRenderer>();              // Set the skin in the item              skinSpriteRenderer.sprite = characterSkin.characterSprite;              // Instanciate the skin Game Object in the Item              skinGO.transform.parent = shopPanels[i].transform;              skinGO.transform.localPosition = new Vector2(0, 27);              // Change the sprite sortin order so it's on top              skinSpriteRenderer.sortingOrder = 22;              skinGO.transform.localScale += new Vector3(-34f, -34f, -34f);              // Set the cost for the "i" item              shopPanels[i].costTxt.text = shopItemsSO[i].baseCost.ToString();          }      }      // Function that changes the scene to the "menu"      public void GoToMenu()      {          Application.LoadLevel(Menu);      }  } |

# Shop Item SO

Este script se encarga de crear un Scriptable Object para la tienda.

## ShopItemSO.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  // Create a new Menu option  [CreateAssetMenu(fileName = "ShopMenu", menuName = "Scriptable Objects/New Show Item", order = 1)]  // Class of SO, Scriptable Object, wich is like a data storage object  public class ShopItemSO : ScriptableObject  {      // Variables of the shop item Scriptable Object      public string title;      public string playerSpriteName;      public ShopTemplate theShopTemplate;      public int baseCost;      public bool equipable;  } |

Quit Confirmation

Este script se encarga de preguntarle al jugador si está seguro que quiere salir al menú.

QuitConfirmation.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class QuitConfirmation : MonoBehaviour  {      // The name of the Menu Scene      public string mainMenuLevel;      // The confirmation quit menu Game Object      public GameObject theConfirmationQuitMenu;      // Update is called once per frame      void Update()      {          // If the "q", "Q", "m" or "M" is pressed          if (Input.GetKey("q") || Input.GetKey("Q") || Input.GetKey("m") || Input.GetKey("M"))          {              // Confirm to quit to main menu              ConfirmQuitToMainMenu();          }          // If the "n", "N"or "Esc" is pressed          if (Input.GetKey("n") || Input.GetKey("N") || Input.GetKeyDown(KeyCode.Escape))          {              // Dont quit to the main menu              NotQuitToMainMenu();          }      }      // Function that asks the player if w¿he wants to quit to the main manu      public void ConfirmQuitToMainMenu()      {          // Activate the confirmation quit menu Game Object          theConfirmationQuitMenu.SetActive(true);          // Set the time scale to 1 (normal)          Time.timeScale = 1f;          // Quit to the main menu          Application.LoadLevel("Main Menu");      }      // Function that aborts the quit to main menu operation      public void NotQuitToMainMenu()      {          // Deactivate the confirmation quit menu Game Object          theConfirmationQuitMenu.SetActive(false);      }  } |

# Shop Template

Este script se encarga de cargar las skins en cada uno de los objetos de la tienda.

## ShopTemplate.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  public class ShopTemplate : MonoBehaviour  {      // Reference to the title and cost text of each item at the shop      public TMP\_Text titleTxt;      public TMP\_Text costTxt;      // Reference to the character database (skins database)      public CharacterDatabase skinsDB;      // A sprite renderer for the displayed item skin      public SpriteRenderer skinSprite;      // Selected skin counter      private int selectedSkin;      // Function that updates the player skin at the game      private void UpdatePlayerSkin(int selectedSkin)      {          // Get the "selectedSkin" player skin          Character player = skinsDB.GetCharacter(selectedSkin);          // Set the skin sprite as the "selectedSkin"          skinSprite.sprite = player.characterSprite;      }      // Funtion that applies the player skin      private void LoadSkin()      {          // Get the selected skin of the player at the player prefs          selectedSkin = PlayerPrefs.GetInt("selectedSkin");      }      // Funtion that saves the selected skin of the player      private void SaveSkin(int selectedSkin)      {          // Set the selected skin of the player at the player prefs          PlayerPrefs.SetInt("selectedSkin", selectedSkin);      }  } |

# Player Controller

Este script se encarga de administrar los movimientos y posición del jugador, así como la skin equipada.

