ANNEX

VIRTUAL REALITY TO ENHANCE SAFETY AND HEALTH IN CONSTRUCTION

ONLINE MULTIPLAYER SERIOUS GAME.

Treball realitzat per:
Marc Arnau Navarro

Dirigit per:
Francisco Javier Mora Serrano
Ignacio Valero López

Grau en:
Enginyeria Civil

Barcelona, setembre de 2018

Departament d’Enginyeria Civil i Ambiental
This Annex is my explanation of all my written scripts of this project.

It goes from most important scripts to less important (although the order could be arguable). In here I try to explain how C# works while programming in Unity3D, in what I think is an understandable and easy way.

I hope that the person who reads this takes in account the number of late night hours put in this and that will be able to forgive me if there are typos or any other kind of mistakes.

Enjoy.
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

// We type this to tell this script that it'll have Networked components.

/* Probably the most important script of all. THIS WILL CONTROL ALL THE SCRIPTS ATTACHED TO THE PLAYER.
Note that when you are using Networking your player prefab gets a copy of itself
in every client/host. So, you may be player1 in your computer and you are seeing the copy of player2 in the screen and vice versa for the person that is playing with you.

Since you would like to control your player and ONLY your player, and you don't want other persons to control yours, we apply this script, that enables all the scripts attached to the prefab of the player only if it is LocalPlayer.

That means that when you are a client the software will run through all the copies of player being displayed and will enable the scripts only for the copy that you are supposed to control. Note that in order for this to work you have to disable them manually in the inspector first.

And how am I going to see what other players do then? Well, that's why you have the Network Identity - which basically says that the gameObject it is attached to exists in the network, and has to be checked as LocalPlayerAuthority for the player prefab -, the Network Transform - that updates the position of the gameObject it is attached to to all the other clients - and the Network Animator - that in case you have animations in your game it will update them too -. Note how this 3 components have to be enabled in the inspector! */

{
    public UnityStandardAssets.Characters.FirstPerson.FirstPersonController fpsController; /*
    We are defining the script FirstPersonController under the name "fpsController". That fact that it is "public" means that we will have to drag it into the inspector. Usually the way to declare things for the script is not so long and messy, it is just because here we're using the Character Controller provided
34 by unity and it is done like this.*/
35 public Camera fpsCamera; //Declare our Camera named fpsCamera.
36 public AudioListener audioListener; //Declare our AudioListener.
37 public LiniaDeVida liniaDeVida; //Goes on with several other scripts...Easy uh?
38 public Helmet helmetScript;
39 public FallDamage2 fallDamage2;
40 public Player_Net player_net;
41 public GameObject UI;
42 public Earmuffs earmuffs;
43 public FirstAid firstAid;
44 public BuildWall donaldTrump;
45 public BuildWall2 donaldTrump2;
46 public BuildWall3 donaldTrump3;
47 public BuildWall4 donaldTrump4;
48 public BuildWall5 donaldTrump5;
49 public BuildWall6 donaldTrump6;
50 public BuildWall7 donaldTrump7;
51 public BuildWall8 donaldTrump8;
52 public BuildWall9 donaldTrump9;
53 public BuildWall10 donaldTrump10;
54 public BuildWall11 donaldTrump11;
55 public BuildWall12 donaldTrump12;
56 public PickUpBlock block;
57 public PickUpBoxRoof blockRoof;
58 public BuildWallWest21 build1;
59 public BuildWallWest22 build12;
60 public BuildWallWest23 build13;
61 public BuildWallWest24 build14;
62 public BuildWallWest6 build15;
63 public BuildWallWest27 build16;
64 public BuildWall28 build17;
65 public BuildWallWest29 build18;
66 public BuildWallWest210 build19;
67 public BuildWallWest211 build10;
68 public BuildWallWest213 build11;
69 public BuildWallWest214 build12;
70 public BuildWallWest15 build13;
71 public BuildWallWest216 build14;
72 public BuildWallWest217 build15;
73 public BuildWallWest218 build16;
74 public BuildWallWest219 build17;
75 public BuildWallEast21 build21;
76 public BuildWallEast22 build22;
77 public BuildWallEast3 build23;
78 public BuildWallEast24 build24;
79 public BuildWallEast26 build25;
80 public BuildWallEast27 build26;
81 public BuildWallEast28 build27;
82 public BuildWallEast29 build28;
83 public BuildWallEast210 build29;
84 public BuildWallEast211 build210;
85 public BuildWallEast213 build211;
public override void OnStartLocalPlayer() /*Here we use the function that will enable all the scripts and components declared before when we are the local player*/
{
    UI.gameObject.SetActive(true); /*Enable he UI for only my player, this way I will see only my points, my life, etc*/
    fpsController.enabled = true; //REALLY IMPORTANT. Enable the controls only for my player.
    fpsCamera.enabled = true; //REALLY IMPORTANT. Enable the camera only for my player (otherwise we would see through the "eyes" of other players who are not us.*/
    audioListener.enabled = true; //REALLY IMPORTANT. This way we will listen only through our "ears".
    liniaDeVida.enabled = true;
    helmetScript.enabled = true;
    fallDamage2.enabled = true;
    player_net.enabled = true;
    earmuffs.enabled = true;
    firstAid.enabled = true;
    donaldTrump.enabled = true; //All this trump stuff is because these scripts build walls. Humor programming at 4am I guess...
    donaldTrump2.enabled = true;
    donaldTrump3.enabled = true;
    donaldTrump4.enabled = true;
    donaldTrump5.enabled = true;
donaldTrump6.enabled = true;
donaldTrump7.enabled = true;
donaldTrump8.enabled = true;
donaldTrump9.enabled = true;
donaldTrump10.enabled = true;
donaldTrump11.enabled = true;
donaldTrump12.enabled = true;
block.enabled = true;
blockRoof.enabled = true;
bUILD1.enabled = true;
bUILD12.enabled = true;
bUILD13.enabled = true;
bUILD14.enabled = true;
bUILD15.enabled = true;
bUILD16.enabled = true;
bUILD17.enabled = true;
bUILD18.enabled = true;
bUILD19.enabled = true;
bUILD10.enabled = true;
bUILD11.enabled = true;
bUILD112.enabled = true;
bUILD113.enabled = true;
bUILD114.enabled = true;
bUILD115.enabled = true;
bUILD116.enabled = true;
bUILD117.enabled = true;
bUILD221.enabled = true;
bUILD222.enabled = true;
bUILD223.enabled = true;
bUILD224.enabled = true;
bUILD225.enabled = true;
bUILD226.enabled = true;
bUILD227.enabled = true;
bUILD228.enabled = true;
bUILD229.enabled = true;
bUILD2210.enabled = true;
bUILD2211.enabled = true;
bUILD2212.enabled = true;
bUILD2213.enabled = true;
bUILD2214.enabled = true;
bUILD2215.enabled = true;
bUILD2216.enabled = true;
bUILD2217.enabled = true;
bUIDSOUTH1.enabled = true;
bUIDSOUTH2.enabled = true;
bUIDSOUTH3.enabled = true;
bUIDSOUTH4.enabled = true;
bUIDSOUTH5.enabled = true;
bUIDSOUTH6.enabled = true;
bUIDSOUTH7.enabled = true;
bUIDSOUTH8.enabled = true;
bUIDSOUTH9.enabled = true;
bUIDSOUTH10.enabled = true;
buildsouth11.enabled = true;
buildsouth12.enabled = true;
pickUpArnes.enabled = true;
textWall.enabled = true;
textBlocks.enabled = true;
endInfo.enabled = true;

gameObject.name = "LOCAL Player"; //Change the name of the Local Player in the inspector to "LOCAL Player" for easier recognition. //Not really necessary but useful.
base.OnStartLocalPlayer(); //End the function here.

void Start()
{
}

}
/*THIS SCRIPT WILL CONTROL MOST OF OUR ANIMATIONS, INCLUDING THE DEATH ANIMATION AND SOME CAMERA EFFECT FOR IT.*/

public class Player_Net : NetworkBehaviour
{
    NetworkAnimator anim; //Declare a variable "anim" that will be of the class "NetworkAnimator".
    //With this we'll control our animations! (some of them).
    GameObject background; //Declare several GameObject type variables. They'll work as UI's.
    GameObjec helmetOn;
    GameObject arnesOn;
    GameObject arnesOff;
    public GameObject player; //Public GameObject that we'll have to drag in the inspector. It'll be the player prefab.
    public CharacterController controller; //Declare our CharacterController -gives the player a collider, velocity, etc.-.
    AudioSource audio_; //Variable audio_ that'll be an AudioSource.
    public UnityStandardAssets.Characters.FirstPerson.FirstPersonController fpsController; //Again the fpsController...
    public Camera fpsCamera; //And our Camera. We'll use it to make a cool effect when we die.
    [SyncVar] bool dead; /*Boolean variables are "true or false" variables. So... it'll be dead=true or dead=false.
    The [syncVar] thing is to sync certain variables from the server to the clients. Obviously this script will be run on a client so that doesn't make much sense here... I couldn't have time to get more into syncvars, sadly.
    Note that although we don't specify it, since it is not public it won't be seen in the inspector.*/
    float rotation; /*Float variables are basically numbers with decimals. Remember that every number with decimal needs to be ended with an "f", just like 1.5f as an example.*/
    float temps;

    // Use this for initialization. Start function runs when the game starts and only then.
    void Start()
    {
    }
anim = GetComponent<NetworkAnimator>(); /*Define where does this
anim variable comes from.
The GetComponent helps us find components in gameObjects. In this
case anim will be the NetworkAnimator
component in the gameobject this script is attached to. Similar
with the next ones, the "player." here is redundant sometimes*/
player.GetComponent<Health2>();
controller = player.GetComponent<CharacterController>();
audio_ = player.GetComponent<AudioSource>();
fpsCamera.GetComponent<Transform>();
background = GameObject.FindGameObjectWithTag("Background"); /*A good
way to find GameObjects in the Network is with
GameObject.FindGameObjectWithTag("[insertTag]"). It is a "slow" way
to define objects, but certainly comfortable.
Remember to put the tag in the gameobject too.*/
helmetOn = GameObject.FindGameObjectWithTag("CascOn");
arneOn = GameObject.FindGameObjectWithTag("Text");
arneOff = GameObject.getTaggeed("Text1");
}

// Update is called once per frame.
void Update()
{
if (Input.GetKey(KeyCode.LeftShift)) //In case we press the left Shift
{
if (Input.GetKey(KeyCode.W)) //If we also press the W (in the First Person Controller that is making us run forward)
{
anim.animator.SetBool("isWalking", false); /*Get the anim variable, access its animator and set a Boolean value.
In the animator we will have previously created those tags as booleans and made the connections between animations.*/
anim.animator.SetBool("isRunning", true);
anim.animator.SetBool("isIdle", false);
anim.animator.SetBool("isWBackwards", false);
anim.animator.SetBool("isRBackwards", false);
}

//Same for S.
else if (Input.GetKey(KeyCode.S))
{
anim.animator.SetBool("isWalking", false);
anim.animator.SetBool("isRunning", false);
anim.animator.SetBool("isIdle", false);
anim.animator.SetBool("isWBackwards", false);
anim.animator.SetBool("isRBackwards", true);
} //If we're not pressing W nor S, the animation played will be the Idle animation.
else {
    anim.animation.SetBool("isWalking", false);
    anim.animation.SetBool("isRunning", false);
    anim.animation.SetBool("isIdle", true);
    anim.animation.SetBool("isWBackwards", false);
    anim.animation.SetBool("isRBackwards", false);
}

} //Same for S.
else if (Input.GetKey(KeyCode.S)) {
    anim.animation.SetBool("isWalking", false);
    anim.animation.SetBool("isRunning", false);
    anim.animation.SetBool("isIdle", false);
    anim.animation.SetBool("isWBackwards", true);
    anim.animation.SetBool("isRBackwards", false);
}

