Cartilla para la creación de videojuegos serios en realidad virtual con enfoque para la salud en Unity 3D

Producto de Apropiación Social del Conocimiento

Universidad Autónoma de Manizales

Nicolás Herrera Sepúlveda

Carolina Márquez Narváez

Tabla de Contenidos

1. Introducción

El presente manual tiene el objetivo de especificar el proceso de creación adecuado de minijuegos serios en realidad virtual para la salud. Estos minijuegos se emplearán en la plataforma de creación de videojuegos Unity 3D, y seguirán ciertos estándares para asegurar la homogeneidad entre los minijuegos de la plataforma.

Este manual podrá dividirse en tres partes. En la primera parte, se especificarán todas las bases para el manejo de la plataforma de Unity, creación del proyecto, y otros protocolos para la creación de la base del minijuego. Luego, se especificará la creación del ambiente virtual y de otros elementos audiovisuales que conforman la parte inmersiva de la plataforma. Por último, habrá una sección dedicada a distintos problemas comunes que se presentan en la plataforma, así como elementos que pueden ser necesarios para la implementación de funcionalidades específicas.

1.1. Objetivo

Esta guía busca instruir en la creación de videojuegos en Unity que sean compatibles y homogéneos entre sí. Esto será posible al seguir el ejemplo del videojuego Alien Squisher. Alien Squisher es un juego en el que el usuario deberá defender una plantación de verduras pisando alienígenas invasores. Para ello, necesitaremos crear un escenario virtual para la plantación de verduras, así como crear un alienígena con un comportamiento independiente, que intente buscar las verduras.

Durante este proceso aprenderá a buscar recursos para añadir a su juego, crear una escena virtual y completarla con objetos y componentes para hacerla tener una lógica y un comportamiento esperado. Al completar este ejemplo, usted ya tendrá las bases para crear sus propios videojuegos y aportar a protocolos de videojuegos serios.

1.2. Consideraciones a tener

Debido a que el propósito del videojuego realizado en esta guía es el de rehabilitación y prevención fisioterapéutica, es importante tener en cuenta que el minijuego tenga una lista de objetivos para lograr que los pacientes realicen su respectivo ejercicio de manera correcta. Así mismo, sirve establecer los objetivos del videojuego antes de iniciar a desarrollarlo. A continuación podrá encontrar un par de guías generales que su minijuego debería seguir.

1.2.1. Videojuegos serios

Los videojuegos realizados en esta guía son videojuegos serios. **Tarja et al** definen los videojuegos serios por su priorización del aprendizaje, concentración y comunicación

por encima del entretenimiento[1]. Por ende, los videojuegos que realice deben buscar ayudar al usuario a realizar sus ejercicios. La parte artística y creativa de los videojuegos deben cumplir el propósito de hacer las tareas del jugador menos monótonas. Por esto se intenta convertir la tarea fisioterapéutica en una actividad con una historia en la que el jugador sea el protagonista.

Esto también significa que el entretenimiento no es el propósito principal de estos videojuegos. Evite hacer juegos en los que hayan actividades que tomen más prioridad que la propuesta por fisioterapia. La actividad fisioterapéutica debe ser el enfoque principal de su juego, y usted debe utilizar el entretenimiento, la inmersión y otras herramientas creativas para lograr este ejercicio.

1.2.2. Videojuegos en realidad virtual orientados en salud

Además de ser videojuegos serios, estos también están enfocados en mejorar la salud del usuario. En el caso de algunos videojuegos para la salud, y en el caso del videojuego desarrollado en esta guía, la población objetivo son adultos mayores. Debe buscar que sus juegos sean una experiencia placentera para el usuario. Evite juegos que aumenten mucho el nivel de estrés del jugador, como simuladores de guerras o juegos con sangre y violencia. Se recomiendan juegos en los que el jugador pueda cumplir una meta clara y relevante, sin poner mucha presión encima sobre las consecuencias de perder el videojuego. Después de todo, el jugador siempre podrá volver a intentar cualquier ejercicio que le presente dificultades.

Otra restricción con la que tendrá que pensar sus videojuegos, es que estos serán desarrollados para ser jugados en una plataforma de realidad virtual. Esta es una plataforma que mientras provee un increíble estado de inmersión en el jugador, también hace surgir ciertos inconvenientes. Por ejemplo, problemas como el mareo inducido por simulación[2], vértigo, y desorientación.

Evite juegos que puedan causarle una sensación de vértigo al jugador, como simular una caída en paracaídas o hacer al jugador volar un vehículo como un helicóptero o un avión. Tampoco es buena idea hacer juegos en los que el jugador deba acelerar a velocidades muy altas, puesto que contrasta con la falta de movimiento que tiene en el mundo real. Si desea hacer un juego basado en navegar un vehículo, introduzca al jugador a velocidades muy bajas, desbloqueando las mayores velocidades a medida que se acostumbra al ambiente del videojuego y a la disparidad con el mundo real.

---

[1] Tarja, S, Mikael, J, Per, B. (2007). Serious Games: An overview

[2] Dużmańska, N, Strojny, P & Strojny, A. (2018). Can Simulator Sickness Be Avoided? A Review on Temporal Aspects of Simulator Sickness. https://www.frontiersin.org/articles/10.3389/fpsyg.2018.02132/full

Por último, considere que hay un espacio de movimiento pequeño para desarrollar el juego. La siguiente figura ilustra cómo se relaciona el espacio de acción del jugador con un hipotético ambiente virtual. El jugador puede observar el ambiente real y sentirse completamente inmerso, pero por restricciones del mundo real no debe desplazarse mucho.

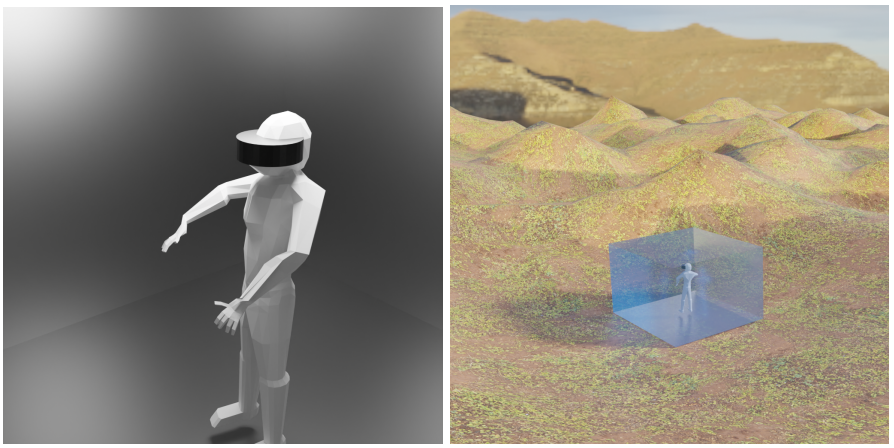


Figura 1.2.2.1. Comparación del jugador en el mundo real, y en el ambiente simulado

Como puede observarse, es fácil olvidar que el mundo real cuenta con unas restricciones de espacio que el videojuego no. Si su videojuego motiva al jugador a caminar mucho y a explorar un mundo abierto, lo más probable es que olvide que está atado al sistema de realidad virtual con cables y que se ubica en una sala real con paredes y más pequeña que el ambiente real. Es entonces su responsabilidad asegurarse de que el videojuego ejecute su función en un área pequeña, y que el jugador recuerde en todo momento no salirse del área designada de juego.

1.3. Requisitos previos y recomendaciones

Esta guía se desarrollará usando el software Unity 3D. Este programa tiene una parte gráfica que se explicará en esta guía, así como una parte en código que nos permitirá programar nuestro juego como necesitemos. Esta parte se desarrollará en el lenguaje de programación de C#, el cual permite crear la lógica de nuestro juego de manera sencilla. Por esto es necesario tener conocimientos básicos de lógica de programación, así como repasar las reglas de sintáctica de C#.

El software necesario para la desarrollar esta guía es el siguiente:

- Unity Hub
- Unity 2019.4.19f1
- Visual Studio Code / Visual Studio Community Edition

Adicionalmente, se recomienda instalar los siguientes programas gratuitos. Aunque no es necesario aprender sobre modelado en 3D, creación de audios o de imágenes, tener estas herramientas puede facilitar el proceso de desarrollo de videojuegos.

- Blender para el trabajo de recursos 3D
- GIMP, Krita o Paint.NET para edición y creación de imágenes
- Audacity para la edición y creación de audio

2. Creación del proyecto y bases para el trabajo en la plataforma Unity 3D

2.1. Diseño del videojuego

2.1.1. Elección del videojuego y creación de una ficha técnica

Al iniciar el proceso de creación de videojuegos serios, el primer paso consiste en idear un videojuego asociado con la actividad específica que deba facilitar. En el caso de esta guía, el videojuego ayudará a realizar una actividad física que corresponde a la rehabilitación de un paciente. Es bueno imaginar un videojuego que encaje bien con la actividad propuesta, por lo cual se realiza una ficha técnica explicando el funcionamiento del videojuego, cómo se relaciona con la actividad física, entre otros aspectos. Para esta guía se utiliza un sistema de códigos para separar a los distintos videojuegos. El código del videojuego representa la actividad 7 del día 5, de la fase 2. La ficha técnica para el videojuego puede encontrarse **aquí.**

2.1.2 Realizar un mockup para el videojuego

Este paso es opcional, aunque ayuda a visualizar mejor el juego y a facilitar la creación del escenario 3D. Este mockup será una representación artística o un esquema de cómo debe quedar el escenario 3D y los objetos con los que el jugador podrá interactuar. Objetos decorativos y otra información no relacionada con el juego no tiene que ir en estos mockups. A continuación podrá observar el mockup para el juego realizado en este manual.

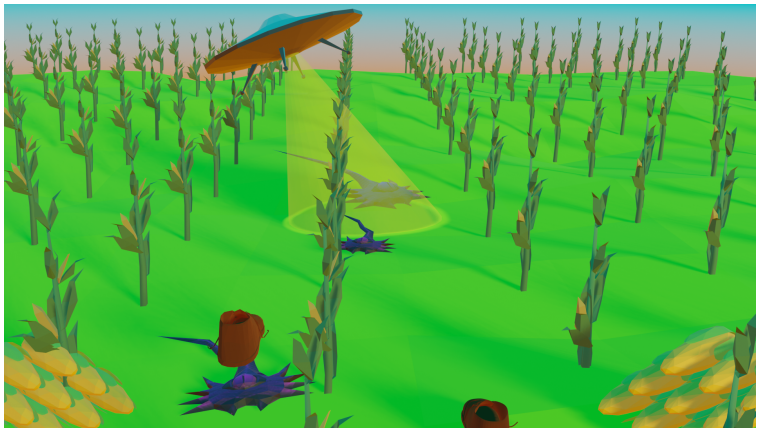


Figura 2.1.1. Mockup para el videojuego Alien Squisher

Como se puede observar, este mockup contiene información acerca de los objetos funcionales dentro del videojuego. Se puede observar una representación de los alienígenas bajando de la nave espacial, las botas que el jugador usará para pisarlos, y los vegetales a los que los alienígenas intentan llegar.

2.2. Creación del proyecto

2.2.1. Iniciar proyecto en Unity 3D

Primero que nada, debemos asegurarnos de tener Unity 3D con la versión correcta de su editor.

2.2.1.1. Descarga del Hub de Unity

Para empezar a trabajar en nuestros juegos, debemos instalar el Hub de Unity así como Unity en nuestro computador. El hub de Unity es la aplicación que nos proveerá el control sobre nuestros proyectos, así como las diferentes versiones de Unity que manejemos. Para descargarlo, deberemos ir a la página de Unity, en la sección de planes personales. Podremos elegir bien puede ser el plan estudiantil, si pertenecemos a una institución educativa, o al plan personal, en caso de que nuestros ingresos no superen un límite anual. Deberemos hacer registro de una cuenta de Unity si no lo hemos hecho antes.