## PlayerController.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class PlayerController : MonoBehaviour  {      /\*       \* The public variables can be seen and modified thru the UI       \*/      // Variables that set the move speed and jump force aplied to the player, setted in the UI      public float moveSpeed;      public float speedLimit;      public float speedMultiplier;      public float speedIncreaseDistance;      private float speedDistanceCounter;      public float jumpForce;      // The time the player can hold the jump button to jump higher      public float jumpTime;      private float jumpTimeCounter;      // The rigidbody of the player, used for movement and physics      private Rigidbody2D player;      // Bool to see if the player is in the ground      public bool grounded;      // The layer wich is suposed to act as ground to let the player jump when standing on it      public LayerMask whatIsGround;      // Our Ground Ckeck obj inside the player      public Transform groundCheck;      // The radius of the Ground Check circle beneath out player      public float groundCheckRadius;      // Collider to register if the player is touching the floor      private Collider2D myCollider;      public bool invencibleActive = false;      // The Game Manager reference      public GameManager theGameManager;      // A reference to the Character database      public CharacterDatabase skinsDB;      // A reference to the Character (skin of the player)      private Character characterSkin;      // A reference to "the bob" Game Object      public GameObject theBob;      // A reference to the SFX Manager      public SFXManager theSFXManager;      // Start is called before the first frame update      void Start()      {          // Get the player rigidbody          player = GetComponent<Rigidbody2D>();          // Get the collider of the player          myCollider = GetComponent<Collider2D>();          // Initialize jumpTimeCounter          jumpTimeCounter = jumpTime;          // Set the speedDistanceCounter to later on increase the movement speed of the player          speedDistanceCounter = speedIncreaseDistance;          // Get the skin equiped of the player          string skinEquiped = PlayerPrefs.GetString("PlayerEquipedSkin");          // For each skin in our skins database search four our equiped skin and set it to the "characterSkin" of our player          for (int i = 0; i < skinsDB.CharacterCount; i++)          {              // If the name of our equiped skin is equal to the "i" skin at our skins database              if (skinEquiped == skinsDB.GetCharacter(i).characterName)              {                  //Get the character i from the charactersDB                  characterSkin = skinsDB.GetCharacter(i);              }          }          // Create a Sprite Renderer for the player skin          SpriteRenderer playerSprite = GetComponent<SpriteRenderer>();          // Set the player equiped skin          playerSprite.sprite = characterSkin.characterSprite;          // Create a Sprite Renderer for "the bob" skin          SpriteRenderer bobSprite = theBob.GetComponent<SpriteRenderer>();          // Set the bob equiped skin          bobSprite.sprite = characterSkin.bobHatlessCharacterSprite;          bobSprite.transform.localScale += new Vector3(-0.3f, -0.3f, -0.3f);      }      // Update is called once per frame      void Update()      {          // Grounded state depends of the circle in the position of our groundCheck object with groundCheckRadius radius and          // comparing if it's touching whatIsGround          grounded = Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius, whatIsGround);          // If the player is beyond the "speedDistanceCounter" and it's move speed is not above the speed limit          if (transform.position.x > speedDistanceCounter && moveSpeed < speedLimit)          {              // Increase the "speedDistanceCounter" by "speedIncreaseDistance" so the speed increments in x + y meters next time (more distance)              speedDistanceCounter += speedIncreaseDistance;              // Change the value of the "speedIncreaseDistance" using the speedMultiplier              speedIncreaseDistance \*= speedMultiplier;              // Change the value of the "moveSpeed" using the speedMultiplier              moveSpeed \*= speedMultiplier;          }          // Aply a force in the "x" axis of the player while maintaining it�s velocity in the "y" axis          player.velocity = new Vector2(moveSpeed, player.velocity.y);          /\*           \*  If SPACE, LEFT-CLICK, UP-ARROW or W are pressed and the player is in the ground, he can jump          \*/          if (Input.GetKeyDown(KeyCode.Space) || Input.GetMouseButtonDown(0) || Input.GetKey("up") || Input.GetKey("w"))          {              if (grounded)              {                  // Maintaining the player "x" axis velocity while adding a jumpforce equal to the jump force value in the "y" axis                  player.velocity = new Vector2(player.velocity.x, jumpForce);                  theSFXManager.PlayJumpSound();              }          }      }      // When a object with a box collider touches another object with a box collider      public void OnCollisionEnter2D(Collision2D collision)      {          // If our player collides with a Game object that have the "killBox" tag          if (!invencibleActive && collision.gameObject.tag == "killBox")//killboxTag == "killBox" &&          {              // Restart the game              theGameManager.RestartGame();          }          // If the player is invencible and collides with a wall, spikes or pothole, set the collision GO trigger to true          if (invencibleActive && (collision.gameObject.name == "wall" || collision.gameObject.name == "spikes" || collision.gameObject.name == "pothole"))          {              collision.gameObject.GetComponent<Collider2D>().isTrigger = true;          }      }  } |