//If we don't press any key, switch the state to the Idle animation.
else {
    anim.animation.SetBool("isWalking", false);
    anim.animation.SetBool("isRunning", false);
    anim.animation.SetBool("isIdle", true);
    anim.animation.SetBool("isWBackwards", false);
    anim.animation.SetBool("isRBackwards", false);
}

if (dead) //The dead boolean will be set true when the function
...A CIVIL\UNITY\Player Multi\Assets\Scripts\Player Net.cs 4

below -see Die()- sets it true;

rotation += 5 * Time.deltaTime; /*+= adds to the variable rotation 5*Time.deltaTime every frame that "dead" is true, so rotation gets bigger and bigger. Time.deltaTime is the time that passes from frame to frame updates.*/

if (rotation >= 360f)
    rotation -= 360f; /*if rotation gets equal or bigger than 360 degrees, subtract 360 from it. See the difference between doing that and setting rotation to 0, because if you set the rotation to 0 when it eas 360.45, it will cause the camera to snap, instead of following the swift rotation!*/

temps += Time.deltaTime; /*Add Time.deltaTime to our variable temps.*/

//Rotate the camera above the dead player.
fpsCamera.transform.position = player.transform.position + new Vector3(0.5f, 2 + 0.1f*temps, -0.5f);
/*Take the fpsCamera gameObject that is our Camera, then its transform and inside its transform get its position. Make the camera's position equal to the players position + a certain vector to center it in aproximately the middle of its dead lying body and that slowly goes up*/
fpsCamera.transform.rotation = Quaternion.Euler(90f, rotation, 0f); /*Rotate the camera downwards (x=90) and make it spin around the y axis with the increasing rotation degree value.*/

public void Die() /*Function die. Another REALLY IMPORTANT thing about public variables and functions is that they can be accessed from other scripts! Specifically this one will be accessed from the Health2 script.*/
{
    dead = true; //Set the dead bool to true.
    anim.SetTrigger("Died"); /*The death animation is an animation that is played once something happens -press a key, get out of life, whatever-. That means it is a trigger, and the proper way to tell the animator to run triggers is anim.SetTrigger("[insertTag]")./n
    audio_.enabled = false; //We cannot make sounds with our player. It is, indeed, dead.
    fpsController.enabled = false; //Also we can't move obviously.

    //Disable any of these UI that may have been active at that moment.
    background.GetComponent<Image>().enabled = false;
    helmetOn.GetComponent<Text>().enabled = false;
    arnesOn.GetComponent<Text>().enabled = false;
arnesOff.GetComponent<Text>().enabled = false;
CmdDead();/*REALLY IMPORTANT!!! So, the way information goes form client to client is the following. The player does something, and then it sends a Command to the Server saying "Hey Server! I've done something!". Then if the command asks for it, the server sends whatever update that the command asks for to the rest of the clients. Example: I want to click on a helmet and make it disappear. If I script it but I dont make a command, I will see the helmet disappear, because the script will have been run locally, but all the other clients won't see the helmet disappear, because the server won't have told them to make it disappear!

All commands have to start to the prefix Cmd, as all the instructions from Server to Client (ClientRpc's) have to start with Rpc.

So basically writing CmdDead() we said: "Hey Server! I did the command!"*/

void CmdDead() //And what is the thing the command wants to do?
{
    RpcDead(); //It wants the server to run the ClientRpc function (RpcDead) to all the clients!
}

void RpcDead() //And what is the RpcDead doing? Well...
{
    gameObject.tag = "Dead"; /*The gameObject that died changes his tag to "Dead". Remember to set it before. All the copies of that certain object will be updated on all the clients. Wohoo!*/
    controller.height = 0.2f;
    controller.radius = 0.15f;
    controller.center = new Vector3(0.2f, 0.14f, -0.9f);
    controller.transform.position = player.transform.position + new Vector3(0.5f, 0, -0.5f);
    //Also change a bunch of properties of the controller collider of that gameObject such as height, radius, center and position.
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.UI;

/*THIS SCRIPT WILL ALLOW OUR PLAYER TO CONNECT TO THE LIFE LINE. THE SCRIPT INVOLVES IMPORTANT INPUTS ON RAYCAST AND COMMANDS/RPC's.*/

/*DISCLAIMER: In this script there is a bit of redundancy due to last minute changes. What this script does is to let you connect to the life line if you have equiped the harness previously. The harness is equiped through the script "PickUpArnes", so when here we declare the boolean variable "arnes" it should really say "lifeLine". Sorry bro.*/

public class LiniaDeVida : NetworkBehaviour {

    public Transform cameraTransform; /*We'll set a Transform variable called cameraTransform, and it'll be public, so we'll have to set the gameObject whose transform we're talking about manually in the inspector. SPOILER ALERT: It will be our camera's transform. Yeah, the one that is only enabled for us. It will shoot a ray forward at every frame. If this ray collides with something we will gain a really useful information that will allow us to interact with it!*/

    GameObject arnesStatus; //Some private gameObjects that will act as UIs.
    GameObject arnesStatus1;
    GameObject myText2;

    GameObject[] Wall; //An ARRAY of GameObjects tha will be called Wall. To declare an array we have to write "[]" after the class.
    BoxCollider[] boxColliders; //Same for out BoxCollider class.

    bool arnes;
    float waitTime = 3f;
    float timer;

    // Use this for initialization
    void Start () {

        arnesStatus = GameObject.FindGameObjectWithTag("Text"); //What is arnes? It is the GameObject with the tag "Text".
        arnesStatus1 = GameObject.FindGameObjectWithTag("Text1"); //You get the point right?
        myText2 = GameObject.FindGameObjectWithTag("Text2"); //Same here.

    }

    // Update is called once per frame
void Update () {
  RaycastHit hit; /*REALLY IMPORTANT! We declare a variable "hit" of the RaycastHit class. This is the ray I was talking about before.*/
  Vector3 rayPos = cameraTransform.position + cameraTransform.forward * 1f; /*Vector 3 class variable called rayPos that will tell us from where the ray starts (the camera position + 1 unit forward).* /
  Debug.DrawRay(rayPos, cameraTransform.forward * 5f, Color.green, 0.5f); /*This I used for debugging purposes, like seeing where the ray goes, what hits, how far... I make it start from rayPos, go forward from the camera position 5 units, paint it green and make it last 0.5 seconds. Remember this is NOT the real ray, just a representation of what it may be.* /
  if (Physics.Raycast(rayPos, cameraTransform.forward, out hit, 5f)) {
    if (hit.transform.tag == "LiniaDeVida") /*If the raycast hits the transform of a gameobject (it really hits its collider, but the info that the hit variable stores is its transform), and its tag is "LiniaDeVida"...*/{
      PickUpArnes script = gameObject.GetComponent<PickUpArnes>(); /*...MAGIC! We declare a class that is PickUpArnes, under the variable script. And what is that script of the class PickUpArnes? Well its the component called PickUpArnes of the gameobject that this script is attached to. It could be another gameobject, but since we haven't specified it by default is the gameobject this script is attached to. And then we can access all its public variables and functions, and more specifically...*/
      if (script.hasArnes) /*If the variable "hasArnes" of script (remember that it is just the name for PickUpArnes script)...{
        if (!arnes) /*...if the boolean variable "arnes" that we had declared before is false (so if we're not connected to the life line)...{
          myText2.GetComponent<Text>().enabled = true; //Go to myText gameobject, get its component Text and set it true. //In the game this text is giving us the option to pick up the harness.}
if (Input.GetMouseButtonDown(0)) //If we press the left mouse button...
{
    arnesStatus.GetComponent<Text>().enabled = true; // Again, set some text active that says that we have the harness and stuff.
    arnesStatus1.GetComponent<Text>().enabled = false;
    myText2.GetComponent<Text>().enabled = false;
    CmdWall(); //Tell the Server "Hey Server! I did something!"
    arnes = true; //Set our boolean arnes to true, so we won't be able to connect ourselves to the life line once we're already connected
}
else
{
    myText2.GetComponent<Text>().enabled = false;
    //If our ray doesn't hit anything tagged as "LiniaDeVida" then don't show the text option to connect to it.
}

if (Input.GetMouseButtonDown(1) && arnes) /*If we're connected to the life line and we press the right mouse button then...*/
{
    arnes = false; /*Our boolean value says that we're not connected to the life line anymore. Obviously the fact of changing a boolean value doesn't make us connect or disconnect, this is made with the command set before.*/
    arnesStatus.GetComponent<Text>().enabled = false; //Text saying we're connected is disabled.
    arnesStatus1.GetComponent<Text>().enabled = true; //Text saying we're disconnected it enabled.
    timer = 0; //Set our float variable timer to 0. We'll use it to make the arnesStatus1 text disappear after a few seconds.
    CmdOffWall(); //More "Notice me senpai" server stuff...
}

if (!arnes) //If indeed we're not connected to the life line anymore...
{
    timer += Time.deltaTime; //Add Time.deltaTime to the timer variable at every frame.
    if (timer > waitTime) //If timer is larger than waitTime, that we set at the beginning of the script as 3 seconds.
    {
        arnesStatus1.GetComponent<Text>().enabled = false; //Make disappear the text saying that we're not connected
    }
anymore.

```csharp
void CmdWall() //What does this command want the Server to do?
{
    RpcCollide(); //Run the RpcCollide.
}

void CmdOffWall()
{
    RpcOffCollide();
}

[ClientRpc]
void RpcCollide() //What does the RpcCollide do?
{
    if (isLocalPlayer) /*REALLY IMPORTANT! The Server sends the
                     RpcCollide to all the clients and says "Hey Clients, the player in
                     control here is the player who sent the command? If it is
                     then..."*/
    {
        Wall = GameObject.FindGameObjectsWithTag("Wall"); /*Our array of
        gameObjects "Wall" that we declared at the begging will be
        all the gameobjects tagged as wall. Note how there is an "s" at
        FindGameObjectSWithTag.*/
        boxColliders = new BoxCollider[Wall.Length]; /*The BoxCollider
        array called as boxColliders that we declared before will
        be an array of BoxColliders with the same lenght as the array of
        walls.*/
        for(int i=0; i < Wall.Length; i++) /*So for an integer i=0, *
        while i < to the lenght of the Wall array, run the "for"
        function and
        add +1 to "i"*/
        {
            boxColliders[i] = Wall[i].GetComponent<BoxCollider>(); /*The
            boxCollider in the position "i" of the array will be
            equal
to the BoxCollider component attatched to the Wall in the
            position [i] of its array.*/
            boxColliders[i].enabled = true; //Enable that collider, that
            originally was disabled.
            /*What this Rpc does is to enable the colliders of some
            invisible walls put at the edges of the building, so when
            you connect to the life line you can't fall. The fact that
            we use the "if(isLocalPlayer) condition is to make the
            box colliders
```
of the wall active ONLY in the player thats connected to the life line network copy of the walls. So basically he won't be able to fall, whereas all the copies of the walls' box colliders won't be active for the other players and they'll be able to fall, if they're not connected to the life line. Pretty cool, uh? ;)*/

```csharp
[ClientRpc]
void RpcOffCollide() { //Reverse process of the RpcCollide for when you disconnect from the life line.
    if (isLocalPlayer)
        { Wall = GameObject.FindGameObjectsWithTag("Wall");
            boxColliders = new BoxCollider[Wall.Length];
            for (int i = 0; i < Wall.Length; i++)
                { boxColliders[i] = Wall[i].GetComponent<BoxCollider>();
                    boxColliders[i].enabled = false;
                }
        }
}
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.UI;

/*THIS SCRIPT WILL ALLOW US TO PICK UP A HELMET AND PUT IT ON OUR HEAD FOR OTHER PLAYERS TO SEE. ALSO SETS THE TRIGGER FOR THE CPR ANIMATION. AGAIN WE'LL WORK WITH RAYCASTS AND COMMANDS/RPC's.*/

public class Helmet : NetworkBehaviour {

    public Transform cameraTransform; //Again the public transform of our camera.

    GameObject helmetFloor; //Declare several gameObjects that we'll define later.
    GameObject cascOn;
    GameObject myText;
    GameObject myText2;

    public GameObject cPRText; //Declare several gameObjects that will be set in the inspector.
    public GameObject helmetHead;

    public bool hasHelmet; //Public bool variable hasHelmet.

    NetworkAnimator anim; //Again we'll be working with the Network Animator class that we'll call anim.

    // Use this for initialization
    void Start () {
        cascOn = GameObject.FindGameObjectWithTag("CascOn"); //Define some of the GameObjects through tags.
        myText = GameObject.FindGameObjectWithTag("Text3");
        myText2 = GameObject.FindGameObjectWithTag("Text2");

        anim = GetComponent<NetworkAnimator>(); //Define the anim variable.
    }

    // Update is called once per frame
    void Update () {
        /*Again a raycast with a similar pattern as the life line script.*/
        RaycastHit hit;
        Vector3 rayPos = cameraTransform.position + cameraTransform.forward * 1f;
        Debug.DrawRay(rayPos, cameraTransform.forward * 5f, Color.green, 0.5f);

        if (Physics.Raycast(rayPos, cameraTransform.forward, out hit, 2f)) {
        
        }
    }
}
if (hit.transform.tag == "Helmet")
{
  if (!hasHelmet)
  {
    myText.GetComponent<Text>().enabled = true;
  }
  
  helmetFloor = hit.transform.gameObject; /*Remember how we didn't define our gameObject variable helmetFloor?
This is because we can make this variable be applied to different gameObjects, as long as they are tagged as "Helmet",
by saying: "What will helmetFloor be? It will be the gameObject whose transform has been hit by the raycast"./
  if (Input.GetMouseButtonDown(0))
  {
    myText.GetComponent<Text>().enabled = false;
    cascOn.GetComponent<Text>().enabled = true;
    CmdHelmet(helmetFloor, hasHelmet); //"Hey Server! Make this stuff... Careful that now I give you some inputs!"
    if (!hasHelmet)
    {
      PointSystem point = gameObject.GetComponent<PointSystem>(); /*Access the PointSystem script that we'll call point,
located in this gameObject as a component named PointSystem*/
      point.Points(10); /*Inside the point script there is a public function called Points, that accepts integer numbers.
      Sum 10 to that variable*/
    }
    hasHelmet = true; //We have a helmet already, so next time we hit a helmet with our raycast we won't be able to pick it up.
  }
else if (hit.transform.tag == "Dead") //If the raycast doesn't hit a Helmet but hits a player that died (therefore tagged as dead):
{
    Debug.Log("PrepareCPR"); //In the console window say "PrepareCpr". Again not important, just for debugging reasons.
    cPRText.GetComponent<Text>().enabled = true; //Enable a text allowing to make the CPR.
    if (Input.GetMouseButtonDown(0))
    {
      EndingInfo script = gameObject.GetComponent<EndingInfo> (); //Access the EndingInfo script attached to this
```csharp
gameObject.
  script.EndGame(); // Run the EndingInfo script EndGame() function.
  anim.SetTrigger("CPR"); // Set the trigger for the animator to make the CPR animation.
}
else {
  myText.GetComponent<Text>().enabled = false;
  CPRText.GetComponent<Text>().enabled = false;
}
}
}