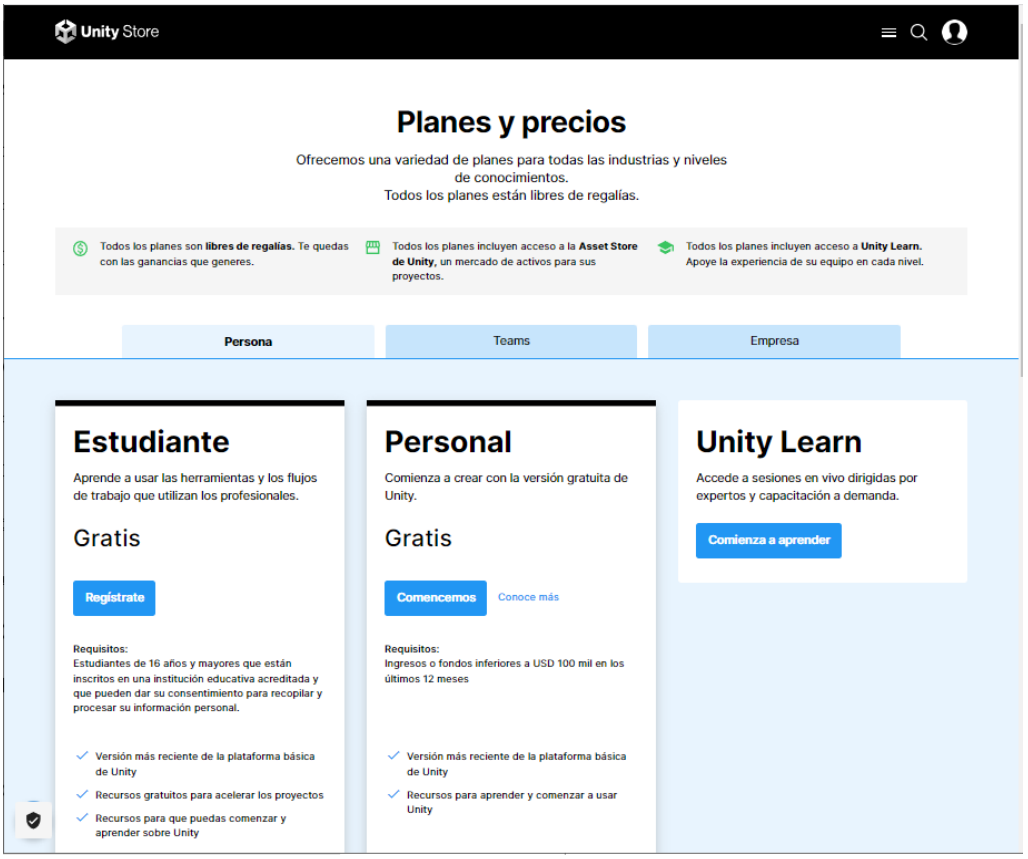


Figura 2.2.1.1.1. Página de planes de Unity.

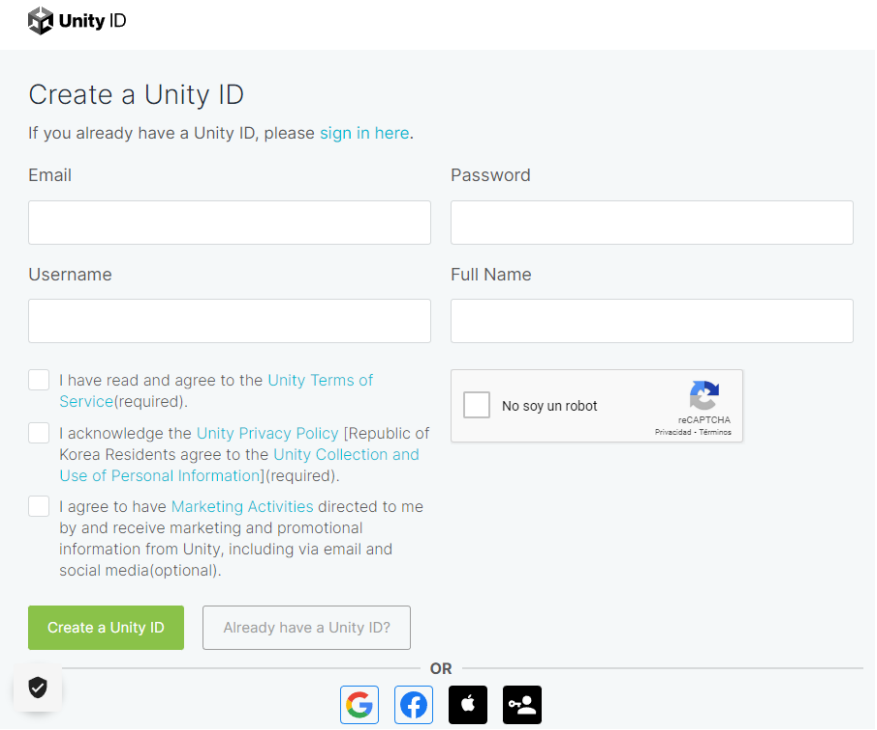


Figura 2.2.1.1.2. Página de registro de Unity

Una vez creado el usuario, podemos acceder a la página de descargas del hub de Unity. Es recomendable descargar la última versión, recalcada con un botón azul.

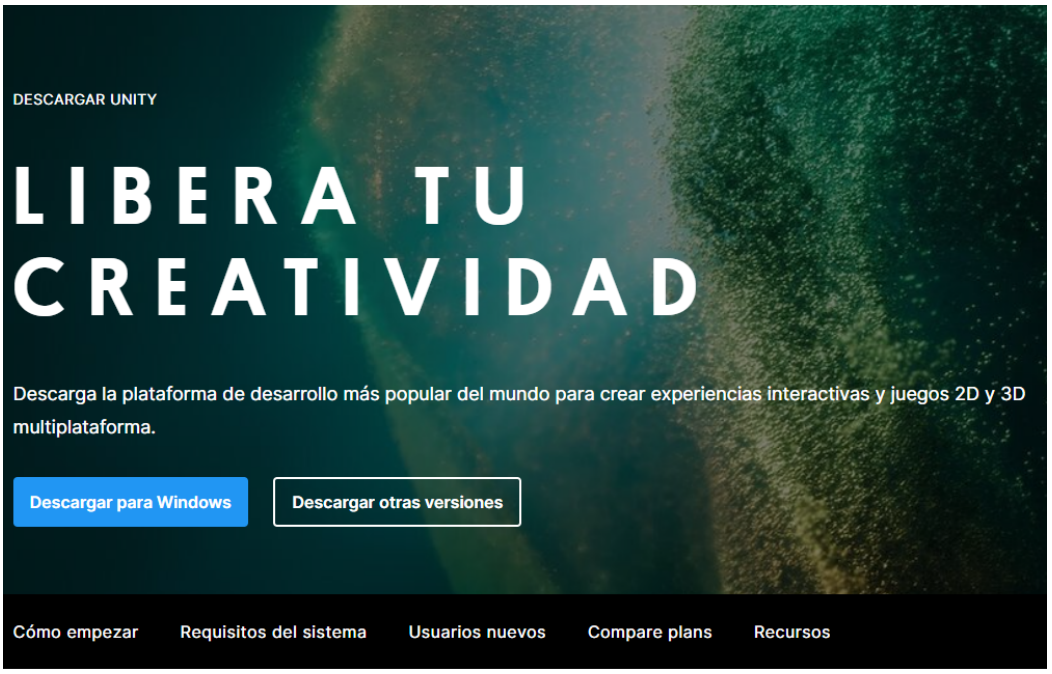


Figura 2.2.1.1.3. Página de descarga de Unity Hub

A partir de aquí podremos realizar la instalación de Unity Hub con el archivo que fue descargado en nuestro computador. Es recomendable dejar las configuraciones predeterminadas que se nos presentan.

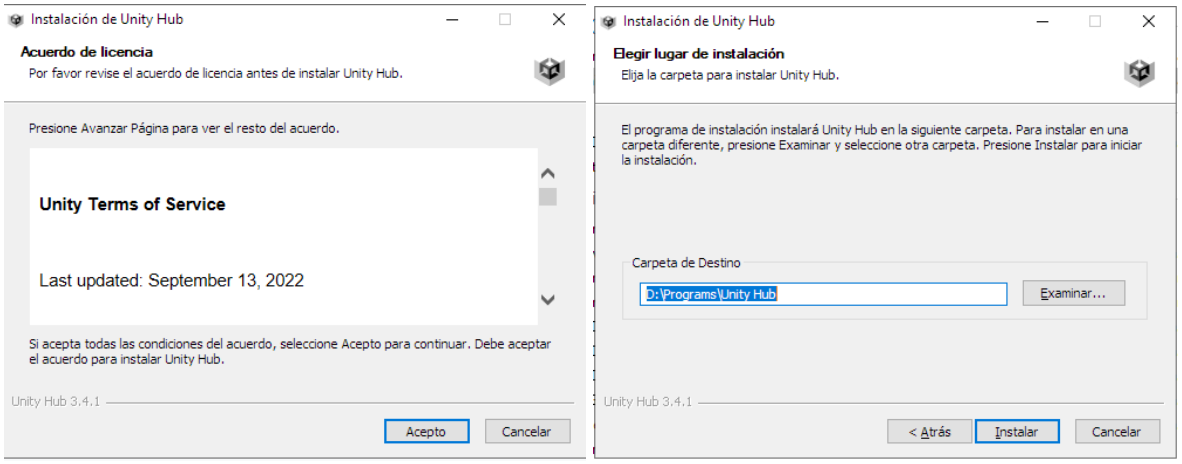


Figura 2.2.1.1.4. Pasos de instalación de Unity Hub

2.2.1.2. Uso de Unity Hub

Una vez instalado, podemos acceder a la aplicación de Unity Hub, donde podremos revisar todos nuestros proyectos y versiones de Unity. Vamos a empezar seleccionando nuestra versión de Unity. Podemos instalarla o descargarla al entrar a la pestaña de instalaciones, donde aparecerán todas las versiones de Unity descargadas.

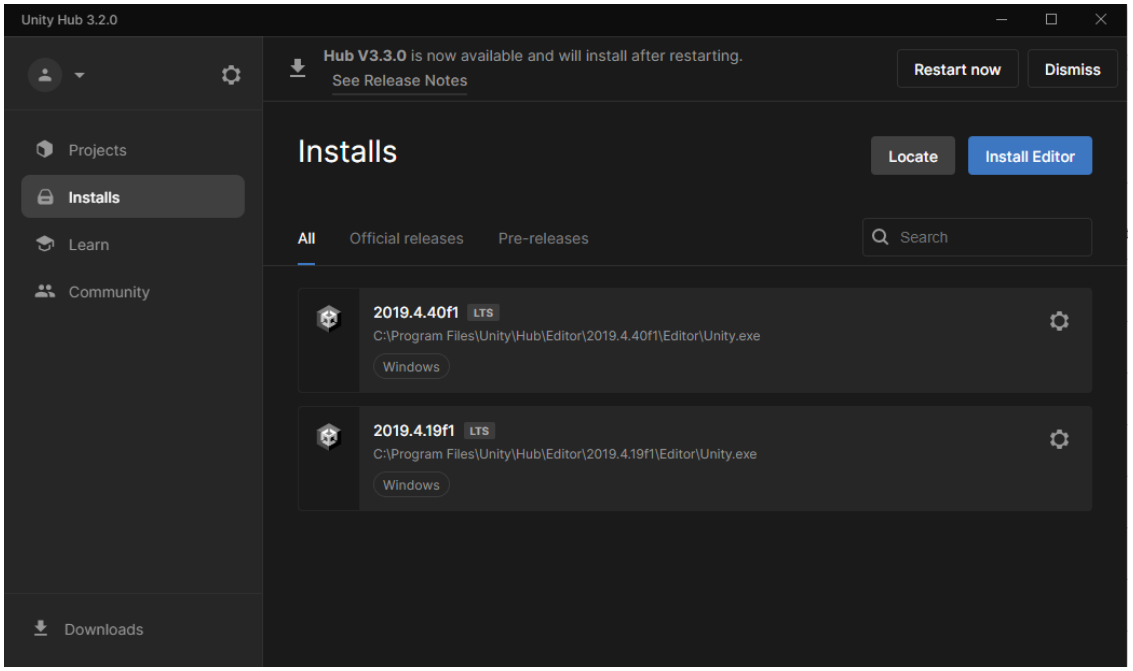


Figura 2.2.1.2.1 Pestaña de instalaciones

En caso de que la versión necesitada no aparezca en la lista, deberá instalarla en la opción de instalar editor. Esta opción generará una ventana en la que aparecerán las versiones de Unity disponibles para descargar. En este tutorial se trabajará en la versión de Unity 2019.4.19f1.