SFX Manager

Este script se encarga de administrar los sonidos del juego.

SFXManager.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class SFXManager : MonoBehaviour  {      public AudioSource audio;      public AudioClip click;      public AudioClip buy;      public AudioClip equip;      public AudioClip jump;      public static SFXManager theSFXManager;      // Function that plays a sound when the Player clicks on something      public void PlayeClickSound()      {          audio.PlayOneShot(click);      }      // Function that plays a sound when the Player buys a skin      public void PlayBuySound()      {          audio.PlayOneShot(buy);      }      // Function that plays a sound when the Player equips a skin      public void PlayEquipSound()      {          audio.PlayOneShot(equip);      }      // Function that plays a sound when the Player jumps      public void PlayJumpSound()      {          audio.PlayOneShot(jump);      }  } |

# Score Manager

Este script se encarga de administrar la puntuación actual y la máxima del jugador.

## ScoreManager.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  // To use UI stuff  using UnityEngine.UI;  public class ScoreManager : MonoBehaviour  {      // Our scores text in the UI      public Text scoreText;      public Text HighscoreText;      public Text coinsText;      // Scores counters to update text      public float scoreCounter;      public float highScoreCounter;      public int coinsCounter;      public float pointsPerSecond;      // If the player is still alive bool      public bool scoreIncreasing;      // Start is called before the first frame update      void Start()      {          // If the player have a High Score saved, set it          if (PlayerPrefs.HasKey("HighScore"))          {              // Get the value stored in the HighScore PlayerPref              highScoreCounter = PlayerPrefs.GetFloat("HighScore");          }          // If the player have coins saved, set them          if (PlayerPrefs.HasKey("Coins"))          {              // Get the value stored in the Coins PlayerPref              coinsCounter = PlayerPrefs.GetInt("Coins");          }      }      // Update is called once per frame      void Update()      {          // If the player is still alive or the game isn't paused          if (scoreIncreasing)          {              // Add the respective points respective to the time the frame takes to hapen, so that in 1 sec we end up having              // pointsPerSecond points in our scoreConter              scoreCounter += pointsPerSecond \* Time.deltaTime;          }          // If the player Score is greater than his previuos high score update the high score          if (scoreCounter > highScoreCounter)          {              // Set the high score counter              highScoreCounter = scoreCounter;              // Save the High Score of the player in his player prefs              PlayerPrefs.SetFloat("HighScore", highScoreCounter);          }          // Update the score text          scoreText.text = "Dist.: " + Mathf.Round(scoreCounter) + " Km";          // Update the high score text          HighscoreText.text = "Dist. max.: " + Mathf.Round(highScoreCounter) + " Km";          // Update the coins text          coinsText.text = coinsCounter.ToString();      }      // Function that adds "coinsToAdd" coins to the player      public void AddCoins(int coinsToAdd)      {          // Add "coinsToAdd" coins to the "coinsCounter" variable of the player          coinsCounter += coinsToAdd;      }  } |

# Object Pooler

Este script se encarga de administrar los objetos del mundo para destruirlos mientras el jugador avanza, esto con la finalidad de tener buen rendimiento al jugar.