[Command]
void CmdHelmet(GameObject helmetFloor, bool hasHelmet) /* Okay so in this Command we send some info to the Server, more specifically which gameObject is the helmet that we have hit with our raycast and if we have a helmet already on or no. */ {
  RpcHelmet(helmetFloor, hasHelmet); // The Rpc function will use those parameters.
}

[ClientRpc]
void RpcHelmet(GameObject helmetFloor, bool hasHelmet) /* When we declare the inputs that can go inside a function, we have to declare which class they belong to. */ {
  if (!hasHelmet) // If the player that sent the command doesn't have a helmet:
    {
      helmetFloor.SetActive(false); // Make the helmet that we hit with the floor disappear (SetActive to false).
      helmetHead.GetComponent<Renderer>().enabled = true; // Set the rendering of the "invisible" helmet that the player wore on his head true;
    }
/* Note the huge difference between this Rpc and the one in the life line script. Here what the server does is to say to all the clients: "Hey Clients. If the your copy of the player that sent the command doesn't have a helmet, make the renderers of his helmet active so everyone can see how safe and fabulous he looks in it!" */

So in this script all copies of the object that we are being
```
...YERIA CIVIL\UNITY\Player_Multi\Assets\Scripts\Helmet.cs

        refered to are affected by the Rcp, whereas in the Life Line script
        the only copies of the walls that got its colliders activated where the
        ones of the player that sent the Command, thanks to the
        "(isLocalPlayer)" condition.*/
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Networking;

/*THIS SCRIPT WILL CONTROL THE HEALTH OF OUR PLAYER.*/

public class Health2 : NetworkBehaviour {

    public const int maxHealth = 100; /*Maximum health at the start.*/
    public float currentHealth = maxHealth; //At the start the current health is equal to the max health.
    public bool dead;
    Player_Net player; //Referencing the Player_Net script and calling it "player".
    public RectTransform healthbar2; //RectTransform is a class that will allow us to easily change the size of our health bar.
    //It is a public RectTransform that we will have to drag into the inspector.

    void Start()
    {
        player = gameObject.GetComponent<Player_Net>(); /*We tell the script to find the Player_Net script named as "player" that we had declared before. Where will the Health2 script find it? In this gameObject, in the component called Player_Net.*/
    }

    public void TakeDamage(float amount) /*We create a function called take damage. Note certain things:
    - It is public. This script will run all the health updates, so it will be accessed by all the scripts that control the different damages.
    - It is not in Update(). This function will run only when other scripts tell it to run.
    - It has an input, that will be a float number, and that we'll call "amount" to work with it in the function.*/
    {
        currentHealth -= amount; //Everytime this function runs, substract the number corresponding to amount from currentHealth;
        int roundAmount = Mathf.RoundToInt(amount); //Create the integer variable roundAmount and make it be the amount variable rounded.
        PointSystem script = gameObject.GetComponent<PointSystem>(); //Access the PointSystem script attatched to this same gameObject.
        script.Points(-1*roundAmount); //IMPORTANT. From the PointSystem script, access its function Points and pass -1*roundAmount.
        //We just accessed a function from another script and passed a value to it!

        if (currentHealth <= 0) //If the variable currentHealth goes below 0...
        {
            //Do something.
        }
    }
}
currentHealth = 0; //Make it 0. We don't want negative health values, do we?
dead = true; //Set the boolean value "dead" to true;
Debug.Log("Dead");
player.Die(); //Run the function "Die()" located in the Player_Net script!
EndingInfo script2 = gameObject.GetComponent<EndingInfo>(); /*Also declare another script "EndingInfo" and run its function EndGame().*/
script2.EndGame();

if (currentHealth > 100) //If health is larger than 100...
{
currentHealth = 100; //Make it 100. This could happen for example if we lost some health and we picked up the FirstAidKit.
}

healthbar2.sizeDelta = new Vector2(currentHealth * 2, healthbar2.sizeDelta.y); /*Access the RectTransform healthBar, its property sizeDelta and make its x value equal to currentHealth*2 (that's only because the healthBar has a length of 200 in the inspector.
Its y component remains the same.*/

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

// THIS SCRIPT WILL CONTROL THE DAMAGE TAKEN AFTER FALLS TO DIFFERENT LEVEL. NOTE THAT IT IS MONOBEHAVIOUR
// SINCE IT DOESN'T HAVE ANYTHING TO DO WITH NETWORKING.

public class FallDamage2 : MonoBehaviour {

    public float startYPos;
    public float endYPos;
    public float damageThreshold = 2f;
    public bool damageMe = false;
    public bool firstCall = true;

    public CharacterController player; //Declare the CharacterController property. It is public because we'll drag it in the inspector.
    //It is really important because the CC has the variable "isGrounded" that returns if the player is grounded or not.

    public AudioClip[] clips; //Define a public array of audio clips called... clips. Not very original, I know.

    // Update is called once per frame
    void Update () {
        if (!player.isGrounded) //If the CC (that we called "player") says that the player is not grounded...
        {
            if (gameObject.transform.position.y > startYPos) //If the y position of our player is higher than startYPos...
            {
                firstCall = true;
            }
            if (firstCall)
            {
               startYPos = gameObject.transform.position.y; //Make the startYPos equal to our player y position.
               firstCall = false; //Then we won't be able to update this startYPos again until our position is higher than startYPos.
               damageMe = true; //Since we're not grounded, we can take damage!
            }
        }

        if (player.isGrounded) //So if our player is hitting the ground...
        {
            endYPos = gameObject.transform.position.y; //endYPos is equal to our gameObject position.
            if (startYPos - endYPos > damageThreshold) //If the difference between our start positon and our end position is higher than our damage threshold (distance of fall that won't make us
take damage)
        
        } //If damage me is true...
        
        {
            Health2 health = gameObject.GetComponent<Health2>(); // Access the Health2 script attached to this same game object.
            if (health != null) //If the script is enabled (there's different ways to say this, this is one of them...)
            {
                PlaySound(); //Run the function PlaySound() stated below.
                health.TakeDamage(20 * (startYPos - endYPos - damageThershold)); //Send a value to the TakeDamage function in the Health2
                //script, equal to 20 times the difference between the startYPos, and endYPos and damageThreshold.
            } //DamageMe is false, we cannot take more damage in this frame.
            damageMe = false;
            firstCall = true; //We can update again our startYPos.
        }
    }

    private void OnControllerColliderHit(ControllerColliderHit hit) /*This function was written because I was having issues going down stairs. Since the player collider was bigger than the width of the steps, the CC didn't count it as isGrounded, because the bottom of the collider wasn't colliding with any horizontal surface, and therefore when I got to the ground, the script thought that I had jumped, and I took a damage that had no sense and it was stupid and frustrating. The OnControllerColliderHit function checks if your CC hits something every frame. In this case I told it that if it hit a collider that in this case was tagged as "Stair", it set the startYPos to 0. This way when the player was going down the stairs its startYPos was always 0, and as soon as it hit the ground or a floor, the startYPos got updated to its real value again. This could be a problem in case we fell from a stair to a huge gap, but in our experience this is not a problem.*/
    {
        if (hit.collider.tag == "Stair")
        {
            startYPos = 0;
        }
    }

    void PlaySound() //We'll play some cool clips to make sure that the person playing knows he has taken damage.
```csharp
{ int randomClip = Random.Range(0, clips.Length); //randomClip will be an integer number chosen randomly from 0 the the length of the clips array.

AudioSource source = gameObject.AddComponent<AudioSource>(); //We define an audio source called "source" that will be Added to our gameObject.

source.clip = clips[randomClip]; //The clip played by the source will be defined as the clip that corresponds to the index "randomClip" of the "clips" array.

source.Play(); //Play the source!

Destroy(source, clips[randomClip].length); //Destroy the source (because after all we don't want a thousand sources in our inspector)

//after a certain amount of time. What time you say? The amount of time will be the length of the clip equal to the index randomClip of the clips Array.
}
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.UI;

/*THIS SCRIPT WILL CONTROL THE PICK UP OF THE EARMUFFS AND IT WORKS SIMILARLY TO THE HELMET SCRIPT.
THERE IS AN EXTRA FEATURE THAT IS THE TURN DOWN OF THE VOLUME WHEN WE PUT THE EARMUFFS ON.*/

public class Earmuffs : NetworkBehaviour {
    public Transform cameraTransform; //Public camera transform for the raycast.
    GameObject earmuffsFloor; //Some GameObjects that will act as text.
    GameObject earmuffsText;
    GameObject earmuffsOn;
    public GameObject earmuffsHead; //The invisible earmuffs that we have on the player's head and that we will drag in the inspector.

    bool hasEarmuffs; //Boolean value "hasEarmuffs".