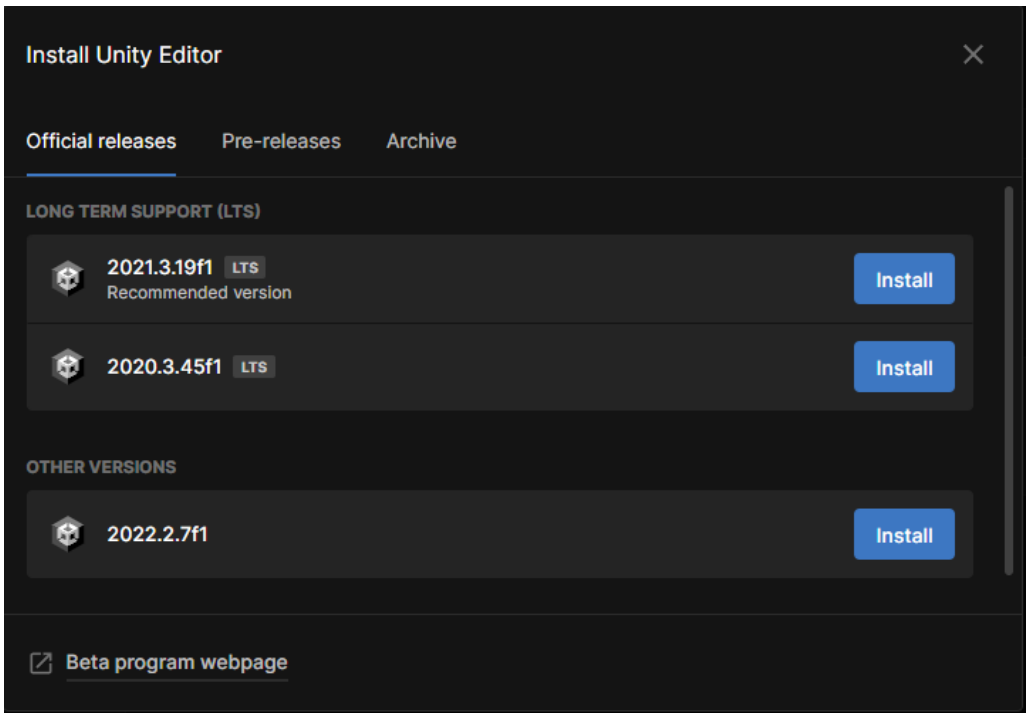


Figura 2.2.1.2.2. Ventana de descarga de versiones de Unity

Luego de tener el editor correcto instalado, podemos acceder a la pestaña de proyectos, donde crearemos un minijuego nuevo. Al estar en esta pestaña, hacer clic en el botón azul para crear un nuevo proyecto. Este botón nos abrirá otra ventana, donde deberemos elegir la opción para crear un proyecto nuevo en 3D (Core).

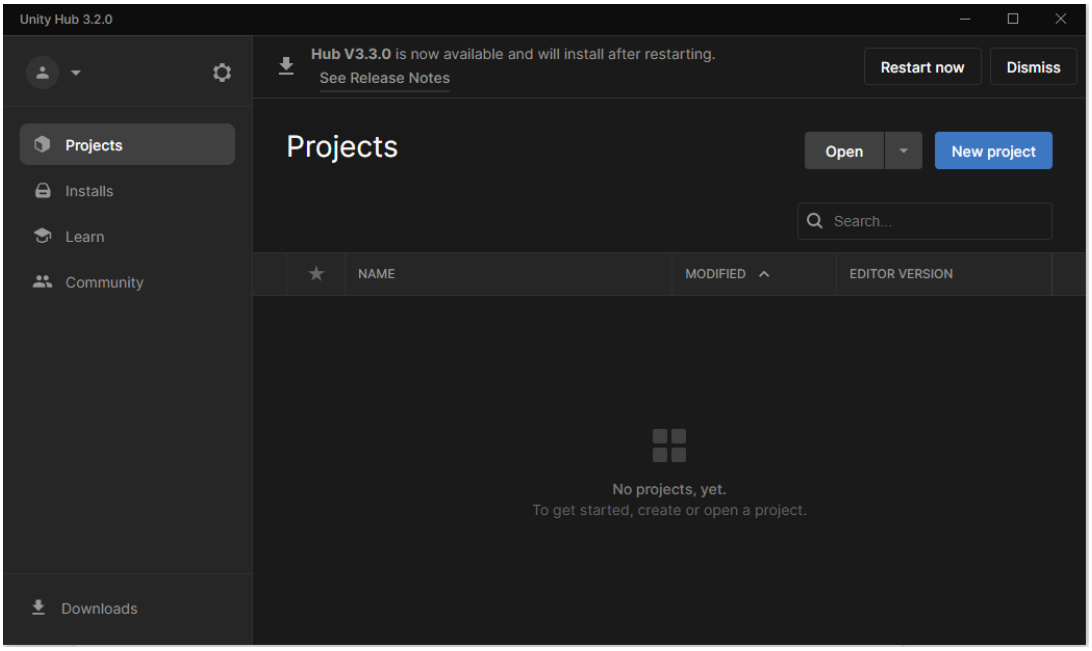


Figura 2.2.1.2.3. Pestaña de manejo de proyectos.

Como se puede observar en la siguiente figura, seleccionamos un nuevo proyecto 3D con la versión del editor de Unity especificada anteriormente. Acá además podremos poner un título para el proyecto, así como una ubicación en nuestro almacenamiento.

Recuerde que el nombre del proyecto debe corresponder al código asignado a su minijuego, así como al título que le asignó en la ficha técnica.

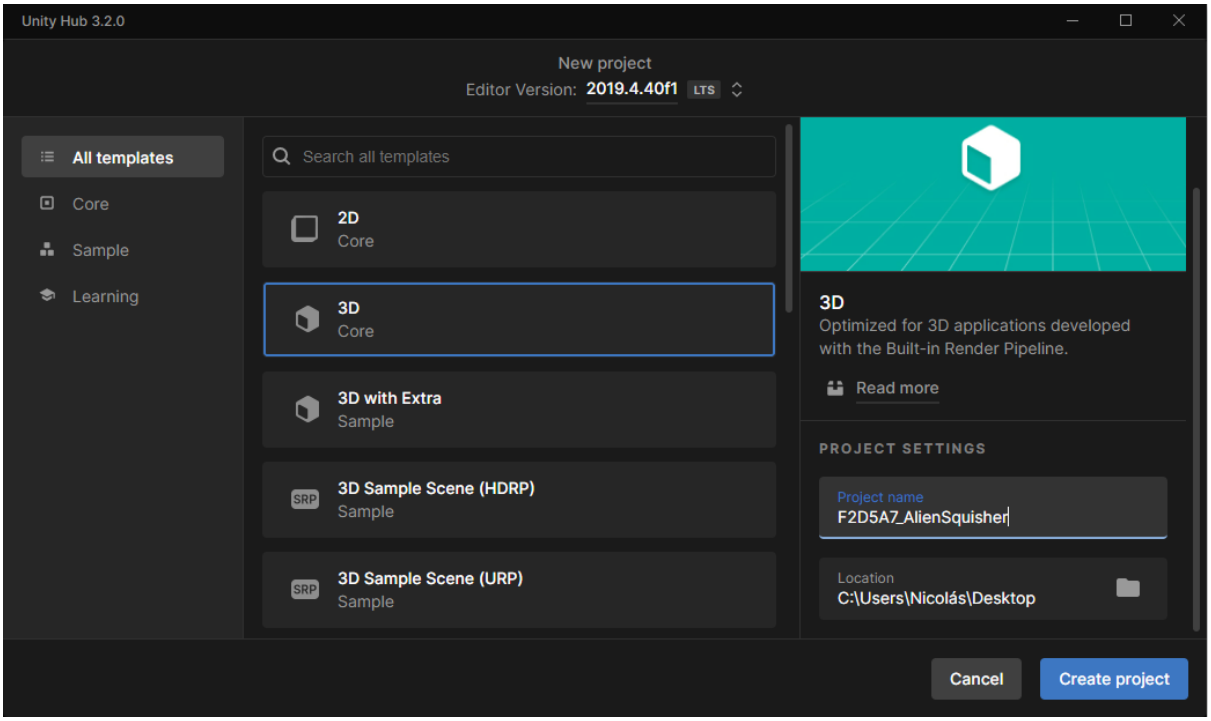


Figura 2.2.1.2.4. Ventana de creación de proyecto con las configuraciones descritas.

2.2.1.3. Interfaz gráfica de Unity

Dando un tiempo para que Unity cargue el proyecto nuevo, nos encontraremos con la interfaz gráfica de Unity, la cual nos permitirá crear el escenario virtual en 3D para nuestro minijuego. A continuación podrá observar dicha interfaz gráfica.

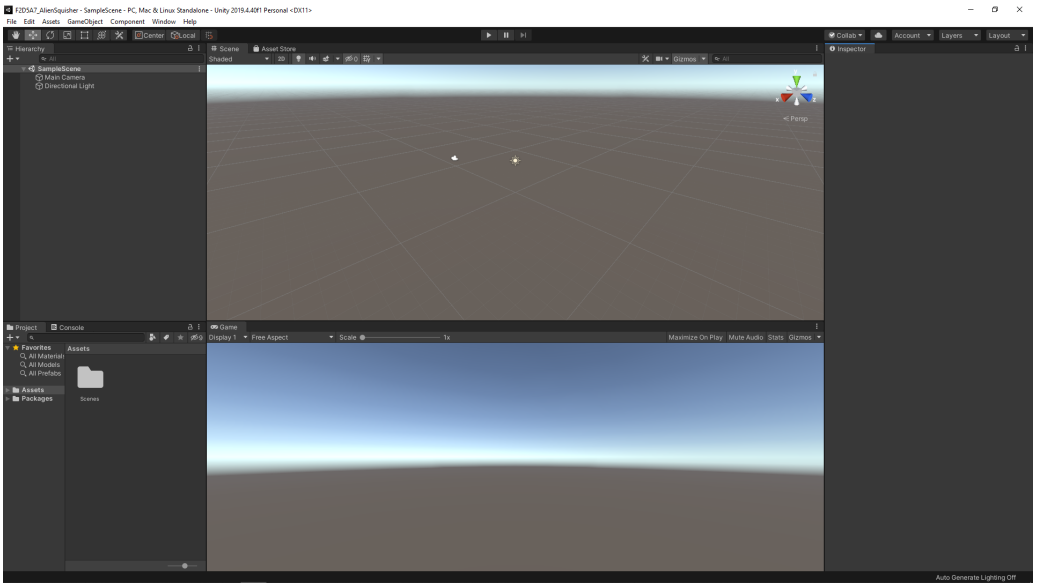


Figura 2.2.1.3.1. Interfaz gráfica de Unity 3D.

2.3 Organización de archivos

Cuando se trata de múltiples minijuegos dentro de un juego global, es necesario seguir las regulaciones sobre el nombramiento de los videojuegos. En este caso, se sigue la regulación de añadir el código del videojuego antes del nombre del mismo, así como otros elementos que puedan repetirse, como scripts. En este caso, Alien Squisher pertenece a la **F**ase **2**, **D**ía **5**, **A**ctividad **7** dentro del conjunto de minijuegos que está siendo desarrollado. Por consiguiente, el código de Alien Squisher es **F2D5A7.** El nombre que utilizaremos en la carpeta principal, ficha técnica y en la escena principal del videojuego será el código del videojuego, separado por un guión bajo del título del videojuego. En este caso, **F2D5A7_AlienSquisher.**

Para crear una carpeta, podemos dar clic derecho en el explorador de archivos de Unity, y seleccionar la opción de crear una carpeta (representada en el menú como *Create Folder*).