## ObjectPooler.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class ObjectPooler : MonoBehaviour  {      public GameObject pooledObject;      public int pooledAmount;      // List of our pooled Game Objects      List<GameObject> pooledObjects;      // Start is called before the first frame update      void Start()      {          // Create a list for our pooled rojects          pooledObjects = new List<GameObject>();          // For each number in the "pooledAmount" variable, add a new Game Object to the Pooled Objects list          for (int i = 0; i < pooledAmount; i++)          {              GameObject obj = (GameObject)Instantiate(pooledObject);              obj.SetActive(false);              pooledObjects.Add(obj);          }      }      // Update is called once per frame      void Update()      {      }      // Function that return a Game Object      public GameObject GetPooledObject()      {          // For each pooled object          for (int i = 0; i < pooledObjects.Count; i++)          {              // If the obj is active in the scene              if (pooledObjects[i].activeInHierarchy)              {                  // Return the pooled object                  return pooledObjects[i];              }          }          // Else, add a new object and return it          GameObject obj = (GameObject)Instantiate(pooledObject);          obj.SetActive(false);          pooledObjects.Add(obj);          return obj;      }  } |

# Ground Generator

Este script se encarga de generar el mundo enfrente del jugador.

## GroundGenerator.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class GroundGenerator : MonoBehaviour  {      // Get the ground objects      public GameObject theGround;      // Create the instance of the floor generation point      public Transform generationPoint;      // The distance between ground      public float distanceBetween;      public float distanceBetweenMin;      public float distanceBetweenMax;      // The width of the ground to generate      private float groundWidth;      // Ground array      public GameObject[] theGroundArray;      // Counter that will select wich ground is generated      private int groundSelector;      // List of the diferent ground widths      private float[] groundWidths;      // A reference to the Object Pool      public ObjectPooler theObjectPool;      // A array of Object Poolers      public ObjectPooler[] theObjectPools;      // A reference to the coin generator      private CoinGenerator theCoinGenerator;      // Start is called before the first frame update      void Start()      {          // Set the width of the ground          groundWidth = theGround.GetComponent<BoxCollider2D>().size.x;          // Here we set the coin generator using FindObjectOfType, in this way Unity handle the search of the desired object          // so we dont have to do it manually using the UI          theCoinGenerator = FindObjectOfType<CoinGenerator>();      }      // Update is called once per frame      void Update()      {          // If the generation point is behind the transform.position.x, generate more ground ahead          if (transform.position.x < generationPoint.position.x)          {              // Random distance between platforms generation              distanceBetween = Random.Range(distanceBetweenMin, distanceBetweenMax);              // Move the position of the object              transform.position = new Vector3(transform.position.x + groundWidth + distanceBetween, transform.position.y, transform.position.z);              groundSelector = Random.Range(0, theGroundArray.Length);              // Create the ground              Instantiate(theGroundArray[groundSelector], transform.position, transform.rotation);              // Add coins              theCoinGenerator.SpawnCoins(new Vector3(transform.position.x, transform.position.y + 1f, transform.position.z));          }      }  } |

# Death Menu Manager

Este script se encarga de administrar los eventos dentro del menú de muerte.