    // Use this for initialization
    void Start () {
        earmuffsText = GameObject.FindGameObjectsWithTag("EarmuffsText"); // Define the GameObjects.
        earmuffsOn = GameObject.FindGameObjectsWithTag("EarmuffsOn");
    }

    // Update is called once per frame
    void Update () {
        RaycastHit hit;
        Vector3 rayPos = cameraTransform.position + cameraTransform.forward * 1f;
        Debug.DrawRay(rayPos, cameraTransform.forward * 5f, Color.green, 0.5f);

        if (Physics.Raycast(rayPos, cameraTransform.forward, out hit, 5f)) // Shoot the ray.
        {
            if (hit.transform.tag == "EarmuffsFloor") //If collides with the gameObject tagged as "EarmuffsFloor"
            {
                if (!hasEarmuffs) //If we don't have the earmuffs on yet...
                {
                    earmuffsText.GetComponent<Text>().enabled = true; //Show the text option to pick up the earmuffs.
                }
            }
            earmuffsFloor = hit.transform.gameObject; //Make the earmuffsFloor gameObject be the object which the ray has hit.
        }
    }
}
if (Input.GetMouseButtonDown(0)) //If we press the left mouse button...
{
    earmuffsText.GetComponent<Text>().enabled = false; //Disable the text option to pick them up.
    earmuffsOn.GetComponent<Text>().enabled = true; //Show the text that says "Earmuffs equipped".
    CmdEarmuffs(earmuffsFloor, hasEarmuffs); //Command to make the server update the new state of the earmuffs to everyone.
    if (!hasEarmuffs) //If we don't have earmuffs... (we actually have them, but the script doesn't know because we haven't told it).
    {
        PointSystem point = gameObject.GetComponent<PointSystem>();
        point.Points(10); //Access the PointSystem script and pass a value of 10 to its function "Points()".
    }
    hasEarmuffs = true; //NOW the script knows we have the earmuffs!
}
else //If our raycast is not hitting an object tagged as "EarmuffsFloor" don't show the option to pick the earmuffs up.
{
    earmuffsText.GetComponent<Text>().enabled = false;
}

[Command]
void CmdEarmuffs(GameObject earmuffsFloor, bool hasEarmuffs) //Hey server! Take the GameObject earmuffsFloor and the bool hasEarmuffs and do an Rpc!
{
    RpcEarmuffs(earmuffsFloor, hasEarmuffs); //Okay Client, and what should I do?
}

[ClientRpc]
void RpcEarmuffs(GameObject earmuffsFloor, bool hasEarmuffs) //Take those inputs and...
{
    if (!hasEarmuffs) //If hasEarmuffs is false...
    {
        earmuffsFloor.SetActive(false); //Make the earmuffsFloor gameObject (the one that the raycast hit) invisible.
        earmuffsHead.GetComponent<Renderer>().enabled = true; //Make the invisible earmuffs in our player's head visible.
    }
if (isLocalPlayer)
{
    AudioListener.volume = 0.1f; //If the Command was sent by the LocalPlayer, turn the volume of his Audio Listener down to 0.1.
    //Those look like a hell of an earmuffs don't they?
}
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Audio;

//THIS SCRIPT WILL CONTROL THE COLLISION OF OUR PLAYER WITH THE EXCAVATOR, AND PLAY SOME AUDIO OF COMPLAINT.

public class CollisionExcavator : MonoBehaviour {

    public AudioClip clip1; //Public AudioClip called "clip1" that we'll set in the inspector.
    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    void Update () {
    }

    private void OnControllerColliderHit(ControllerColliderHit collision) {
        //Again, when our Controller Collider hits another collider...*/
        if (collision.gameObject.tag == "Excavadora") //If it hits a
        {
            Debug.Log("collided");
            Health2 health = gameObject.GetComponent<Health2>(); //Access the Health2 script attached to the player.
            AudioSource source = gameObject.AddComponent<AudioSource>(); // Access the audio source attached to the player.
            source.clip = clip1; //The clip that the source will have to play is "clip1"
            source.Play(); //Play the clip!
            Destroy(source, clip1.length); //Destroy the clip once it is finished.
            if (health != null)
            {
                health.TakeDamage(5); //Pass a value of 5 to the TakeDamage function of the health script. Note that this function is run every frame, so if you keep touching the excavator you'll keep passing 5, after 5, after 5, and you'll die pretty quick...
            }
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.UI;

//THIS SCRIPT WILL ALLOW US TO PICK UP THE FIRST AID KIT.

public class FirstAid : NetworkBehaviour {

    public Transform cameraTransform; //Yaaaaay raycasts
    GameObject firstAid; //GameObject that will be our FirstAid box.
    GameObject firstAidText; //Text allowing us to pick it up.
    // Use this for initialization
    void Start () {
        firstAidText = GameObject.FindGameObjectWithTag("FirstAidText"); //Define the firstAidText GameObject declared before.
    }

    // Update is called once per frame
    void Update () {
        RaycastHit hit;
        Vector3 rayPos = cameraTransform.position + cameraTransform.forward * 1f;
        Debug.DrawRay(rayPos, cameraTransform.forward * 5f, Color.green, 0.5f);

        if (Physics.Raycast(rayPos, cameraTransform.forward, out hit, 5f)) {
            if (hit.transform.tag == "FirstAid") {
                firstAid = hit.transform.gameObject; //The firstAid gameObject will be the raycasted object if it is tagged as "FirstAid"
                firstAidText.GetComponent<Text>().enabled = true;

                if (Input.GetMouseButtonDown(0)) {
                    Health2 health = gameObject.GetComponent<Health2>();
                    if (health != null) {
                        firstAidText.GetComponent<Text>().enabled = false;
                        health.TakeDamage(-25); //We pass a negative value, so we're actually recovering life!
                        CmdFirstAid(firstAid); //Command passing the raycasted gameObject.
                    }
                }
            }
        }
        else
    }
```csharp
    {
        firstAidText.GetComponent<Text>().enabled = false;
    }

    
    

    [Command]
    void CmdFirstAid(GameObject firstAid)
    {
        RpcFirstAid(firstAid);
    }

    [ClientRpc]
    void RpcFirstAid(GameObject firstAid)
    {
        firstAid.SetActive(false); // The server tells the clients to set unactive all the copies of the gameObject "firstAid".
    }
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

//THIS SCRIPT WILL CONTROL THE TIME IN THE GAME.

public class Timer : MonoBehaviour {
    public Text gameTimerText; //Again some text.
    public float gameTimer = 901f; //Seconds of game.
    public int seconds;
    public int minutes;

    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    void Update () {
        gameTimer -= Time.deltaTime; //At every frame subtract Time.deltaTime from our gameTimer.
        seconds = (int)(gameTimer % 60); //Aply these operation to transform form gameTimer to seconds and minutes.
        minutes = (int)(gameTimer / 60) % 60;
        string timerString = string.Format("{0:00}:{1:00}", minutes, seconds); //Same process as in the script PointSystem.
        gameTimerText.text = timerString;

        if (gameTimer <= 0) //If the time gets to 0, set the gameTimer to 0 and run the function EndGame() from the EndingInfo script.
        {
            gameTimer = 0;
            EndingInfo script2 = gameObject.GetComponent<EndingInfo>();
            script2.EndGame();
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

//THIS SCRIPT WILL CONTROL THE POINT SYSTEM.

public class PointSystem : MonoBehaviour {

    public Text pointText; //Public text that will show us our points on screen.
    public int points; //Our points.

    // Use this for initialization
    void Start()
    {
    }

    public void Points(int pointsExtra) //Function Points that has an integer argument called pointsExtra. It is not in Update(), so will only run when another script tells it to run.
    {
        points += pointsExtra; //Add to our current points the pointsExtra amount.
        if (points <= 0)
        {
            points = 0; //If points is equal or minor than 0, make it 0;
        }
        string pointString = string.Format("{0}pts", points); /*This line controls how the variable points in the script actually turns in something we see on our screen. We declare the string called "pointString", and we say that it will be a string, in the format of a number {} followed by the "word" pts. The number inside the brackets will be the variable points.*/
        pointText.text = pointString; //Then the text of our GameObject’s text will be, indeed, the pointString string.
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Networking;

// THIS SCRIPT WILL CONTROL THE PICK UP OF THE BLOCKS TO BUILD THE WALLS
// WHEN THEY ARE PILED. ALSO IT WILL CONTROL THE LOAD OF THOSE BLOCKS THAT WE
// PICKED UP ONTO THE PLATFORM OF THE CRANE AND WE'LL SET THE CRANE'S MOVEMENT IN MOTION.

public class PickUpBlock : NetworkBehaviour {

    public Transform cameraTransform; // Camera transform for the raycast.
    public GameObject myBlocks; // The public game object to be dragged in the inspector that will be our blocks.
    public GameObject myTakeText; // Text allowing us to pick up the blocks.
    Renderer[] blockRenderer; // Array of renderers of the blocks that we'll define later.
    GameObject[] blocks; // Array of GameObjects that will correspond to the blocks that we'll define later.

    public bool hasBlocks; // We'll be using a new animation when we pick the blocks, so we declare the NetworkAnimator component.

    // Use this for initialization
    void Start () {
        anim = GetComponent<NetworkAnimator>(); // What is anim? It is the Component Network Animator of the game object this script is attached to.
        myTakeText = GameObject.FindGameobjectWithTag("TakeBlocks"); // We define what game object "myTakeText" is.
    }

    // Update is called once per frame
    void Update () {
        RaycastHit hit;
        Vector3 rayPos = cameraTransform.position + cameraTransform.forward * 1f;
        Debug.DrawRay(rayPos, cameraTransform.forward * 5f, Color.green, 0.5f);

        if (Physics.Raycast(rayPos, cameraTransform.forward, out hit, 5f)) // Raycast.
        {
            if (hit.transform.tag == "Block") // If we hit a collider with the tag Block...
            {
            }
        }
    }
}
        { myTakeText.GetComponent<Text>().enabled = true; //Show the option to pick blocks up. Debug.Log("Can pick up blocks!"); if (Input.GetMouseButtonDown(0)) //If we press the left mouse button...
        {
            PointSystem script = gameObject.GetComponent<PointSystem>(); script.Points(5); //Add 5 points to the Point() function in the PointSystem script attached to this gameObject. hasBlocks = true; anim.animator.SetBool("isMovingBlock", true); //Play the animation related to the "isMovingBlock" bool. //It'll be updated through the Network thanks to the Network animator. CmdBlock(); //"Hey Server! Do something!
        }
        else
        {
            myTakeText.GetComponent<Text>().enabled = false; //If our Raycast doesn't hit anything tagged as "Block", don't enable gameObject myTakeText.
        }
    }

    if (Physics.Raycast(rayPos, cameraTransform.forward, out hit, 5f)) //If the Raycast hits a collider tagged as "Box"
    {
        if (hit.transform.tag == "Box")
        {
            if (hasBlocks) //If we previously had picked up the blocks...
            {
                Debug.Log("Can Load");
                if (Input.GetMouseButtonDown(0)) //If we press the left mouse button...
                {
                    PointSystem script = gameObject.GetComponent<PointSystem>(); script.Points(5); //Add 5 points again. hasBlocks = false; //We don't have the blocks anymore, we put them on the platform. anim.animator.SetBool("isMovingBlock", false); //We stop playing the animation. CmdPlatform(); //"Hey Server! Do something else!
                }
            }
        }
    }
```csharp
[Command]
void CmdBlock()
{
    RpcBlock();
}

[ClientRpc]
void RpcBlock()
{
    myBlocks.SetActive(true); // Set the invisible gameObject (blocks) parented to the player active for all the clients.
}