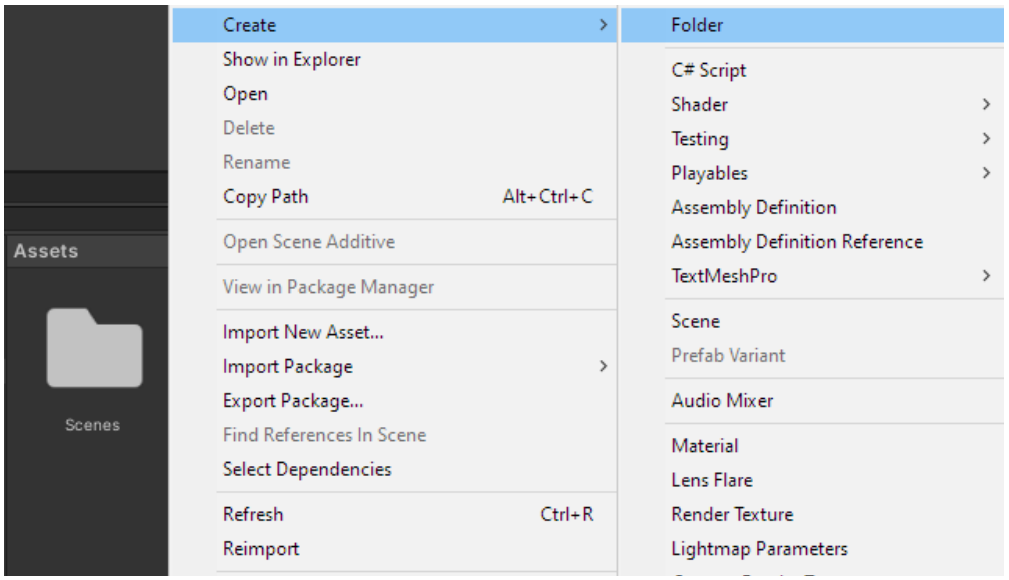


Figura 2.3.1. Creación de una carpeta nueva en Unity

Dado el nombre que definimos para el videojuego y la nomenclatura adecuada, el nombre que llevará la carpeta principal para este minijuego será **F2D5A7_AlienSquisher,** y por defecto debe ir dentro de la carpeta de Assets**:**

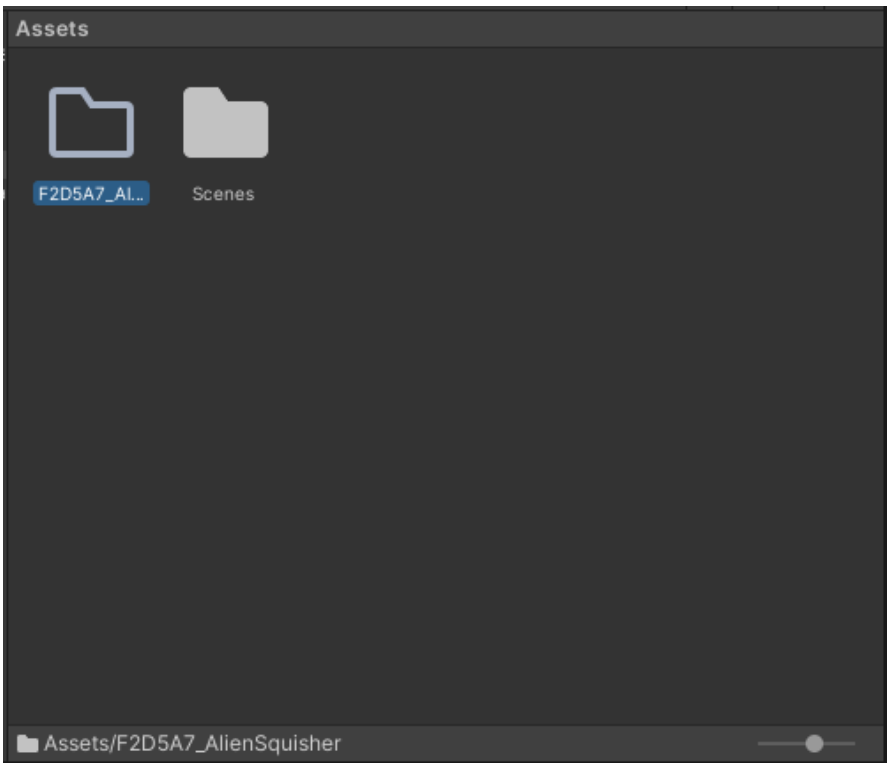


Figura 2.3.2. Carpeta para minijuego Alien Squisher

Dentro de esta carpeta, debe separarse por categorías en inglés cada elemento necesario para realizar el videojuego. Generalmente, los elementos utilizados en la creación de juegos en realidad virtual son: modelos 3D, materiales, prefabricados, importaciones de assets, escenas, scripts, sonidos y texturas. Tendríamos así carpetas llamadas: *Models, Materials, Prefabs, Imports, Scenes, Scripts, Sounds,* y *Textures*. Organizando nuestras carpetas, tendremos esta plantilla para trabajar nuestro juego.

Figura 2.3.2. Organización de la carpeta para el minijuego.

Habiendo organizado nuestro espacio de trabajo, podemos empezar a desarrollar nuestro minijuego, comenzando por crear una escena en la carpeta correspondiente, haciendo clic derecho sobre el navegador de archivos. La escena en la que trabajemos nuestro minijuego debe llevar el mismo nombre de la carpeta del videojuego: El código de nuestro videojuego separado con un guión bajo de su nombre. La escena entonces también llevará el nombre **F2D5A7_AlienSquisher**.

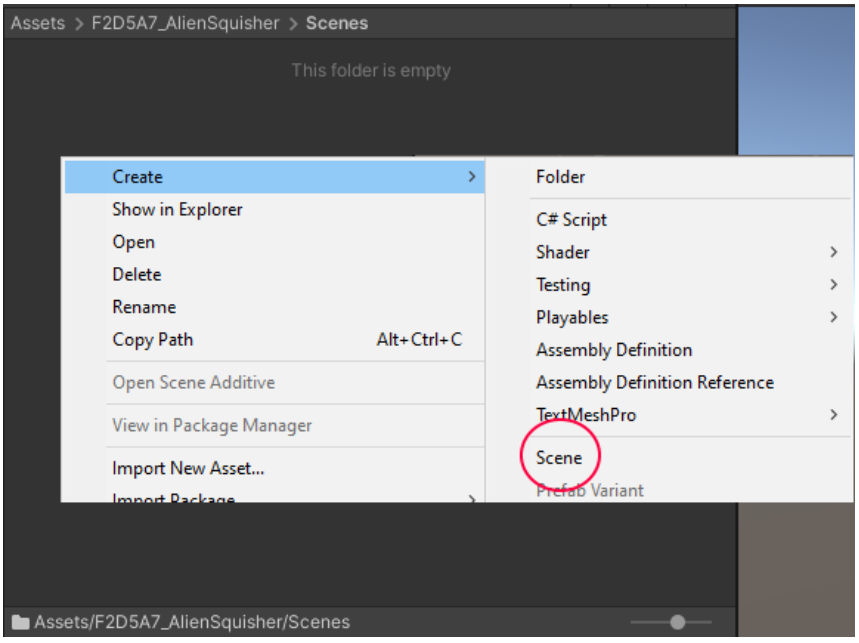


Figura 2.3.3. Menú para la creación de archivos.\

2.4. Navegación en la interfaz gráfica de Unity

Al crear cualquier escena nueva, nos encontraremos con la interfaz gráfica de Unity, que puede ver explicada a continuación. En el costado izquierdo de la interfaz gráfica podremos encontrar un editor de la jerarquía de elementos en la escena (ventana de *Hierarchy,* señalada con el color amarillo) y el sistema de archivos y consola de depuración (ventanas de *Project* y *Console* señaladas con el color magenta). En el centro nos encontraremos con un editor para la escena (*Scene*), una pestaña para la tienda de recursos (*Asset Store*) y una pestaña de previsualización para el minijuego (*Game*, todas señaladas con el color azul). A la derecha se encuentra el inspector (ventana *Inspector,* señalada con el color amarillo), en el cual podremos modificar las propiedades de elementos en el videojuego. Finalmente en la parte superior se pueden encontrar la barra de herramientas (señalada con el color magenta) donde podemos encontrar opciones de navegación, ejecución y colaboración en editor, y el menú (señalado en azul) donde podemos ver otras opciones para guardar archivos, cambiar la configuración o acceder a otros componentes de Unity.

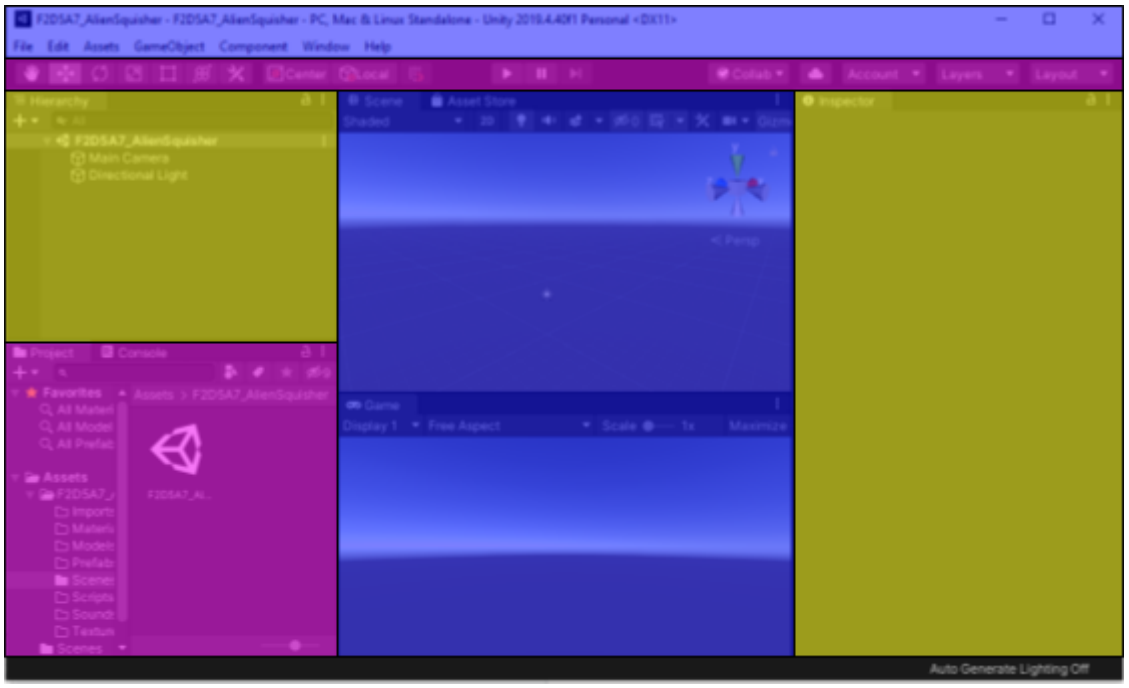


Figura 2.4.1. Interfaz gráfica de Unity.

Si desea, puede cambiar el orden de estas ventanas a su gusto arrastrando las pestañas y soltándolas donde prefiera. Sin embargo, es recomendable mantener un espacio de trabajo ordenado, y preferiblemente mantener el orden predeterminado de Unity.

En especial, es de gran importancia aprender a manejar el editor de la escena, pues es esta pestaña en donde podremos realizar toda la construcción de nuestros

videojuegos. A pesar de que no es necesario, se recomienda el manejo de Unity con un ratón óptico, puesto que la navegación en el editor es similar al control del personaje de un videojuego.