## DeathMenuManager.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  // To use UI stuff  using UnityEngine.UI;  public class DeathMenuManager : MonoBehaviour  {      // The scores texts in the UI      public Text scoreText;      public Text highScoreText;      private float currentScore;      private float maxScore;      // The coin text in the UI      public Text coinsText;      private int coins;      // A reference to the character database (skins database)      public CharacterDatabase skinsDB;      // A character (skin)      private Character characterSkin;      // A reference to the player Game Object      public GameObject thePlayer;      // A reference to the bob Game Object      public GameObject theBob;      // Start is called before the first frame update      void Start()      {          // If the player have a High Score saved, set it          if (PlayerPrefs.HasKey("HighScore"))          {              // Get the value stored in the HighScore PlayerPref              maxScore = PlayerPrefs.GetFloat("HighScore");          }          // If the player have coins saved, set them          if (PlayerPrefs.HasKey("Coins"))          {              // Get the value stored in the Coins PlayerPref              coins = PlayerPrefs.GetInt("Coins");          }          // If the player have a current score saved, set ir          if (PlayerPrefs.HasKey("CurrentScore"))          {              // Get the value stored in the Curent Scocre PlayerPref              currentScore = PlayerPrefs.GetFloat("CurrentScore");          }          else          {              currentScore = 0f;          }          // Update the score, high score and coins texts of the UI          scoreText.text = "Dist.: " + Mathf.Round(currentScore) + " Km";          highScoreText.text = "Dist. m�x.: " + Mathf.Round(maxScore) + " Km";          coinsText.text = coins.ToString();          // Get the player equiped skin          string skinEquiped = PlayerPrefs.GetString("PlayerEquipedSkin");          // For each skin at our skins database search the equiped one          for (int i = 0; i < skinsDB.CharacterCount; i++)          {              // If the name of our equiped skin is equal to the "i" skin at our skins database              if (skinEquiped == skinsDB.GetCharacter(i).characterName)              {                  //Get the character i from the charactersDB                  characterSkin = skinsDB.GetCharacter(i);              }          }          // Create a Sprite Renderer for the player skin          SpriteRenderer playerSprite = thePlayer.GetComponent<SpriteRenderer>();          // Set the player equiped skin          playerSprite.sprite = characterSkin.characterHatlessDeadSprite;          // Resize and position the skin          playerSprite.transform.localScale += new Vector3(0.05f, 0.05f, 0.05f);          playerSprite.transform.Rotate(Vector3.forward \* 2);          // Create a Sprite Renderer for the player skin          SpriteRenderer bobSprite = theBob.GetComponent<SpriteRenderer>();          // Set the bob equiped skin          bobSprite.sprite = characterSkin.bobCharacterSprite;          // Resize and position the skin          bobSprite.transform.localScale += new Vector3(0.2f, 0.2f, 0.2f);          bobSprite.transform.Rotate(Vector3.forward \* 2);      }      // Function that restart the game by reloading the game scene      public void RestarGame()      {          Application.LoadLevel("EndlessRuner");      }      // Function that loads the main menu scene      public void QuitToMainMenu()      {          Application.LoadLevel("Main Menu");      }  } |

# Ground Destroyer

Este script se encarga de destruir el piso detrás del jugador, esto con la finalidad de mantener un buen rendimiento en el juego.

## GroundDestroyer.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class GroundDestroyer : MonoBehaviour  {      // The point where to destroy the ground      public GameObject groundDestructionPoint;      // Start is called before the first frame update      void Start()      {          // Get the ground destruction point from the scene          groundDestructionPoint = GameObject.Find("GroundDestructionPoint");      }      // Update is called once per frame      void Update()      {          // If the position of the scene object is less than the groundDestructionPoint          if (transform.position.x < groundDestructionPoint.transform.position.x)          {              // Then destroy the ground              Destroy(gameObject);          }      }  } |

# Check Ground

Este script revisa si el jugador está tocando o no el suelo.

## CheckGround.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class CheckGround : MonoBehaviour  {      // Variable that says if the player is touching the ground or not      public static bool isGrounded;      // Function that identifies if the player is touching the ground      private void OnTriggerEnter2D(Collider2D collision)      {          // The player is touching the ground          isGrounded = true;      }      // Function that identifies if the player is at the air      private void OnTriggerExit2D(Collider2D collision)      {          // The player is not touching the ground          isGrounded = false;      }  } |

# Move Background

Este script se encarga de mover el fondo para tener un efecto parallax.