[Command]
void CmdPlatform()
{
    RpcPlatform();
}

[ClientRpc]
void RpcPlatform()
{
    blocks = GameObject.FindGameObjectsWithTag("Box"); // The array of GameObjects we declared as blocks is the amount of GameObjects tagged as "Box".
    blockRenderer = new Renderer[blocks.Length]; // blockRenderer is an array of Renderers that has the length of the blocks' array.
    for (int i = 0; i < blocks.Length; i++) // For an integer number i=0, if its smaller than the length of the array blocks, do something and add i+1;
    {
        blocks[i].layer = 0; /* Put that block in the layer 0. That layer is the default one. This is because in the scene they were defined in a custom layer that didn't collide with the players - so they were invisible and "uncollidable". Now we make them collidable. */
        blockRenderer[i] = blocks[i].GetComponent<Renderer>(); // Now we define what the elements in the blockRenderer array are exactly.
        blockRenderer[i].enabled = true; // We enable them, so now we see the blocks!
        blocks[i].tag = "BlockRoof"; // We'll tag them as BlockRoof. That's for the next script, when we pick them up in the roof.
    }
    myBlocks.SetActive(false); // Set the blocks parented to my player unactive for all the clients.
    GameObject crane = GameObject.FindGameObjectsWithTag("Crane"); // Declare a gameObject called Crane. It'll be the crane.
    crane_animate1 script = crane.GetComponent<crane_animate1>(); // We
```
get a script attached to the gameObject crane, not ourselves!

// This script will control the upwards movement of the crane. The crane's movement will be done through 2 scripts.

script.isMoving = true; // We activate 2 boolean variables from that script, so it starts running and therefore moving the crane.

script.enabled = true; // We enable the script. This is because both scripts that control the crane enable and disable themselves,

// But for the first movement we have to tell it when to start going upwards!

script.Start(); // This is also to restart the position of the crane. This way the movement between the two stages that the crane has is smooth.

}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

//THIS SCRIPT IS REALLY SIMILAR TO THE PICKUPBLOCKS ONE.
//WE'LL PICK UP THE BLOCKS FROM THE PLATFORM ON THE ROOF.

public class PickUpBoxRoof : NetworkBehaviour {

    public Transform cameraTransform;
    public GameObject myBlocks;

    Renderer[] blockRenderers;
    GameObject[] blocks;

    public bool canBuild;
    public int clicks = 0; //Integer that will control we've run out of blocks to build and we need another load of blocks.

    NetworkAnimator anim;

    // Use this for initialization
    void Start()
    {
        anim = GetComponent<NetworkAnimator>();
    }

    // Update is called once per frame
    void Update()
    {
        RaycastHit hit;
        Vector3 rayPos = cameraTransform.position + cameraTransform.forward * 1f;
        Debug.DrawRay(rayPos, cameraTransform.forward * 5f, Color.green, 0.5f);

        if (Physics.Raycast(rayPos, cameraTransform.forward, out hit, 5f))
        {
            if (hit.transform.tag == "BlockRoof")
            {
                GameObject crane = GameObject.FindGameObjectWithTag("Crane");
                crane_animate1 script = crane.GetComponent<crane_animate1> ();
                if (script.canPickBlocksUp) //In the crane script "crane_animate1" there is a boolean that is set to true when the platform
                    //touches the roof.
                    {
                        if (Input.GetMouseButtonDown(0))
                            
            
        
    
}
PointSystem script2 = gameObject.GetComponent<PointSystem>();
    script2.Points(5);
    canBuild = true; //We have blocks now, so... We can build!
    anim.animator.SetBool("isMovingBlock", true); //Also start again the animation holding the blocks.
    CmdCanBuild();
    }
}

if (clicks > 11 && canBuild) //If we put 12 blocks we run out of them. This click variable will come from the //several BuildWall scripts every time we build a wall.
{
    anim.animator.SetBool("isMovingBlock", false); //Animation holding blocks is over.
    CmdJobDone();
    clicks = 0; //Set clicks back to 0 so next time we have 12 blocks again.
    canBuild = false; //We can't build anymore :(
}

[Command]
void CmdCanBuild()
{
    RpcCanBuild();
}

[Command]
void CmdJobDone()
{
    RpcJobDone();
}

[ClientRpc]
void RpcCanBuild()
{
    myBlocks.SetActive(true); //Set the blocks parented to our gameObject active for all the clients.
    blocks = GameObject.FindGameObjectsWithTag("BlockRoof"); //Similar procedure as with PickUpBlocks.
    blockRenderer = new Renderer[blocks.Length];
    for (int i = 0; i < blocks.Length; i++)
    {
        blocks[i].layer = 12; //Bring the blocks back to the layer where they can’t collide with the player, because they don’t exist
    }
in the platform.

    blockRenderer[i] = blocks[i].GetComponent<Renderer>();
    blockRenderer[i].enabled = false;
    blocks[i].tag = "Box"; //Bring the tag of the blocks back to Box.

    GameObject crane = GameObject.FindGameobjectWithTag("Crane"); //Find the GameObject crane.
    Crane script = crane.GetComponent<Crane>(); //Find the second script in crane that controls its movement downwards.
    script.isMoving = true; //We allow it to start moving.
    script.enabled = true; //We enable it. At this point the first crane script will have disabled itself, so both scripts won't overlap.
    script.Start(); //Restart so the transition of positions between scripts is smooth.

    [ClientRpc]
    void RpcJobDone()
    {
        myBlocks.SetActive(false); //Make the blocks parented to our gameObject unactive for all the clients.
    }
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Audio;

//THIS SCRIPT WILL CONTROL THE COLLISION OF OUR PLAYER WITH THE PLATFORM.

public class CollisionPlatform : MonoBehaviour
{
    public AudioClip clips; //Public audio clip to play when we collide with the platform.

    // Use this for initialization
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
    }

    private void OnControllerColliderHit(ControllerColliderHit collision)
    {
        if (collision.gameObject.tag == "Platform") //If we collide with the Crane (gameObject tagged as "Platform")...
        {
            GameObject crane = GameObject.FindGameObjectWithTag("Crane");
            crane_animate1 script = crane.GetComponent<crane_animate1>();
            if (script.isMoving) //If the variable isMoving from the crane_animate1 script is true (so, if the platform is going upwards)...
            {
                Debug.Log("collided");
                Health2 health = gameObject.GetComponent<Health2>();
                if (health != null)
                {
                    PlaySound(); //Run the function PlaySound() declared below.
                    health.TakeDamage(5); //Take Damage in the healthy script.
                }
            }
            Crane script2 = crane.GetComponent<Crane>();
            if (script2.isMoving) //If the variable isMoving from the Crane script is true (so, if the platform is going downwards)...
            {
                Debug.Log("collided");
                Health2 health = gameObject.GetComponent<Health2>();
                if (health != null)
                {
                }
        }
    }
}
PlaySound(); // Run the function PlaySound() declared below.

health.TakeDamage(5); // Take Damage in the healthy script.

void PlaySound() // Same function PlaySound as some of the described before.
{
    AudioSource source = gameObject.AddComponent<AudioSource>();
    source.clip = clips;
    source.Play();
    Destroy(source, clips.length);
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.UI;

// THIS SCRIPT WILL ALLOW US TO PICK UP THE HARNESS. IT'S EXACTLY THE SAME // PROCESS AS PICKING UP THE HELMET, SO THERE WON'T BE AN EXPLANATION TO IT.

public class PickUpArnes : NetworkBehaviour {
    public Transform cameraTransform;
    GameObject arnesFloor;
    public GameObject arnesBody;
    GameObject myText4;
    GameObject arnesOn;
    GameObject myText2;
    public bool hasArnes;

    // Use this for initialization
    void Start()
    {
        myText4 = GameObject.FindGameObjectsWithTag("Text4");
        arnesOn = GameObject.FindGameObjectsWithTag("ArnesOn");
        myText2 = GameObject.FindGameObjectsWithTag("Text2");
    }

    // Update is called once per frame
    void Update()
    {
        RaycastHit hit;
        Vector3 rayPos = cameraTransform.position + cameraTransform.forward * 1f;
        Debug.DrawRay(rayPos, cameraTransform.forward * 5f, Color.green, 0.5f);

        if (Physics.Raycast(rayPos, cameraTransform.forward, out hit, 5f))
        {
            if (hit.transform.tag == "Arnes")
            {
                if (!hasArnes)
                {
                    myText4.GetComponent<Text>().enabled = true;
                }
                arnesFloor = hit.transform.gameObject;
                if (Input.GetMouseButton(0))
                {
                    myText4.GetComponent<Text>().enabled = false;
                    arnesOn.GetComponent<Text>().enabled = true;
                    CmdArnes(arnesFloor, hasArnes);
                }
            }
        }
    }
}
```csharp
    if (hasArnes)
    {
        PointSystem point = gameObject.GetComponent<PointSystem>();
        point.Points(10);
        hasArnes = true;
    }

    else
    {
        myText4.GetComponent<Text>().enabled = false;
    }

    [Command]
    void CmdArnes(GameObject arnesFloor, bool hasArnes)
    {
        RpcArnes(arnesFloor, hasArnes);
    }

    [ClientRpc]
    void RpcArnes(GameObject arnesFloor, bool hasArnes)
    {
        if (!hasArnes)
        {
            arnesFloor.SetActive(false);
            arnesBody.SetActive(true);
        }
    }
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Networking;

// THIS SCRIPT WILL ALLOW US TO ACTIVATE THE TEXT TO ALLOWS THE PLAYER TO
// BUILD EACH SECTION OF WALL. ALSO STOPS THE

public class ActivateTextWall : NetworkBehaviour
{
    NetworkAnimator anim; // We’ll be working with animations again.
    public int count = 0; // Public integer that we’ll use to count how many blocks have we built.
    public Transform cameraTransform; // Camera transform for the raycast.
    public GameObject blocks; // Our public gameObject blocks that will be the "Invisible" blocks to be built.
    GameObject myBuildText; // Text that will allow us to pick up the object.
    bool canSumPoints1; // Bool value for every block to be built in the scene. It shows if it can sum punts or not.
    bool canSumPoints2;
    bool canSumPoints3;
    bool canSumPoints4;
    bool canSumPoints5;
    bool canSumPoints6;
    bool canSumPoints7;
    bool canSumPoints8;
    bool canSumPoints9;
    bool canSumPoints10;
    bool canSumPoints11;
    bool canSumPoints12;
    bool canSumPoints13;
    bool canSumPoints14;
    bool canSumPoints15;
    bool canSumPoints16;
    bool canSumPoints17;
    bool canSumPoints18;
    bool canSumPoints19;
    bool canSumPoints20;
    bool canSumPoints21;
    bool canSumPoints22;
    bool canSumPoints23;
    bool canSumPoints24;
    bool canSumPoints25;
    bool canSumPoints26;
    bool canSumPoints27;
    bool canSumPoints28;
    bool canSumPoints29;
    bool canSumPoints30;
    bool canSumPoints31;
    bool canSumPoints32;
    bool canSumPoints33;
bool canSumPoints134;
bool canSumPoints135;
bool canSumPoints136;
bool canSumPoints137;
bool canSumPoints138;
bool canSumPoints139;
bool canSumPoints140;
bool canSumPoints141;
bool canSumPoints142;
bool canSumPoints143;
bool canSumPoints144;
bool canSumPoints145;
bool canSumPoints146;
bool canSumPoints147;
bool canSumPoints148;
bool canSumPoints149;
bool canSumPoints150;
bool canSumPoints151;
bool canSumPoints152;
bool canSumPoints153;
bool canSumPoints154;
bool canSumPoints155;
bool canSumPoints156;
bool canSumPoints157;
bool canSumPoints158;

// Use this for initialization
void Start()
{
    canSumPoints1 = true; // Since no block is built yet all of them can sum points.
    canSumPoints2 = true;
    canSumPoints3 = true;
    canSumPoints4 = true;
    canSumPoints5 = true;
    canSumPoints6 = true;
    canSumPoints7 = true;
    canSumPoints8 = true;
    canSumPoints9 = true;
    canSumPoints10 = true;
    canSumPoints11 = true;
    canSumPoints12 = true;
    canSumPoints13 = true;
    canSumPoints14 = true;
    canSumPoints15 = true;
    canSumPoints16 = true;
    canSumPoints17 = true;
    canSumPoints18 = true;
    canSumPoints19 = true;
    canSumPoints20 = true;
    canSumPoints21 = true;
    canSumPoints22 = true;
    canSumPoints23 = true;
canSumPoints124 = true;
canSumPoints125 = true;
canSumPoints126 = true;
canSumPoints127 = true;
canSumPoints128 = true;
canSumPoints129 = true;
canSumPoints130 = true;
canSumPoints131 = true;
canSumPoints132 = true;
canSumPoints133 = true;
canSumPoints134 = true;
canSumPoints135 = true;
canSumPoints136 = true;
canSumPoints137 = true;
canSumPoints138 = true;
canSumPoints139 = true;
canSumPoints140 = true;
canSumPoints141 = true;
canSumPoints142 = true;
canSumPoints143 = true;
canSumPoints144 = true;
canSumPoints145 = true;
canSumPoints146 = true;
canSumPoints147 = true;
canSumPoints148 = true;
canSumPoints149 = true;
canSumPoints150 = true;
canSumPoints151 = true;
canSumPoints152 = true;
canSumPoints153 = true;
canSumPoints154 = true;
canSumPoints155 = true;
canSumPoints156 = true;
canSumPoints157 = true;
canSumPoints158 = true;

myBuildText = GameObject.FindGameWithTag("TextBuild"); //We find our gameObject acting as text.
anim = gameObject.GetComponent<NetworkAnimator>(); //Define anim.