Los controles básicos para moverse en el editor de la escena son como se muestran a continuación.

- Sostener el clic derecho para activar la navegación en la escena
- Con clic derecho sostenido:
    - W,A,S,D, para desplazarse en la escena
        - W para desplazarse hacia adelante
        - A para desplazarse hacia la izquierda
        - S para desplazarse hacia atrás
        - D para desplazarse hacia la derecha
    - Mover el ratón para mover la mirada
- Sostener el clic tercero para desplazarse en la escena sin necesidad de teclado

3. Construcción de la escena

3.1. Crear un elemento 3D básico

Las escenas virtuales están compuestas de elementos 3D geométricos. Estos elementos pueden ser derivados de distintos lugares, el más simple siendo los objetos primitivos que provee Unity. Para entender cómo funcionan estos elementos, crearemos un cubo. Podremos crear objetos en la pestaña de GameObject, en la sección de objetos primitivos en 3D, como puede observarse a continuación.

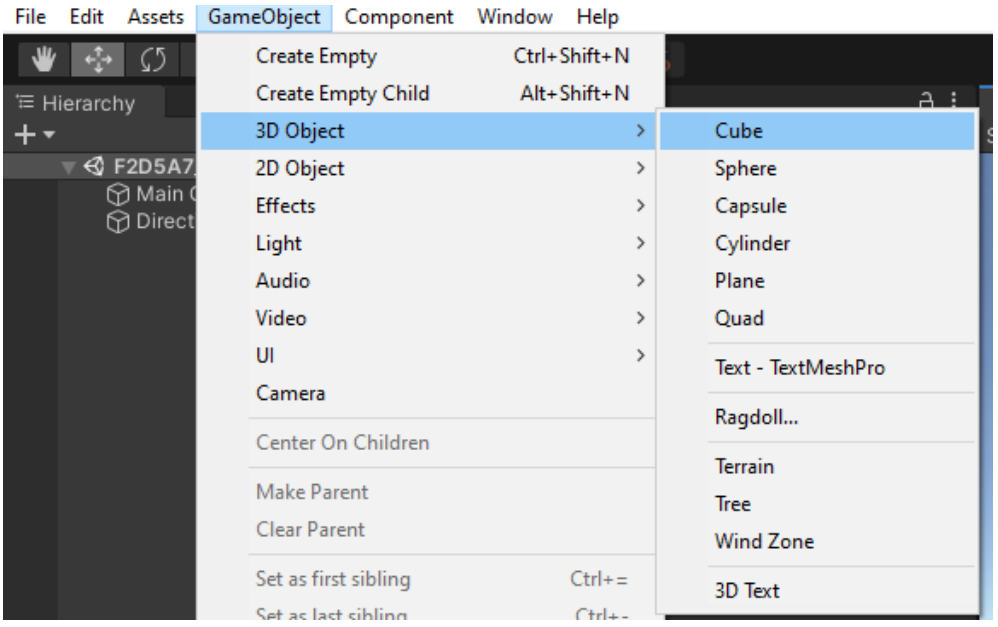


Figura 3.1.1. Creación de un cubo.

Aparecerá un cubo en nuestra escena. Con el cubo en la escena, los siguientes controles son importantes:

- Hacer clic izquierdo en un objeto para seleccionarlo
- Luego de seleccionar un objeto, puede presionar F para centrar la vista en el objeto

Hacer clic izquierdo en el objeto ocasionará que el inspector ahora nos muestre las propiedades del cubo, además de mostrarnos 3 flechas representando cada vector de posición del objeto. Cabe mencionar ciertas propiedades, como los vectores de transformación. Estos vectores podrán ser modificados para cambiar la posición, rotación y tamaño de un objeto. Podremos además ver el colisionador físico del objeto y el material usado, los cuales serán importantes para el desarrollo del videojuego más adelante.

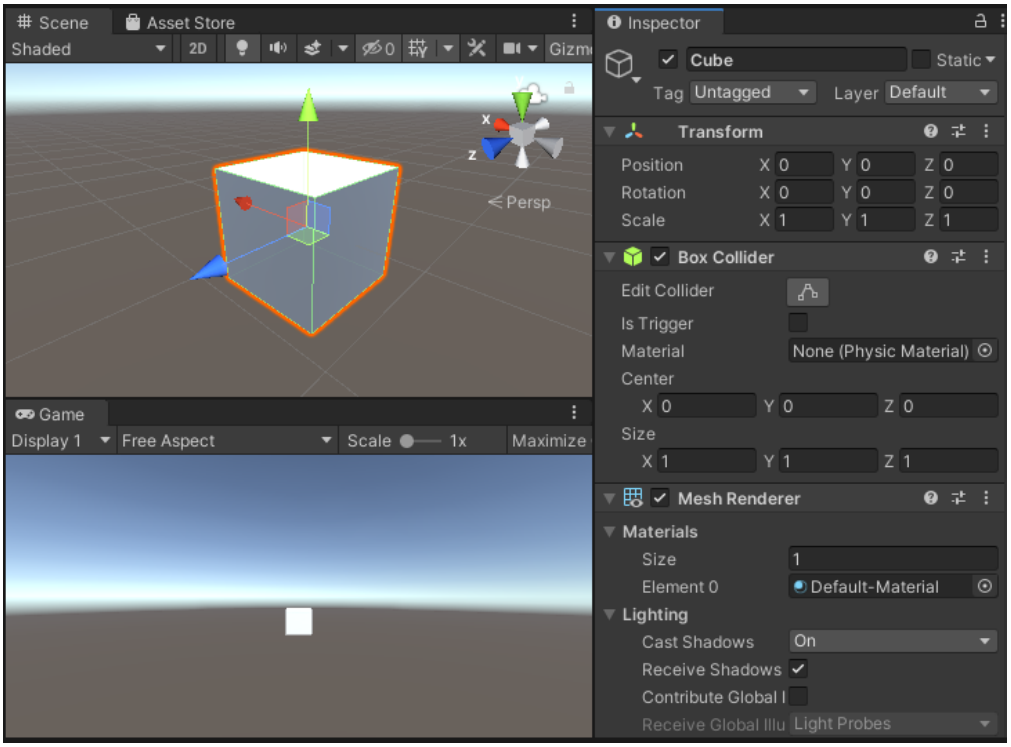


Figura 3.1.2. Inspector con cubo seleccionado.

Podemos ahora experimentar mover los vectores de transformación, o utilizar las herramientas de Unity para rotar, mover o cambiar el tamaño del cubo. Estas opciones se encuentran en la barra de herramientas de Unity, como se pudo ver en el capítulo anterior.

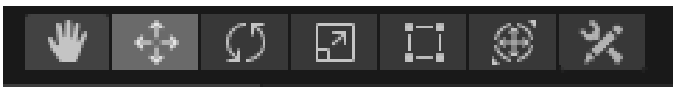


Figura 3.1.3. Herramientas para transformación de objetos en Unity.

En orden de aparición, estas herramientas permiten:

- Desplazarse en la escena 3D (representado con el ícono de mano)
- Desplazar un objeto en la escena 3D (representado con el ícono de la cruceta)

- Rotar un objeto en 3D (representado con el ícono de las dos flechas formando un círculo)
- Cambiar el tamaño de un objeto en 3D alterando sus dimensiones (representado con el ícono de dos cuadrados de diferentes tamaños, con una flecha en medio de ellos)
- Cambiar el tamaño de un objeto en 3D alterando el espacio que ocupa (representado con el ícono de la caja)
- Alterar cualquier propiedad del objeto 3D simultáneamente (ícono de la cruceta encerrada por un círculo con dos flechas en diagonal)
- Utilizar herramientas específicas de cada elemento, como modificar el colisionador en el cubo, entre otras (representada con el ícono de una llave inglesa y un lápiz)

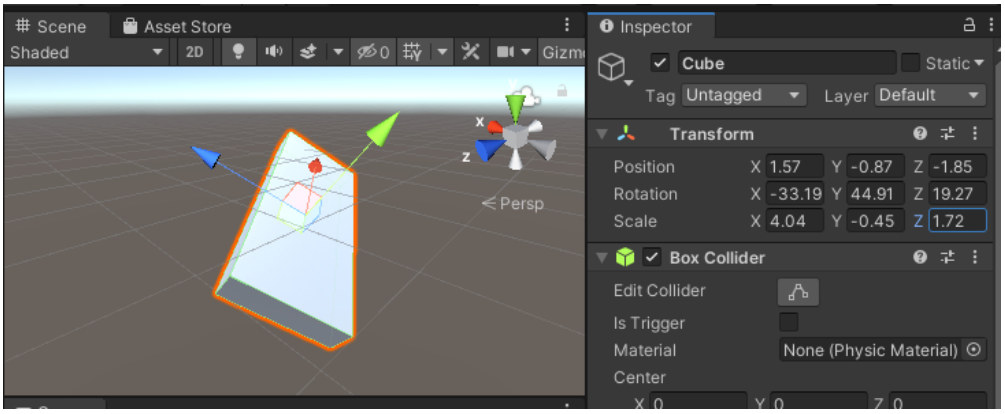


Figura 3.1.4. Cubo luego de haber sido reposicionado, rotado, y escalado en sus 3 distintos ejes.

3.2. Uso de materiales

Luego de haber creado el elemento básico, podemos utilizar materiales para darle cualquier aspecto. Los materiales nos permiten cambiar la manera en la que un objeto se ve, para asimilar materiales en la vida real como metal, plástico, madera, etc. Existen infinitos tipos de materiales que podemos crear para que se ajusten a lo que necesitemos visualmente. Para crear un material, navegaremos a la carpeta de materiales, dándole clic izquierdo en el explorador de archivos.

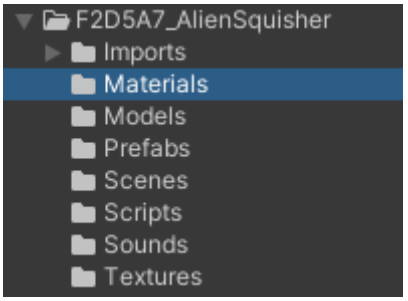


Figura 3.2.2. Carpeta de materiales en el navegador de archivos.

Y crearemos ahí un nuevo material. Como nombre, le pondremos “Acero”.

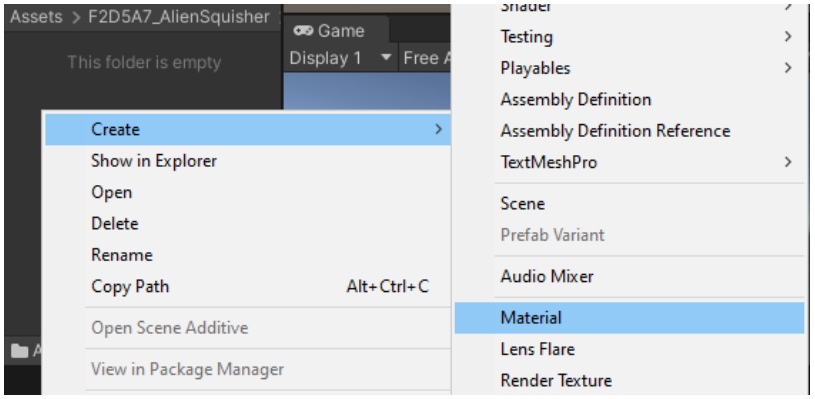


Figura 3.2.2. Creación de un material nuevo.

Luego de crear el material, el inspector abrirá las propiedades de dicho material. Es aquí donde modificaremos la apariencia de nuestro objeto. Como su nombre lo indica, los materiales representan aspectos para distintos materiales del mundo real. Por ende, es importante pensar en el material del objeto que estamos recreando. En este caso, estamos creando el acero. Por ende, ajustaremos un color (*Albedo*) grisáceo, pondremos la propiedad de metálico en 1 (En la vida real, los materiales sólo pueden ser o no ser metálicos. Por ende, se recomienda trabajar la propiedad de metálico como 0 o 1 únicamente), y ajustaremos la suavidad a 0.66. Esta suavidad representa qué tanto del exterior refleja este material y se puede encontrar en Unity bajo el nombre *Smoothness*. Una suavidad en 0 sólo refleja su propio color, mientras que una suavidad en 1 refleja una mezcla del exterior con el color del material. Los otros valores no son necesarios para mostrar muchos objetos opacos, pero es recomendable experimentar con ellos para observar qué apariencia producen.