## MoveBackground.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class MoveBackground : MonoBehaviour  {      // The lenght of the element with the script      private float length;      // The start position of the object with the script      private float startPosition;      // A reference to the camera Game Object      private GameObject camera;      // The amount of parallax effect made private to the rest of scripts but available to edit in the UI      [SerializeField] private float parallaxEffect;      // Start is called before the first frame update      void Start()      {          // Fetch the camera to the Camera          camera = GameObject.Find("Main Camera");          // Get the "x" axis position of the object with the script          startPosition = transform.position.x;          // Get the lengt of the Sprite Renderer          length = gameObject.GetComponent<SpriteRenderer>().bounds.size.x;      }      // Update is called once per frame      void Update()      {          // The position of the object with the script over time          float temp = (camera.transform.position.x \* (1 - parallaxEffect));          // Distance to move the background acording to the camera and parallax effect amount          float distance = (camera.transform.position.x \* parallaxEffect);          // Move the "x" axis of the object with the script while maintaing its "y" and "z" vectors          transform.position = new Vector3(startPosition + distance, transform.position.y, transform.position.z);          // If the temp value is greater than the start position and the length added, then move the background to the right          if (temp > startPosition + length)          {              startPosition += length;              // Else, if the temp value is lower than the start position and the length substracted, then move the background to the left          }          else if (temp < startPosition - length)          {              startPosition -= length;          }      }  } |

# Coin Generator

Este script se encarga de generar monedas en el mundo.

## CoinGenerator.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class CoinGenerator : MonoBehaviour  {      // A reference to the coin pool      public ObjectPooler coinPool;      public float distanceBetweenCoins;      // Start is called before the first frame update      public void SpawnCoins(Vector3 startPosition)      {          // Get a coin          GameObject coin1 = coinPool.GetPooledObject();          // Set the coin in the game          coin1.transform.position = startPosition;          coin1.SetActive(true);          // Get a coin          GameObject coin2 = coinPool.GetPooledObject();          // Set the coin in the game          coin2.transform.position = new Vector3(startPosition.x - distanceBetweenCoins, startPosition.y, startPosition.z);          coin2.SetActive(true);          // Get a coin          GameObject coin3 = coinPool.GetPooledObject();          // Set the coin in the game          coin3.transform.position = new Vector3(startPosition.x + distanceBetweenCoins, startPosition.y, startPosition.z);          coin3.SetActive(true);      }  } |

# Pause Menu

Este script se encarga de administrar los eventos que ocurren en el menú de pausa.

## PauseManu,cs

|  |
| --- |
| using UnityEngine.UI;  using UnityEngine.EventSystems;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class PauseMenu : MonoBehaviour  {      // Name of the main menu scene      public string mainMenuLevel;      // A reference to the pause menu      public GameObject thePauseMenu;      // A reference to the quit confirmation menu      public GameObject theQuitConfirmationMenu;      // A reference to the time manager      public TimeManager theTimeManager;      void Start()      {          // Here we set the time manager using FindObjectOfType, in this way Unity handle the search of the desired object          // so we dont have to do it manually using the UI          theTimeManager = FindObjectOfType<TimeManager>();      }      // Update is called once per frame      void Update()      {          // If the player press "Esc" or "p"          if (Input.GetKeyDown(KeyCode.Escape) || Input.GetKeyDown(KeyCode.P))          {              // Pause the game              theTimeManager.paused = true;              PauseGame();          }          // If the game is paused and the player press "Esc" or "p"          if (Time.timeScale <= 0 && (Input.GetKeyDown(KeyCode.Escape) || Input.GetKeyDown(KeyCode.P)))          {              // Unpause the game              theTimeManager.paused = false;              UnpauseGame();          }          // If the player press "r" or "R"          if (Input.GetKey("r") || Input.GetKey("R"))          {              // Restart the game              RestarGame();          }          // If the player press "q", "Q", "m" or "M"          if (Input.GetKey("q") || Input.GetKey("Q") || Input.GetKey("m") || Input.GetKey("M"))          {              // Quit to main menu              QuitToMainMenu();          }      }      // Function that pauses the game      public void PauseGame()      {          // Freeze the time of the game so it's paused          Time.timeScale = 0f;          thePauseMenu.SetActive(true);          theTimeManager.paused = true;      }      // Function that unpauses the game      public void UnpauseGame()      {          // Unfreeze the time of the game so it's unpaused          Time.timeScale = 1f;          thePauseMenu.SetActive(false);          theTimeManager.paused = false;      }      // Function that restarts the game      public void RestarGame()      {          // Close the pause menu          thePauseMenu.SetActive(false);          // Set the time to 1 (normal)          Time.timeScale = 1f;          // Reset the player          FindObjectOfType<GameManager>().ResetPlayer();      }      // Function that quits to the main menu      public void QuitToMainMenu()      {          // Oppen the quit confirmation menu          theQuitConfirmationMenu.SetActive(true);      }  } |