// Update is called once per frame
void Update()
{
    PickUpBoxRoof script2 = gameObject.GetComponent<PickUpBoxRoof>();
    RaycastHit hit;
    Vector3 rayPos = cameraTransform.position + cameraTransform.forward *
    Debug.DrawRay(rayPos, cameraTransform.forward * 5f, Color.green, 0.5f);

    if (Physics.Raycast(rayPos, cameraTransform.forward, out hit, 5f)) //Raycast.
if (script2.canBuild) // If the variable canBuild from the PickUpBoxRoof is true, then...
    if (hit.transform.tag == "WallDef") // If the raycast hit a gameObject tagged as "WallDef"...
        if (canSumPoints1) // If it canSumPoints1 (so, if it's not built yet)...
            myBuildText.GetComponent<Text>().enabled = true; // Show the text allowing us to pick it up.
            if (Input.GetMouseButtonDown(0)) // If we press the left mouse button...
                PointSystem script = gameObject.GetComponent<PointSystem>();
                script.Points(1); // Add 1 to the points in the PointsSystem script function Points().
                canSumPoints1 = false; // We cannot build this concrete block anymore.
                myBuildText.GetComponent<Text>().enabled = false; // Disable our text inviting to build.
                count += 1; // Add one to the count of blocks built.
        }
    }
else if (hit.transform.tag == "WallDef2") // Eat, drink, sleep and repeat...
    if (canSumPoints2)
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints2 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
else if (hit.transform.tag == "WallDef3")
if (canSumPoints3)
{
    myBuildText.GetComponent<Text>().enabled = true;
    if (Input.GetMouseButtonDown(0))
    {
        PointSystem script = 
        gameObject.GetComponent<PointSystem>();
        script.Points(1);
        canSumPoints3 = false;
        myBuildText.GetComponent<Text>().enabled = false;
        count += 1;
    }
}

else if (hit.transform.tag == "WallDef4")
{
    if (canSumPoints4)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = 
            gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints4 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}

else if (hit.transform.tag == "WallDef5")
{
    if (canSumPoints155)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = 
            gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints155 = false;
            myBuildText.GetComponent<Text>().enabled = false;
        }
    }
}
count += 1;

} 

} 

} 

else if (hit.transform.tag == "WallDef6") 
{ 

if(canSumPoints5) 
{ 

myBuildText.GetComponent<Text>().enabled = true; 

if (Input.GetMouseButtonDown(0)) 
{
 PointSystem script = 

gameObject.GetComponent<PointSystem>(); 

script.Points(1); 

canSumPoints5 = false; 

myBuildText.GetComponent<Text>().enabled = false; 

} 

} 

else if (hit.transform.tag == "WallDef7") 
{ 

if(canSumPoints6) 
{ 

myBuildText.GetComponent<Text>().enabled = true; 

if (Input.GetMouseButtonDown(0)) 
{
 PointSystem script = 

gameObject.GetComponent<PointSystem>(); 

script.Points(1); 

canSumPoints6 = false; 

myBuildText.GetComponent<Text>().enabled = false; 

} 

} 

else if (hit.transform.tag == "WallDef8") 
{ 

if (canSumPoints7) 
{ 

myBuildText.GetComponent<Text>().enabled = true; 

if (Input.GetMouseButtonDown(0)) 
{
 PointSystem script = 

gameObject.GetComponent<PointSystem>(); 

} 

} 

}
```csharp
    script.Points(1);
    canSumPoints7 = false;
    myBuildText.GetComponent<Text>().enabled = false;
    count += 1;
}

else if (hit.transform.tag == "WallDef9")
{
    if (canSumPoints8)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints8 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
    else if (hit.transform.tag == "WallDef10")
    {
        if (canSumPoints9)
        {
            myBuildText.GetComponent<Text>().enabled = true;
            if (Input.GetMouseButtonDown(0))
            {
                PointSystem script = gameObject.GetComponent<PointSystem>();
                script.Points(1);
                canSumPoints9 = false;
                myBuildText.GetComponent<Text>().enabled = false;
                count += 1;
            }
        }
    }
    else if (hit.transform.tag == "WallDef11")
    {
        if (canSumPoints157)
        {
            myBuildText.GetComponent<Text>().enabled = true;
        }
    }
```
if (Input.GetMouseButtonDown(0))
{
    PointSystem script = gameObject.GetComponent<PointSystem>();
    script.Points(1);
    canSumPoints157 = false;
    myBuildText.GetComponent<Text>().enabled = false;
    count += 1;
}

else if (hit.transform.tag == "WallDef12")
{
    if (canSumPoints158)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints158 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}

else if (hit.transform.tag == "South1")
{
    if (canSumPoints10)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints10 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}
else if (hit.transform.tag == "South2")
{
    if (canSumPoints11)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButton(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints11 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}
else if (hit.transform.tag == "South3")
{
    if (canSumPoints12)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButton(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints12 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}
else if (hit.transform.tag == "South4")
{
    if (canSumPoints13)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButton(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints13 = false;
myBuildText.GetComponent<Text>().enabled = false;
count += 1;
}
else if (hit.transform.tag == "South5")
{
    if (canSumPoints14)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints14 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}
else if (hit.transform.tag == "South6")
{
    if (canSumPoints15)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints15 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}
else if (hit.transform.tag == "South7")
{
    if (canSumPoints16)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {

PointSystem script = gameObject.GetComponent<PointSystem>();
script.Points(1);
canSumPoints16 = false;
myBuildText.GetComponent<Text>().enabled = false;
count += 1;
}
}

else if (hit.transform.tag == "South8")
{
    if (canSumPoints17)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
canSumPoints17 = false;
myBuildText.GetComponent<Text>().enabled = false;
count += 1;
        }
    }
}

else if (hit.transform.tag == "South9")
{
    if (canSumPoints18)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
canSumPoints18 = false;
myBuildText.GetComponent<Text>().enabled = false;

        }
    }
}
else if (hit.transform.tag == "South10")
{
    if (canSumPoints19)
myBuildText.GetComponent<Text>().enabled = true;
    if (Input.GetMouseButtonDown(0))
    {
        PointSystem script = gameObject.GetComponent<PointSystem>();
        script.Points(1);
        canSumPoints19 = false;
        myBuildText.GetComponent<Text>().enabled = false;
        count += 1;
    }
}

else if (hit.transform.tag == "South11")
{
    if (canSumPoints20)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints20 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}

else if (hit.transform.tag == "South12")
{
    if (canSumPoints121)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints121 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}
else if (hit.transform.tag == "West1")
{
    if (canSumPoints122)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = GameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints122 = false;
            myBuildText.GetComponent<Text>().enabled = true;
            count += 1;
        }
    }
}
else if (hit.transform.tag == "West2")
{
    if (canSumPoints123)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = GameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints123 = false;
            myBuildText.GetComponent<Text>().enabled = true;
            count += 1;
        }
    }
}
else if (hit.transform.tag == "West3")
{
    if (canSumPoints124)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = GameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints124 = false;
            myBuildText.GetComponent<Text>().enabled = true;
            count += 1;
        }
    }
}
else if (hit.transform.tag == "West4")
{
    if (canSumPoints125)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints125 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}

else if (hit.transform.tag == "West6")
{
    if (canSumPoints126)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints126 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}

else if (hit.transform.tag == "West7")
{
    if (canSumPoints127)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints127 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}
else if (hit.transform.tag == "West8")
{
    if (canSumPoints128)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints128 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}

else if (hit.transform.tag == "West9")
{
    if (canSumPoints129)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints129 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}

else if (hit.transform.tag == "West10")
{
    if (canSumPoints130)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints130 = false;
            myBuildText.GetComponent<Text>().enabled = false;
        }
    }
}
count += 1;

} else if (hit.transform.tag == "West11") {
    if (canSumPoints131)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script =
            gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints131 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}

else if (hit.transform.tag == "West13") {
    if (canSumPoints132)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script =
            gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints132 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}

else if (hit.transform.tag == "West14") {
    if (canSumPoints133)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script =
            gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints133 = false;
            myBuildText.GetComponent<Text>().enabled = false;
        }
    }
}
false;
    count += 1;
}

else if (hit.transform.tag == "West15")
{
    if (canSumPoints134)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseDownAny(0))
        {
            PointSystem script =
            gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints134 = false;
            myBuildText.GetComponent<Text>().enabled =
false;
        }
    }
    count += 1;
}

else if (hit.transform.tag == "West16")
{
    if (canSumPoints135)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseDownAny(0))
        {
            PointSystem script =
            gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints135 = false;
            myBuildText.GetComponent<Text>().enabled =
false;
        }
    }
    count += 1;
}

else if (hit.transform.tag == "West17")
{
    if (canSumPoints136)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseDownAny(0))
        {
            PointSystem script =
            gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints136 = false;
        }
    }
}
myBuildText.GetComponent<Text>().enabled = false;
    count += 1;
  }
  }

else if (hit.transform.tag == "West18")
{
  if (canSumPoints137)
  {
    myBuildText.GetComponent<Text>().enabled = true;
    if (Input.GetMouseButton(0))
    {
      PointSystem script =
      gameObject.GetComponent<PointSystem>();
      script.Points(1);
      canSumPoints137 = false;
      myBuildText.GetComponent<Text>().enabled = false;
      count += 1;
    }
  }
}

else if (hit.transform.tag == "West19")
{
  if (canSumPoints138)
  {
    myBuildText.GetComponent<Text>().enabled = true;
    if (Input.GetMouseButton(0))
    {
      PointSystem script =
      gameObject.GetComponent<PointSystem>();
      script.Points(1);
      canSumPoints138 = false;
      myBuildText.GetComponent<Text>().enabled = false;
      count += 1;
    }
  }
}

else if (hit.transform.tag == "East1")
{
  if (canSumPoints139)
  {
    myBuildText.GetComponent<Text>().enabled = true;
    if (Input.GetMouseButton(0))
    {
      PointSystem script =
      gameObject.GetComponent<PointSystem>();
      script.Points(1);
canSumPoints139 = false;
myBuildText.GetComponent<Text>().enabled = false;
count += 1;
}
else if (hit.transform.tag == "East2")
{
    if (canSumPoints140)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonUp(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints140 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}
else if (hit.transform.tag == "East3")
{
    if (canSumPoints141)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonUp(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints141 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
}
else if (hit.transform.tag == "East4")
{
    if (canSumPoints142)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonUp(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
        }
    }
}
...L\UNITY\Player_Multi\Assets\Scripts\ActivateTextWall.cs

```csharp
script.Points(1);
canSumPoints142 = false;
myBuildText.GetComponent<Text>().enabled = false;
count += 1;
}

else if (hit.transform.tag == "East6")
{
    if (canSumPoints143)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
canSumPoints143 = false;
myBuildText.GetComponent<Text>().enabled = false;
count += 1;
        }
    }
}

else if (hit.transform.tag == "East7")
{
    if (canSumPoints144)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
canSumPoints144 = false;
myBuildText.GetComponent<Text>().enabled = false;
count += 1;
        }
    }
}

else if (hit.transform.tag == "East8")
{
    if (canSumPoints145)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script =
```
gameObject.GetComponent<PointSystem>();
  script.Points(1);
  myBuildText.GetComponent<Text>().enabled = false;
  canSumPoints145 = false;
  count += 1;
}
else if (hit.transform.tag == "East9")
{
  if (canSumPoints146)
  {
    myBuildText.GetComponent<Text>().enabled = true;
    if (Input.GetMouseButtonDown(0))
    {
      PointSystem script = gameObject.GetComponent<PointSystem>();
      script.Points(1);
      canSumPoints146 = false;
      myBuildText.GetComponent<Text>().enabled = false;
      count += 1;
    }
  }
}
else if (hit.transform.tag == "East10")
{
  if (canSumPoints147)
  {
    myBuildText.GetComponent<Text>().enabled = true;
    if (Input.GetMouseButtonDown(0))
    {
      PointSystem script = gameObject.GetComponent<PointSystem>();
      script.Points(1);
      canSumPoints147 = false;
      myBuildText.GetComponent<Text>().enabled = false;
      count += 1;
    }
  }
}
else if (hit.transform.tag == "East11")
{
  if (canSumPoints148)
  {
    myBuildText.GetComponent<Text>().enabled = true;
  }
}
PointSystem script = 
gameObject.GetComponent<PointSystem>();
script.Points(1);
canSumPoints148 = false;
myBuildText.GetComponent<Text>().enabled = false;
count += 1;
}
}
}
else if (hit.transform.tag == "East13")
{
  if (canSumPoints149)
  {
    myBuildText.GetComponent<Text>().enabled = true;
    if (Input.GetMouseButtonDown(0))
    {
      PointSystem script = 
      gameObject.GetComponent<PointSystem>();
      script.Points(1);
canSumPoints149 = false;
myBuildText.GetComponent<Text>().enabled = false;
count += 1;
    }
  }
}
else if (hit.transform.tag == "East14")
{
  if (canSumPoints150)
  {
    myBuildText.GetComponent<Text>().enabled = true;
    if (Input.GetMouseButtonDown(0))
    {
      PointSystem script = 
      gameObject.GetComponent<PointSystem>();
      script.Points(1);
canSumPoints150 = false;
myBuildText.GetComponent<Text>().enabled = false;
count += 1;
    }
  }
}
else if (hit.transform.tag == "East15")
{
  if (canSumPoints151)
  {
    myBuildText.GetComponent<Text>().enabled = true;
    if (Input.GetMouseButtonDown(0))
    {
...L\UNITY\Player_Multi\Assets\Scripts\ActivateTextWall.cs 23