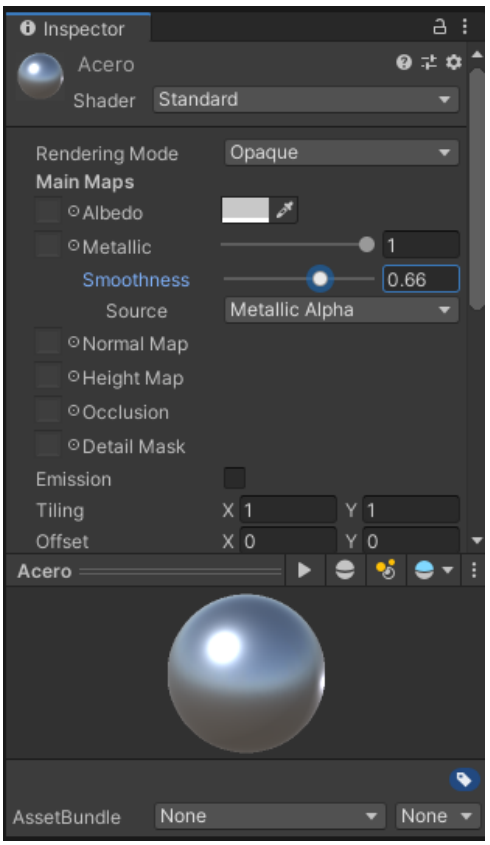


Figura 3.2.3. Propiedades para el material de acero.

Para lograr aspectos de materiales no opacos, como puede serlo una luz encendida, un vidrio, o el agua, debemos cambiar el *Shader* utilizado, así como el modo de renderizado (que se encuentra como *Rendering Mode* en las propiedades del material). Los *Shaders* son algoritmos que permiten modificar la manera en la que los materiales funcionan. Sin entrar en mucho detalle, pueden explicarse como una plantilla para cada material, sobre la cual podemos añadir la apariencia deseada. A continuación, podrá ver varios ejemplos de materiales. De nuevo, se recomienda experimentar con estos ejemplos para lograr la variedad de tener diferentes objetos en la escena virtual. En estos ejemplos se cambia el *Shader* utilizado, así como las distintas propiedades de cada uno para obtener materiales similares a los que se pueden encontrar en la vida real.

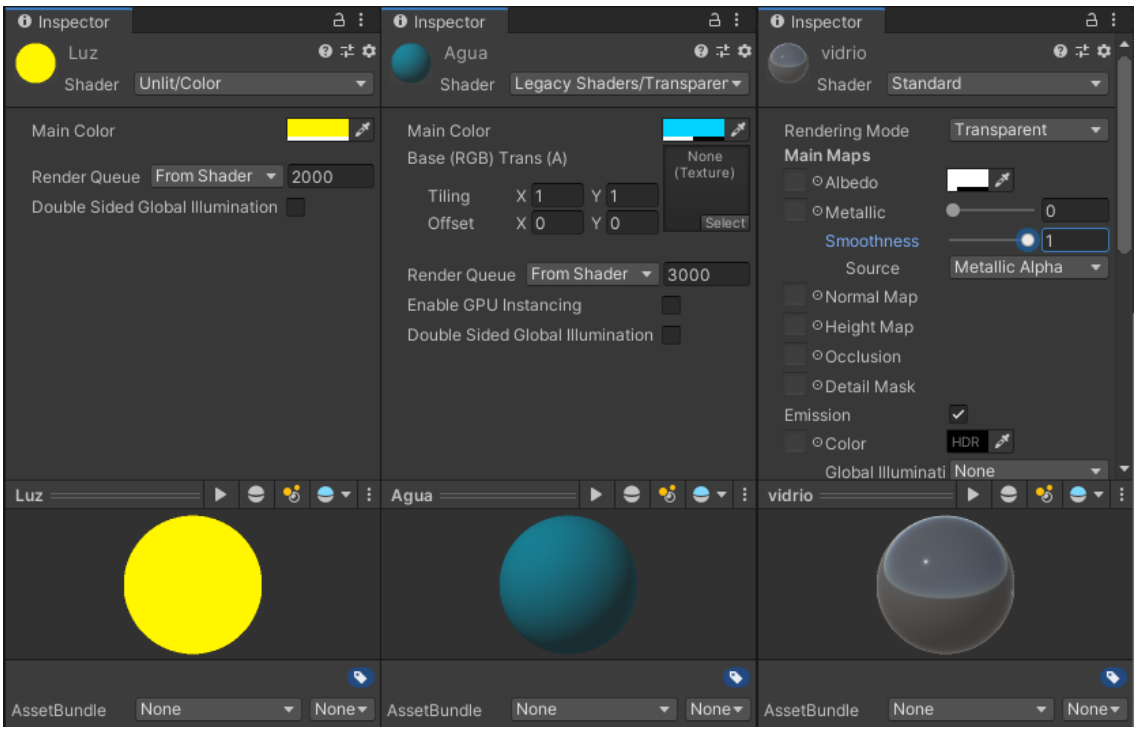


Figura 3.2.4. Ejemplos de diferentes materiales.

Para finalizar, podemos arrastrar el material de nuestra elección desde el explorador de archivos, hasta el objeto que queramos aplicar dicho material en el editor de la escena. Ahora podremos ver el objeto con la apariencia del material que hemos creado.

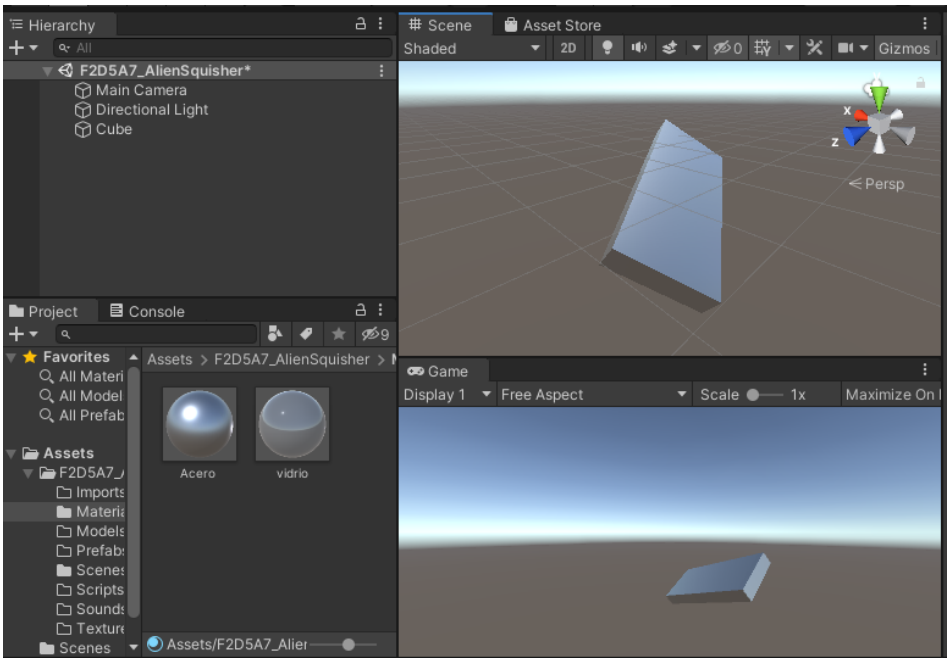


Figura 3.2.5. Cubo con material de acero aplicado.

3.3. Creación de terreno

Ahora, crearemos un terreno, debido a que es un elemento que todos los videojuegos deben contener. Este elemento servirá como un suelo, el cual el jugador podrá ver al mirar hacia abajo, evadiendo cualquier sentido de vértigo. Este elemento podrá encontrarse en el menú de crear objetos, en la sección de objetos primitivos, como se evidenció anteriormente con la creación del cubo. Se puede encontrar como *Terrain*.

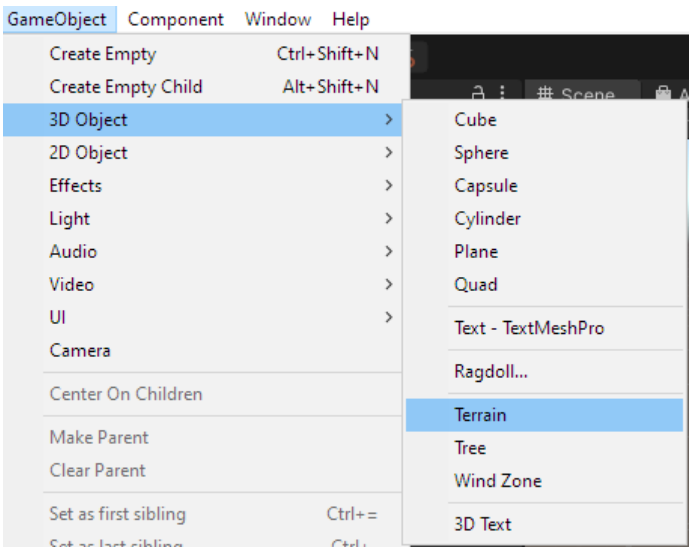


Figura 3.3.1. Opción para crear terreno.

Esto creará un plano que podremos modificar a nuestro antojo para hacer planicies, montañas, etc. Sin embargo, lo principal es transportar el terreno, pues este es generado con una esquina sobre el centro de nuestra escena 3D. Esto significa que será difícil crear toda una escena que tenga una cantidad proporcional de suelo en todas las direcciones. Lo recomendado es cambiar el vector de posición a (-500, 0, -500). Esto se debe a que el terreno por defecto mide 1000 x 1000 unidades de Unity, y -500 lo ubicará justo en la mitad de la escena.

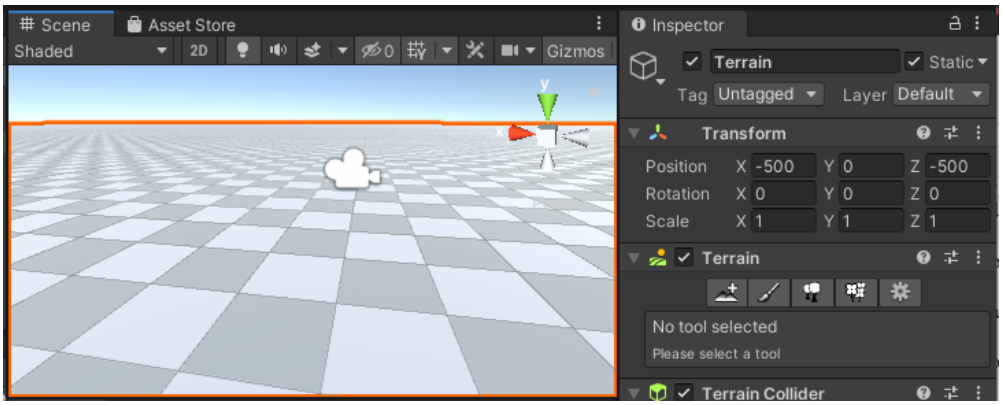


Figura 3.3.1. Terreno ubicado en la escena.

Adicionalmente, se creará un archivo adicional en nuestro proyecto, conteniendo nuestro nuevo terreno. Para distinguirlo de los demás terrenos podemos renombrarlo y

enviarlo a la carpeta de modelos en nuestro videojuego. Por simplicidad, el código de nuestro videojuego y la palabra ‘Terrain’ debería ser suficiente para distinguirlo. Puede cambiar la ubicación del terreno arrastrando el archivo hacia la carpeta de modelos.

Figura 3.3.2. Terreno ubicado en la nueva carpeta.

Luego de tener un terreno, podemos modificarlo a nuestro antojo. Por ejemplo, podemos pintar césped para que parezca un campo en lugar de la cuadrícula por defecto. Para hacer esto, deberemos hacer clic en la herramienta de pincel, que se encuentra en la sección de terreno en el inspector. Luego de esto, cambiaremos la primera opción a “Pintar textura”.