# Death Menu

Este script se encarga de administrar los eventos dentro de la pestaña de muerte.

## DeathMenu.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class DeathMenu : MonoBehaviour  {      // The name of the main menu      public string mainMenuLevel;      // Function that restarts the game      public void RestarGame()      {          // Load the game scene          Application.LoadLevel("EndlessRuner");      }      // Function that quits to the main menu      public void QuitToMainMenu()      {          // Load the main menu scene          Application.LoadLevel("Main Menu");      }  } |

# Game Manager

Este script se encarga de administrar lo que le pasa al jugador cuando pone pausa, muere e inicia el juego.

## GameManager.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class GameManager : MonoBehaviour  {      // Reference to the position of the platform generator      public Transform platformGenerator;      public Vector3 platformStartPoint;      // Reference to the player      public PlayerController thePlayer;      // Starting point of the player      private Vector3 playerStartPoint;      // Start speed of the player      public float startSpeed;      // Reference to the bob      public BobController theBob;      // Starting point of the bob      private Vector3 bobStartPoint;      // Array of ground      private GroundDestroyer[] groundList;      // Reference to the score manager      private ScoreManager theScoreManager;      // Reference to the death menu      public DeathMenu theDeadthMenu;      // The name of the death scene      public string theDeathScene;      // Reference to the pause button      public PauseMenu thePauseButton;      public bool powerupReset;      // Start is called before the first frame update      void Start()      {          // Set the position of the platform start point          platformStartPoint = platformGenerator.position;          // Set the player start point          playerStartPoint = thePlayer.transform.position;          // Here we set the score manager using FindObjectOfType, in this way Unity handle the search of the desired object          // so we dont have to do it manually using the UI          theScoreManager = FindObjectOfType<ScoreManager>();          // Set the player move speed          thePlayer.moveSpeed = startSpeed;      }      // Update is called once per frame      void Update()      {      }      // Function that restarts the game      public void RestartGame()      {          // Stop increasing the score          theScoreManager.scoreIncreasing = false;          // Deactivate the player obj to restart it          thePlayer.gameObject.SetActive(false);          // Save the score and coins of the payer          PlayerPrefs.SetFloat("CurrentScore", theScoreManager.scoreCounter);          PlayerPrefs.SetInt("Coins", theScoreManager.coinsCounter);          // Activate the game menu          theDeadthMenu.gameObject.SetActive(true);          // Change the scene to the death scene          Application.LoadLevel(theDeathScene);      }      // Function that resets the player      public void ResetPlayer()      {          // Deactivate the death menu          theDeadthMenu.gameObject.SetActive(false);          // The array of ground is going to be all the platforms with the type/script PlatformDestroyer          groundList = FindObjectsOfType<GroundDestroyer>();          // Make all the floor ahead inactive          for (int i = 0; i < groundList.Length; i++)          {              groundList[i].gameObject.SetActive(false);          }          // Reset the position of the player and the platform generator to the start position          thePlayer.transform.position = playerStartPoint;          thePlayer.moveSpeed = startSpeed;          // Reset the platform generator point position          platformGenerator.transform.position = platformStartPoint;          // Save the current coins collected          PlayerPrefs.SetInt("Coins", theScoreManager.coinsCounter);          // Activate the player obj after restarting it          thePlayer.gameObject.SetActive(true);          // Reset the score          theScoreManager.scoreCounter = 0;          theScoreManager.scoreIncreasing = true;          powerupReset = true;      }  } |

# Character Database

Este script se encarga de administrar la base de datos de skins.