1043 {
1044     PointSystem script = gameObject.GetComponent<PointSystem>();
1045     script.Points(1);
1046     canSumPoints151 = false;
1047     myBuildText.GetComponent<Text>().enabled = false;
1048     count += 1;
1049 }
1050 }
1051 }
1052 }
1053 else if (hit.transform.tag == "East16")
1054 {
1055     if (canSumPoints152)
1056     {
1057         myBuildText.GetComponent<Text>().enabled = true;
1058         if (Input.GetMouseButtonDown(0))
1059         {
1060             PointSystem script = gameObject.GetComponent<PointSystem>();
1061             script.Points(1);
1062             canSumPoints152 = false;
1063             myBuildText.GetComponent<Text>().enabled = false;
1064             count += 1;
1065         }
1066     }
1067 }
1068 }
1069 }
1070 else if (hit.transform.tag == "East17")
1071 {
1072     if (canSumPoints153)
1073     {
1074         myBuildText.GetComponent<Text>().enabled = true;
1075         if (Input.GetMouseButtonDown(0))
1076         {
1077             PointSystem script = gameObject.GetComponent<PointSystem>();
1078             script.Points(1);
1079             canSumPoints153 = false;
1080             myBuildText.GetComponent<Text>().enabled = false;
1081             count += 1;
1082         }
1083     }
1084 }
1085 }
1086 else if (hit.transform.tag == "East18")
1087 {
1088     if (canSumPoints154)
1089     {
1090         myBuildText.GetComponent<Text>().enabled = true;
if (Input.GetMouseButtonDown(0))
{
    PointSystem script = gameObject.GetComponent<PointSystem>();
    script.Points(1);
    canSumPoints154 = false;
    myBuildText.GetComponent<Text>().enabled = false;
    count += 1;
}
else if (hit.transform.tag == "East19")
{
    if (canSumPoints155)
    {
        myBuildText.GetComponent<Text>().enabled = true;
        if (Input.GetMouseButtonDown(0))
        {
            PointSystem script = gameObject.GetComponent<PointSystem>();
            script.Points(1);
            canSumPoints155 = false;
            myBuildText.GetComponent<Text>().enabled = false;
            count += 1;
        }
    }
    else if (hit.transform == null) //If our raycast doesn't hit anything don't show the text.
    {
        myBuildText.GetComponent<Text>().enabled = false;
    }
    else //If we hit something else, text to false.
    {
        myBuildText.GetComponent<Text>().enabled = false;
    }
    else //Again. Just in case the 2 first times it doesn't work, I guess...
    {

myBuildText.GetComponent<Text>().enabled = false;

if (count >= 56) //If the number of blocks built is greater equal or greater than 56 then it means the walls are built and the job is done!
{
    anim.animator.SetBool("isMovingBlock", false); //Stop the animation where we hold the block.
    CmdBlocksFalse(); //Hey server! Do something!
}

[Command]
void CmdBlocksFalse()
{
    RpcBlocksFalse();
}

[ClientRpc]
void RpcBlocksFalse() //Oh hello clients. Set the blocks of the gameObject they're parented to to unactive please.
{
    blocks.SetActive(false);
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Networking;

// THIS SCRIPT WILL SHOW US THE TEXT TO ALLOW US TO LOAD THE BLOCKS ONTO
// THE CRANE'S PLATFORM. WE WON'T GET INTO IT AS IT'S BEEN DONE BEFORE.

public class TextBlocks : NetworkBehaviour
{
    public Transform cameraTransform;

    GameObject myLoadText;

    // Use this for initialization
    void Start()
    {
        myLoadText = GameObject.FindGameObjectWithTag("LoadBlocks");
    }

    // Update is called once per frame
    void Update()
    {
        RaycastHit hit;
        Vector3 rayPos = cameraTransform.position + cameraTransform.forward * 1f;
        Debug.DrawRay(rayPos, cameraTransform.forward * 5f, Color.green, 0.5f);

        if (Physics.Raycast(rayPos, cameraTransform.forward, out hit, 5f))
        {
            if (hit.transform.tag == "Box")
            {
                PickUpBlock script = gameObject.GetComponent<PickUpBlock>();
                if (script.hasBlocks)
                {
                    myLoadText.GetComponent<Text>().enabled = true;
                }
            }
            else
            {
                myLoadText.GetComponent<Text>().enabled = false;
            }
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.UI;


public class EndingInfo : NetworkBehaviour {

    public Transform cameraTransform; // Camera Transform for the raycast.
    public GameObject fenceDoorText; // A public GameObject that will be the entrance door of the fence.
    public GameObject cPRText; // Text allowing us to perform CPR if someone died.
    public Text gamePointsFinal; // Text showing the final points.
    public Text gameHealthFinal; // Text showing the final health.
    public Text gameTimeFinal; // Text showing the final time.
    public GameObject finalInfo; // GameObject finalInfo. It is the UI that will contain the info.
    public GameObject lifeUI; // Various UI text GameObjects.
    public GameObject timeUI;
    public GameObject pointsUI;
    public GameObject reticule;
    public GameObject liniaOn;
    public GameObject cascOn;
    public GameObject earmuuffsOn;
    public GameObject arnesOn;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        RaycastHit hit;
        Vector3 rayPos = cameraTransform.position + cameraTransform.forward * 1f;
        Debug.DrawRay(rayPos, cameraTransform.forward * 5f, Color.green * 0.5f);

        if (Physics.Raycast(rayPos, cameraTransform.forward, out hit, 5f)) {
            if (hit.transform.tag == "FenceDoor") // If our raycast hits a the fence door tagged as "FenceDoor"
            {
            }
        }
    }
}

44
45        fenceDoorText.GetComponent<Text>().enabled = true; //Text allowing us to exit the construction site.
46        if (Input.GetMouseButtonDown(0)) //If we click run EndGame() function.
47            {
48                EndGame();
49            }
50        }
51
52    else //If we don't hit the door, set the text to unenabled.
53            {
54                fenceDoorText.GetComponent<Text>().enabled = false;
55            }
56
57
58    }  
59
60    public void EndGame() //Function EndGame() {
61        {
62            Timer script6 = gameObject.GetComponent<Timer>(); //Get the script Timer in attached to this gameObject.
63            string timeString = string.Format("Tiempo: {0:00}:{1:00}" , script6.minutes, script6.seconds); //Create a string "timeString" and define it.
64            gameTimeFinal.text = timeString; //Make the timeString display the gameTimeFina gameObject text.
65            PointSystem script = gameObject.GetComponent<PointSystem>(); //Same with the points. 
66            string pointString = string.Format("Puntuación: {0}pts", script.points);
67            gamePointsFinal.text = pointString;
68            Health2 script2 = gameObject.GetComponent<Health2>(); //Same with the time, although in the middle we round the currentHealth to an integer.
69            int finalPoints = Mathf.RoundToInt(script2.currentHealth);
70            string healthString = string.Format("Vida: {0}/100", finalPoints);
71            gameHealthFinal.text = healthString;
72            FallDamage2 script3 = gameObject.GetComponent<FallDamage2>(); //Get the FallDamage script.
73            CollisionExcavator script4 = gameObject.GetComponent(CollisionExcavator()); //Get the Collision Excavator script.
75            finalInfo.SetActive(true); //Set the UI displaying all the info to active.
76            script.enabled = false; //Disable the PointSystem script. We'll get no more point updates.
77            script2.enabled = false; //Disable the Health2 script. We'll get no more health updates.
78            script3.enabled = false; //Disable the FallDamage script. We'll get no more fall damage updates.
no more damage for falls (even though we wouldn't get none because Health2 is disabled).

script4.enabled = false; //Disable the CollisionExcavator script. Although our player will still be able to collide with it, we won't get damage.

script5.enabled = false; //Disable the Collision Platform script.

fpsController.enabled = false; //Disable the controllers!!! This way our player will stay still and we won't be able to move it.

lifeUI.SetActive(false); //Set some other UI's and text's to unactive.

timeUI.SetActive(false);

pointsUI.SetActive(false);

reticule.SetActive(false);

liniaOn.SetActive(false);

cascOn.SetActive(false);

earmuuffsOn.SetActive(false);

arnesOn.SetActive(false);

fenceDoorText.SetActive(false);

cPRText.SetActive(false);

cPRTex.SetActive(false);
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

//THIS SCRIPT WILL ALLOW US TO BUILD THE WALLS. THIS SCRIPT IS
//REPEATED IN A VERY SIMILAR WAY FOR ALL THE BLOCKS TO BE BUILT.
//THE ONLY THING THAT CHANGES BETWEEN SCRIPTS IS THE TAG OF
//THE WALL.

public class BuildWall : NetworkBehaviour {

    public Transform cameraTransform; //Camera for the transform.

    Renderer[] brickRender; //Array of Renderers called "brickRenderer".
    GameObject[] wall; //Array of GameObjects called "wall".

    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    void Update () {
        RaycastHit hit;
        Vector3 rayPos = cameraTransform.position + cameraTransform.forward *
            1f;
        Debug.DrawRay(rayPos, cameraTransform.forward * 5f, Color.green, 0.5f);

        if (Physics.Raycast(rayPos, cameraTransform.forward, out hit, 5f))
        {
            if (hit.transform.tag == "WallDef") //If our raycast hits a
                gameObject tagged as "WallDef".
            {
                Debug.Log("Build");
                PickUpBoxRoof script = gameObject.GetComponent<PickUpBoxRoof>();
                if (script.canBuild) //If the variable canBuild from
                    PickUpBoxRoof is true...
                {
                    if (Input.GetMouseButtonDown(0)) //If we press the left
                        mouse...
                    {
                        script.clicks += 1; //Add one to the clicks variable
                        in the PickUpBoxRoof script.
                        CmdBuildWall(); //Hey Server! Do something!
                    }
                }
            }
        }
    }
}
[Command]
void CmdBuildWall()
{
    RpcBuildWall();
}

[ClientRpc]
void RpcBuildWall()
{
w = GameObject.FindGameObjectsWithTag("WallDef"); //Same stuff
define that we did several times already.
brickRender = new Renderer[w.length];
for (int i = 0; i < wall.Length; i++)
{
    wall[i].layer = 0;
    brickRender[i] = wall[i].GetComponent<Renderer>();
    brickRender[i].enabled = true;
}
}
using UnityEngine;
using System.Collections;

// THIS SCRIPT WILL CONTROL THE UPWARD MOVEMENT OF THE CRANE, WHEN BEING LOADED WITH THE BLOCKS.

class crane_animate1 : MonoBehaviour
{
    public Animator animator; // Declare the animator.
    public AnimatorControllerParameter animParam; //_this came with the prefab, I don't know what it does but I didn't want to remove it.
    public float rotateYaw; // Floats that will act as variables.
    public float dolly;
    public float hook;
    public bool canPickBlocksUp; // Bool that will allow our player to pick up the blocks from the platform.