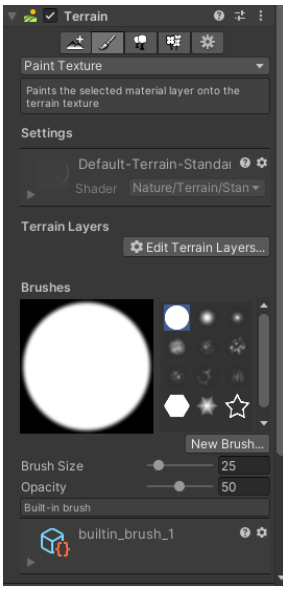


Figura 3.3.3. Pintar textura en terreno

Ahora, es necesario buscar una textura para aplicar a nuestro terreno para poder aplicar como la primera capa. Opcionalmente, también podemos buscar una adicional para mezclar ambas en el terreno. Revise la sección de recursos para encontrar lugares que proveen texturas, incluído texturas de césped para aplicar. En este caso, se utilizarán dos diferentes texturas de la página **Polyhaven**. Las texturas que se utilizarán son “*Brown Mud Leaves 01*” y “*Forest Ground 01*”. Se pueden utilizar en formato jpeg, jpg o png.

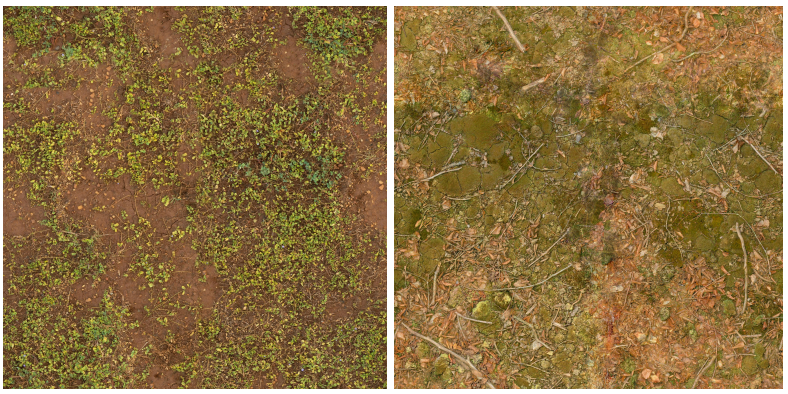


Figura 3.3.4. Texturas de césped utilizadas. Fuente: https://polyhaven.com/

El siguiente paso es copiar estas dos texturas a nuestra carpeta de texturas dentro del proyecto, y crear una capa base para la textura, haciendo clic en el botón de “Editar capas de terreno”. Esto abrirá una pestaña donde podremos elegir nuestra textura base.

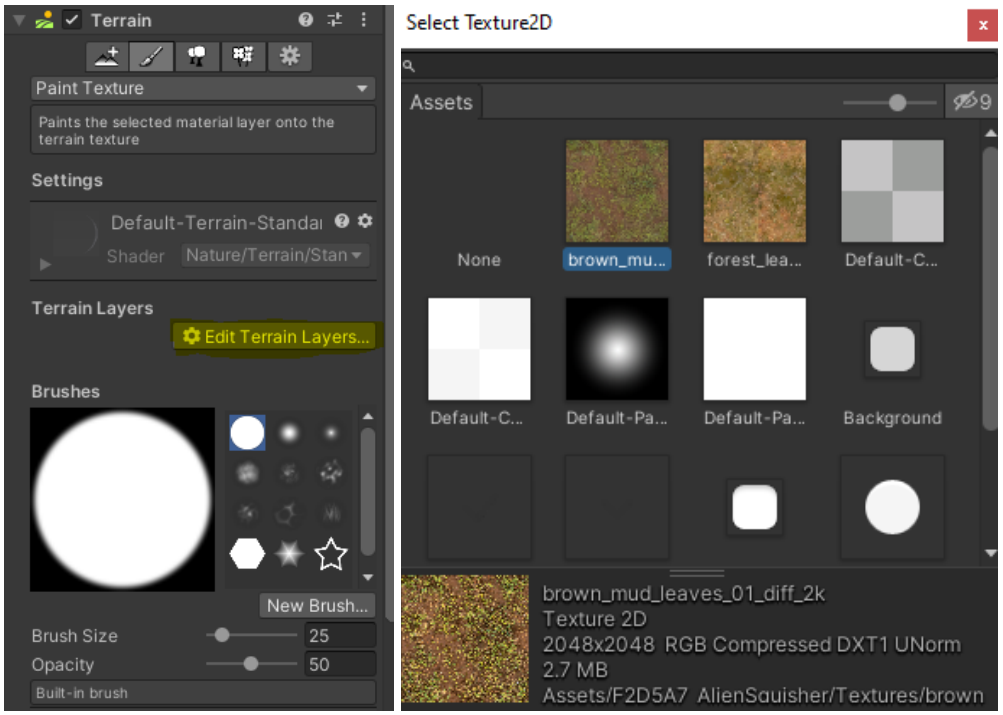


Figura 3.3.5. Crear capa de terreno.

Esto aplicará la capa de terreno a todo nuestro objeto de terreno. Además, similar al uso de texturas, podremos modificar algunas propiedades de la capa. Por ejemplo, podemos modificar la propiedad de tiling para cambiar el tamaño de la textura en el terreno o las propiedades del material. Como utilizamos texturas de alta resolución, las haremos ver más grandes en el terreno.

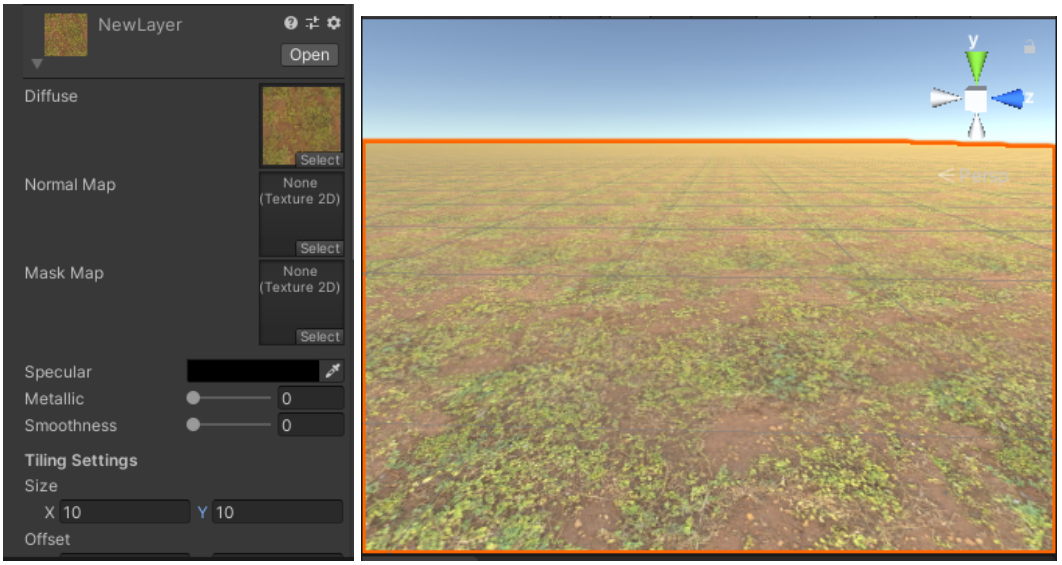


Figura 3.3.6. Aplicar primera capa de terreno

Como se mencionó anteriormente, podemos agregar más capas de texturas para romper el patrón en el terreno. Así podemos solucionar el problema que viene con utilizar texturas muy pequeñas para un terreno muy grande, y que generalmente producirán repetición. Para ello, repetiremos el mismo paso de crear una capa nueva de terreno, la cual podremos modificar nuevamente a nuestra conveniencia, y aplicar encima de la existente. Para hacer esto, deberemos ir al editor de la escena y ‘pintar’ la textura sobre el terreno que tenemos. Esto será más fácil con las opciones de terreno en el inspector.

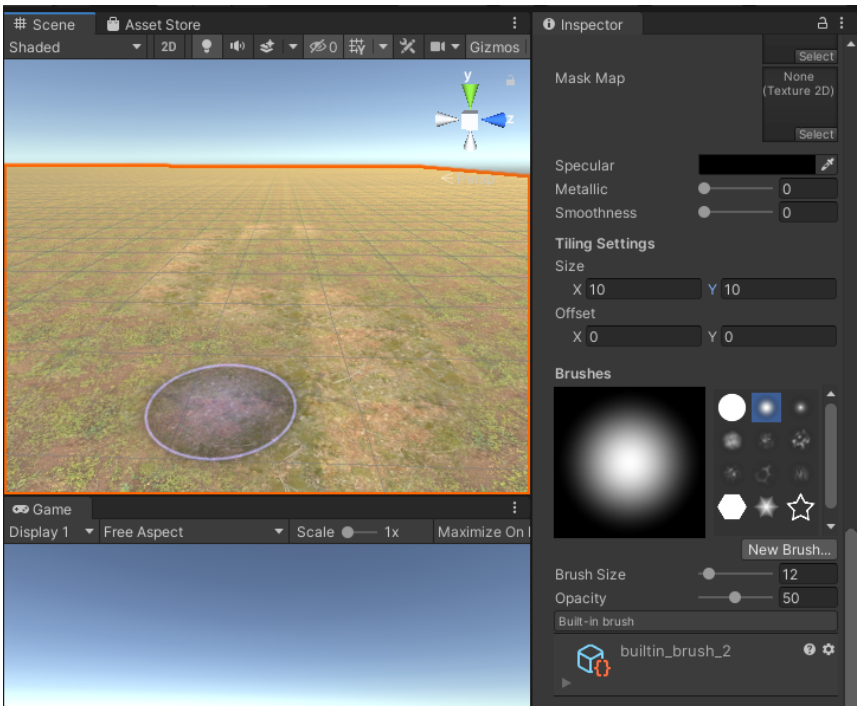


Figura 3.3.7. Aplicar segunda capa de terreno sobre la existente.

Finalmente, podemos utilizar la misma herramienta de pintar terreno para agregar profundidad. Esto podemos hacerlo con la misma herramienta de pintar terreno, seleccionando la opción de “Agregar o reducir terreno”

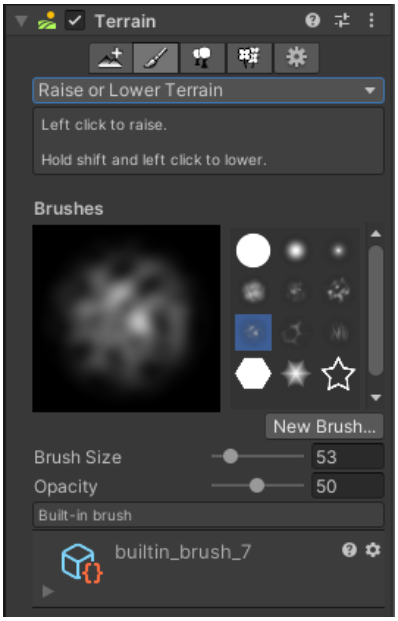


Figura 3.3.8. Herramienta para agregar o reducir terreno.

Como en el paso anterior, entraremos ahora al editor de la escena para crear montañas y planicies. Es importante considerar que al estar realizando un juego de realidad virtual, es bueno dejar un área circular o cuadrada alrededor del jugador, donde no haya modificaciones en el terreno. También es necesario mencionar que sólo hace falta añadir cambios en el terreno alrededor del jugador, pues no podrá ver más allá de las colinas.