## CharacterDatabase.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  // Add an option to the asset menu called character database  [CreateAssetMenu]  public class CharacterDatabase : ScriptableObject  {      // Array of characters (skins)      public Character[] character;      // Function that return the amount of characters (skins) saved      public int CharacterCount      {          get          {              return character.Length;          }      }      // Function that return the character (skin) at the index provided      public Character GetCharacter(int index)      {          return character[index];      }  } |

# Character

Este script se encarga de administrar la información independiente de cada skin del juego.

## Character.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  [System.Serializable]  public class Character  {      // Skin name      public string characterName;      // Player with hat      public Sprite characterSprite;      // Car with out hat      public Sprite characterHatlessDeadSprite;      // Bob with out hat      public Sprite bobHatlessCharacterSprite;      // Bob with hat      public Sprite bobCharacterSprite;  } |

# Camera Controller

Este script se encarga de administrar el movimiento de la cámara en el juego.

## CameraController.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class CameraController : MonoBehaviour  {      // Create an instance to get the player      public PlayerController player;      // Variable to store the last position of the player      private Vector3 lastPlayerPosition;      // Variable to store the distance to move the camera so it follows the player      private float distanceToMove;      // Start is called before the first frame update      void Start()      {          // Get the player in the scene          player = FindObjectOfType<PlayerController>();          // Set the position of the player to the lastPlayerPosition variable          lastPlayerPosition = player.transform.position;      }      // Update is called once per frame      void Update()      {          // Get the distance to move the camera          distanceToMove = player.transform.position.x - lastPlayerPosition.x;          // Move the camera          transform.position = new Vector3(transform.position.x + distanceToMove, transform.position.y, transform.position.z);          // Set the lastPlayerPosition to the current position, so the movement continues thru the update loop          lastPlayerPosition = player.transform.position;      }  } |

# Main Menu

Este script se encarga de administrar los eventos que ocurren en el menú principal, tales como inicar el juego, ir a la tienda, salir del juego y cambiar las skins cada vez que se entra al menú principal.

## MainMenu,cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using System;  public class MainMenu : MonoBehaviour  {      // Name of the game scene      public string PlayGameLevel;      // Name of the shop scene      public string GoToShop;      // Reference to the character database (skins database)      public CharacterDatabase skinsDB;      // Reference to a character(skin)      private Character characterSkin;      // Reference to the player      public GameObject thePlayer;      // Reference to the bob      public GameObject theBob;      void Start()      {          // Generate a number between 0 and the amoun of skins in the game -1          int selectRandomSkin = UnityEngine.Random.Range(0, skinsDB.CharacterCount - 1);          //Get the character i from the charactersDB          characterSkin = skinsDB.GetCharacter(selectRandomSkin);          // Create a Sprite renderer for the player          SpriteRenderer playerSprite = thePlayer.GetComponent<SpriteRenderer>();          // Equip the skin;          playerSprite.sprite = characterSkin.characterSprite;          // Position the skin;          playerSprite.transform.localScale += new Vector3(0.05f, 0.05f, 0.05f);          // Create a Sprite renderer for the bob          SpriteRenderer bobSprite = theBob.GetComponent<SpriteRenderer>();          // Equip the skin;          bobSprite.sprite = characterSkin.bobHatlessCharacterSprite;          // Position the skin;          bobSprite.transform.localScale += new Vector3(0.2f, 0.2f, 0.2f);      }      // Function that loads the game scene      public void PlayGame()      {          Application.LoadLevel(PlayGameLevel);      }      // Function that loads the store scene      public void EnterStore()      {          Application.LoadLevel(GoToShop);      }      // Function that quits the game      public void QuitGame()      {          Application.Quit();      }  } |