    public bool isMoving = false; // Bool that will declare that the platform is not moving
    float randomYawIncrease; // Velocity at which the previous floats will increase.
    float randomDollyIncrease;
    float randomHookIncrease;
    public float temps;

    void Start()
    {
        randomYawIncrease = 5f;
        randomDollyIncrease = 10f;
        randomHookIncrease = 10f;
        temps = 0;
        rotateYaw = 0; // Starting Position of the crane's yaw and hook.
        dolly = 50;
    }

    void Update()
    {
        MoveUp(); // Rune the function MoveUp().
    }

    public void MoveUp()
    {
        temps += Time.deltaTime; // Define the float temps. It will add
rotateYaw += Time.deltaTime * randomYawIncrease; //The rotation of the yaw will be equal to the time * velocity.
dolly += Time.deltaTime * randomDollyIncrease; //The rotation of the dolly will be equal to the time * velocity.
hook = (temps * (-randomHookIncrease) + 86); //The rotation of the hook will be equal to the time * (-velocity)
//because it’s going down, and + 86 because this way it is at ground level

if (rotateYaw >= 60)
{
    rotateYaw = 60; //If yaw is at 60 degrees stop.
}
if (dolly >= 90)
{
    dolly = 90; //If dolly is at height 90 stop.
}
if (hook <= 10)
{
    hook = 10; //If hook is at 10 then
    if (rotateYaw == 60)
    {
        hook = -((temps * (-randomHookIncrease)) + 100) / 2.0f; //Take down the hook.
        if (hook >= 43.8f) //When the platform is touching the top floor...
        {
            canPickBlocksUp = true; //Our player will be able to pick up the blocks.
            hook = 43.8f; //The hook will stay in position.
        }
        temps = 0; //Temps will be reset.
        isMoving = false; //The platform won't be moving, so our player won't get hurt by it.
        this.enabled = false; //Disable this script to let the “Crane” script control the movement.
    }
}
}

animator.SetFloat("Rotate_YAW", Mathf.Abs(rotateYaw) % 360); // animator values.
animator.SetFloat("dolly", dolly);
animator.SetFloat("hook", hook);
...i\Assets\Assets_TowerCranes-1\scripts\crane_animate1.cs
3
90
91
92
93
94
95
96
using UnityEngine;
using System.Collections;

//THIS SCRIPT WILL CONTROLL THE CRANE GOING DOWN.

public class Crane : MonoBehaviour
{
    //Same paranmeters as in crane_animate1.
    public Animator animator;
    public AnimatorControllerParameter animParam;

    public float rotateYaw;
    public float dolly;
    public float hook;

    public bool isMoving = false;

    float randomYawIncrease;
    float randomDollyIncrease;
    float randomHookIncrease;

    //2 floats to control the time here.
    public float temps1;
    public float temps2;

    public void Start()
    {
        randomYawIncrease = 6f; //Velocities
        randomDollyIncrease = 10f;
        randomHookIncrease = 10f;
        rotateYaw = 60; //Initial Positions
        dolly = 90;
        temps1 = 0;
    }

    void Update()
    {
        GoDown();
    }

    void GoDown()
    {
        temps1 += Time.deltaTime; //Define temps1.

        Hook = (temps1 * (-randomHookIncrease) + 43.8f); //Move up the hook.

        if (hook <= 10) //When the hook gets to 10...
    }
hook = 10;
dolly -= Time.deltaTime * randomDollyIncrease; //Start moving dolly.
rotateYaw -= Time.deltaTime * randomYawIncrease; //Start moving yaw.

if (dolly <= 50) //When dolly hits 50...
{
    temps2 += Time.deltaTime; //Declare temps2...
dolly = 50;
hook += (temps2 * randomHookIncrease); //Get hook down.

    if (rotateYaw <= 0) //When yaw is at 0 degrees.
    {
        rotateYaw = 0;
        if (hook >= 86) //If hook is larger than 86...
        {
            hook = 86; //Hook equals 86.
            temps1 = 0; //Reset temps1 and temps2
temps2 = 0;
            isMoving = false; //We cannot get hurt because it's not moving.

            this.enabled = false; //Disable this script so that "crane_animate1" can control it.
        }
    }
}

animator.SetFloat("Rotate_YAW", Mathf.Abs(rotateYaw) % 360); // Animator parameters.
animator.SetFloat("dolly", dolly);
animator.SetFloat("hook", hook);
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

// THIS SCRIPT WILL ALLOW US TO DRAW PATHS TO BE FOLLOWED BY OUR AI'S.
// Check the following YouTube video for a better explanation:
// https://www.youtube.com/watch?v=1aBjTa3xQzE&t=1959s

public class EditorPathScript : MonoBehaviour
{

    public Color rayColor = Color.white;
    public List<Transform> path_objs = new List<Transform>();
    Transform[] theArray;

    void OnDrawGizmos()
    {
        Gizmos.color = rayColor;
        theArray = GetComponentsInChildren<Transform>();
        path_objs.Clear();

        foreach (Transform path_obj in theArray)
        {
            if (path_obj != this.transform)
            {
                path_objs.Add(path_obj);
            }
        }

        for (int i = 0; i < path_objs.Count; i++)
        {
            Vector3 position = path_objs[i].position;
            if (i > 0)
            {
                Vector3 previous = path_objs[i - 1].position;
                Gizmos.DrawLine(previous, position);
                Gizmos.DrawWireSphere(position, 0.3f);
            }
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

//THIS SCRIPT WILL CONTROL THE MOVEMENT OF THE EXCAVATOR

public class MoveOnPathScript : MonoBehaviour {

    public EditorPathScript PathToFollow; //Public field called Editor Path
        //Script where we'll drag the script in the inspector.
    public int CurrentWayPointID = 0; //Current point in the path.
    public float speed; //Speed of the excavator.
    private float reachDistance = 2.0f; //Distance at which we will pass to
        //the next point.
    public float rotationSpeed = 2.0f; //Rotation speed.

    Vector3 last_position;
    Vector3 current_position;

    // Use this for initialization
    void Start () {
        last_position = transform.position;
    }

    // Update is called once per frame
    void Update () {
        float distance = Vector3.Distance(PathToFollow.path_objs
                [CurrentWayPointID].position, transform.position); //We'll call
            //distance to the
        //distance between the point the excavator is going to and the
        //position of the excavator.
        transform.position = Vector3.MoveTowards(transform.position,
                PathToFollow.path_objs[CurrentWayPointID].position, Time.deltaTime * speed);
        //Move the excavator from its position to the next point position at
            //a certain velocity.
        var lookPos = PathToFollow.path_objs[CurrentWayPointID].position -
                transform.position; //Look towards the next point.
        lookPos.y = 0;
        var rotation = Quaternion.LookRotation(lookPos); //Rotate with
            //lookPos.
        transform.rotation = Quaternion.Slerp(transform.rotation, rotation, Time.deltaTime * rotationSpeed); //Make the rotation smooth.

        if (distance <= reachDistance)
        {
            CurrentWayPointID++;
            //Change point when reached the minimum
        }
distance.

if (CurrentWayPointID >= PathToFollow.path_objs.Count)
{
    CurrentWayPointID = 0; // Go back to point 0 if the path is over.
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

// THIS SCRIPT WILL CONTROL THE BEHAVIOUR OF THE A.I. BOT PERFORMING SOME WORK
// ON THE FIRST FLOOR. ITS MOVEMENT IS BASED ON THE SAME PRINCIPLE AS THE
// EXCAVATOR, SO ONLY THE NEW PARTS WILL BE COMMENTED.

public class MoveOnPathScript2 : MonoBehaviour
{
    public EditorPathScript PathToFollow;
    public int CurrentWayPointID = 0;
    public float speed;
    private float reachDistance = 1.0f;
    public float rotationSpeed = 5.0f;
    public GameObject particles; // Public GameObject called "particles".
    CapsuleCollider capsule; // A capsule collider called "capsule"
    Vector3 last_position;
    Vector3 current_position;
    public bool crouch = false; // Public bool to tell if it is crouching or not.
    Animator anim; // Animator called "anim".

    // Use this for initialization
    void Start()
    {
        capsule = gameObject.GetComponent<CapsuleCollider>(); // Define the capsule as the CapsuleCollider component attached to the A.I. bot.
        anim = GetComponent<Animator>(); // Same with anim.
        last_position = transform.position;
        particles.SetActive(false); // Make the particles invisible.
    }

    // Update is called once per frame
    void Update()
    {
        anim.SetBool("IsWalking", true); // Make the AI bot walk.
        float distance = Vector3.Distance(PathToFollow.path_objs[CurrentWayPointID].position, transform.position) / reachDistance;
        transform.position = Vector3.MoveTowards(transform.position, PathToFollow.path_objs[CurrentWayPointID].position, Time.deltaTime * speed);

        var lookPos = PathToFollow.path_objs[CurrentWayPointID].position - transform.position;
lookPos.y = 0;
var rotation = Quaternion.LookRotation(lookPos);
transform.rotation = Quaternion.Slerp(transform.rotation, rotation, Time.deltaTime * rotationSpeed);

if (distance <= reachDistance)
{
    CurrentWayPointID++;
}

if (CurrentWayPointID >= PathToFollow.path_objs.Count)
{
    CurrentWayPointID = 10; //When reaching the end of his path, stop.
    anim.SetBool("IsWalking", false); //Stop the walking animation.
    crouch = true; //Crouch is true.
}

anim.SetBool("IsCrouching", true); //Play the crouched animation.

if (crouch) //If it is crouching
{
    particles.SetActive(true); //Make the particles visible (the bot has started his work).
    gameObject.GetComponent<AudioSource>().enabled = true; //Play the AudioSource attached to the bot with the work sound.
    capsule.height = 1.8f; //Change capsule height and center. (I'm quite sure it's more or less the same).
    capsule.center = new Vector3(0, 0.8f, 0);
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

// THIS SCRIPT CONTROLS THE MOVEMENT OF THE OTHER A.I.BOT.
// THE SCRIPT IS REALLY SIMILLAR TO THE TWO "MOVEONPATHSCRIPT"
// COMENTED BEFORE.

public class MoveOnPathScript3 : MonoBehaviour
{
    public EditorPathScript PathToFollow;
    public int CurrentWayPointID = 0;
    public float speed;
    private float reachDistance = 1.0f;
    public float rotationSpeed = 5.0f;
    CapsuleCollider capsule;

    Vector3 last_position;
    Vector3 current_position;

    Animator anim;
    // Use this for initialization
    void Start()
    {
        capsule = gameObject.GetComponent<CapsuleCollider>();
        anim = GetComponent<Animator>();
        last_position = transform.position;
    }

    // Update is called once per frame
    void Update()
    {
        anim.SetBool("IsWalkingWheel", true);
        float distance = Vector3.Distance(PathToFollow.path_objs[CurrentWayPointID].position, transform.position);
        transform.position = Vector3.MoveTowards(transform.position, PathToFollow.path_objs[CurrentWayPointID].position, Time.deltaTime * speed);

        var lookPos = PathToFollow.path_objs[CurrentWayPointID].position - transform.position;
        lookPos.y = 0;
        var rotation = Quaternion.LookRotation(lookPos);
        transform.rotation = Quaternion.Slerp(transform.rotation, rotation, Time.deltaTime * rotationSpeed);
        if (distance <= reachDistance)
49     {
50         CurrentWayPointID++;
51     }
52
53     if (CurrentWayPointID >= PathToFollow.path_objs.Count)
54     {
55         CurrentWayPointID = 0;
56     }
57
58 }
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Audio;

// THIS SCRIPT WILL MAKE THE PLAYERS GET HURT IF THEY GET TOO CLOSE TO THE SPARKS.

public class TriggerEspurna : MonoBehaviour {

    public AudioClip clips; // Sound of the work being performed.

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }

    void OnTriggerStay(Collider other) // If a collider stays inside the trigger...
    {
        { 
            if (other.transform.tag == "Player" && path.crouch) // If the collider inside the trigger is a player and the AI is crouching (so it has already started to work)...
            {
                Health2 player = other.GetComponent<Health2>(); // Get the Health2 script attached to the object that has collided with the trigger (The Player)
                PlaySound(); // Play the sound of damage.
                if (player != null) // If health is not null
                {
                    player.TakeDamage(0.5f); // Take a damage of 0.5 for every frame the player is inside the trigger.
                }
            }
        }

    }

    void PlaySound()
    {
        AudioSource source = gameObject.AddComponent<AudioSource>(); // Add an audio source called source.
        source.clip = clips; // The clip that the source will play is called clips.
        source.Play(); // Play the source.
Destroy(source, clips.length); //Destroy the source when "clips" has stopped playing.