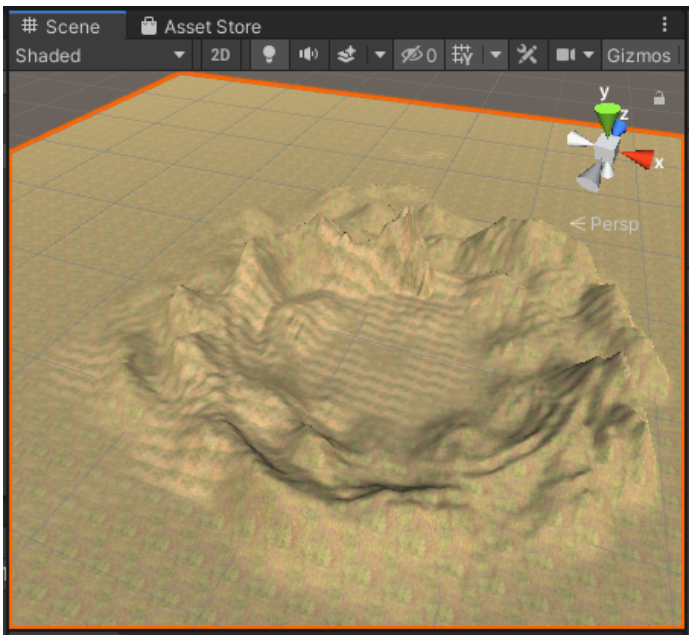


Figura 3.3.9. Terreno modificado con una planicie en el centro.

Es importante aclarar que este tipo de terrenos no será el ideal para utilizar en nuestros videojuegos, debido a que utiliza texturas y tiene alta resolución, por lo que no cumple de acuerdo con los requerimientos para low poly. En la siguiente sección se podrá ver cómo descargar recursos low poly. Se recomienda usar uno de los terrenos incluídos en estos recursos.

3.4. Importación de recursos externos

A continuación podrá ver distintas maneras de importar recursos externos a Unity, cómo seleccionarlos, cómo modificarlos, y cómo utilizarlos en su escena.

3.4.1. Criterios para el uso de recursos

Debido al costo computacional de realizar juegos en realidad virtual, y a la necesidad para producir múltiples escenas en corto tiempo, es altamente recomendado buscar que el estilo artístico de todos los minijuegos sea similar. Por ello, se busca un estilo Low-Poly, o de bajo número de polígonos. Esto permite crear assets nuevos con facilidad, así como reducir el costo computacional de las escenas, al no requerir modelos con mucha complejidad.

Esto significa que al buscar recursos, debemos tener en cuenta que los modelos 3D sean Low-Poly. Es decir, minimalistas y generalmente sin texturas de alta resolución.

Otro criterio para los archivos es el formato, ya que Unity no soporta todos los formatos de archivos naturalmente para mantener consistencia en las escenas. A continuación presento una tabla de los formatos de archivo recomendados para cada tipo de archivo.

Tipo de archivo	**Formatos recomendados para Unity**
Materiales	.mat, .mtl
Modelos 3D	.fbx, .obj
Texturas	.jpg, .png, .tga
Scripts	.cs
Sonido	.wav, .mp3, .ogg

Es necesario además seleccionar sólo los elementos necesarios para utilizar en nuestro videojuego. Por ejemplo, si debemos descargar un paquete con elementos de

bosques para introducir un árbol a nuestro videojuego, dejaremos sólo el árbol junto con sus dependencias dentro de nuestra carpeta.

3.4.2. Importar de la tienda de recursos de Unity

La manera más sencilla de importar algo necesario para nuestro videojuego es buscarlo en la tienda de Unity. Por ejemplo, podemos añadir una granja para nuestro videojuego, ya que este será el lugar donde los aliens atacarán (ver ficha técnica). Para esto, deberemos acceder a la tienda de assets, ubicada en la pestaña siguiente a nuestro editor de la escena. Luego ingresamos en el buscador el nombre de lo que estemos buscando. Si se trata de un recurso en 3D, es importante agregar la palabra clave Low-Poly.

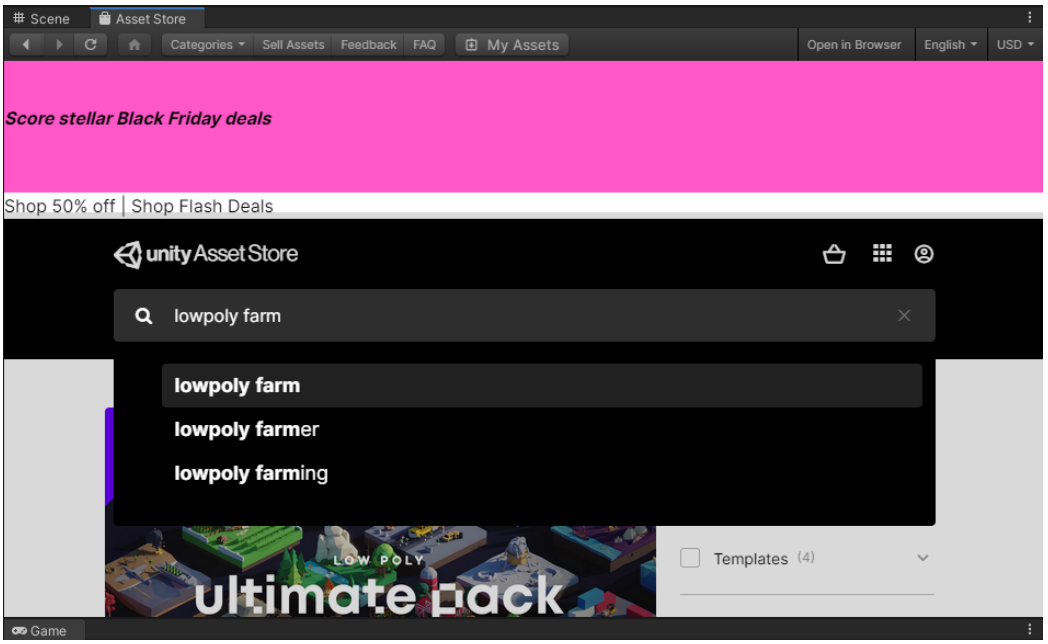


Figura 3.4.2.1. Tienda de assets.

Luego de recibir los resultados, es importante filtrar por el tipo de consulta y el rango de precios que queremos pagar. En este caso, la categoría es 3D y buscamos recursos gratis. Podemos filtrar en el menú de la derecha.

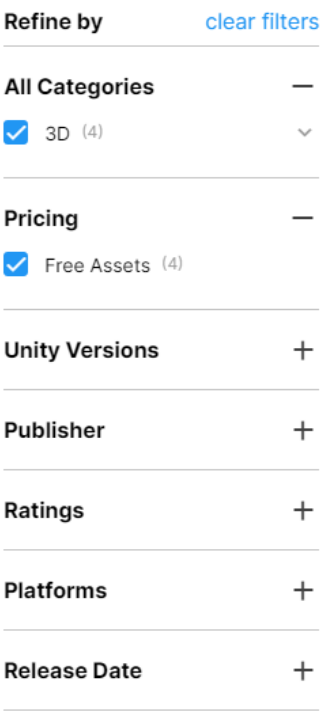


Figura 3.4.2.2. Filtros en la tienda de Unity.

Ahora sólo debemos escoger un recurso de nuestro agrado, y que contenga exactamente lo que necesitamos para nuestro videojuego. En este caso, se escogió el recurso “Low Poly Farm Pack Lite” de *JustCreate*. Para instalarlo, sólo es necesario entrar a la página del recurso, y hacer clic en “descargar”. Seguido esto, se hará clic en el mismo botón para importar el recurso al videojuego, y seleccionar únicamente los elementos que deseemos.

*Para obtener más recursos de la tienda de recursos de Unity o de demás tiendas de recursos, se recomienda buscar los assets tanto en inglés como en español.

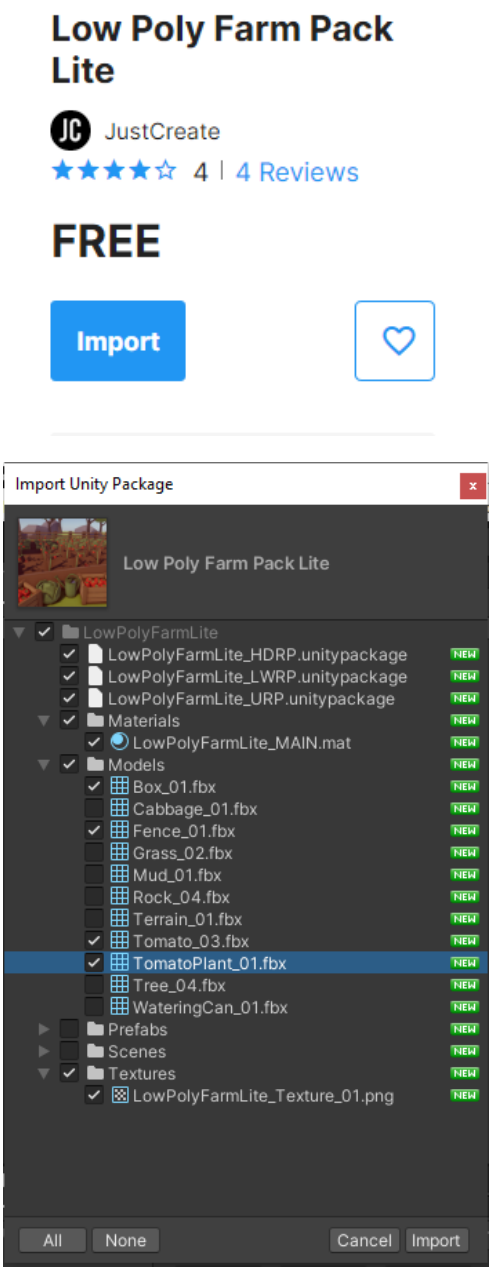


Figura 3.4.2.3. Instalación de recursos desde la tienda de Unity.

Por último moveremos la carpeta generada por el recurso a nuestra carpeta de Imports, y añadiremos lo necesario a la escena. En este caso, podremos arrastrar los modelos 3D deseados a la escena como hicimos con el cubo anteriormente, y también podremos aplicarles sus respectivos materiales.

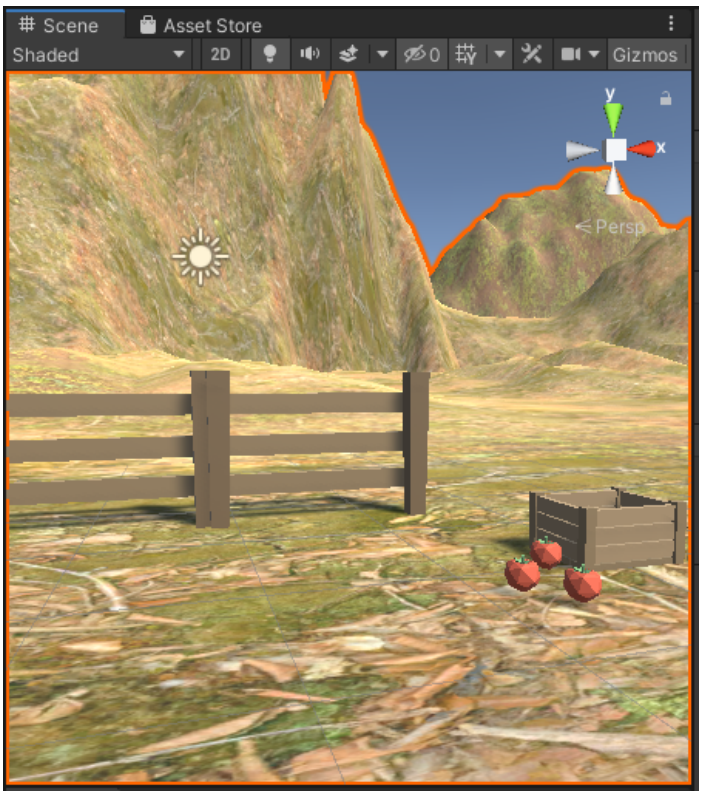


Figura 3.4.2.4. Adición de los recursos de la tienda de Unity a la escena.

3.4.3. Instalación de recursos externos a Unity

Existe la posibilidad de que el recurso necesario para nuestro videojuego no se encuentre en la tienda de recursos de Unity, ya sea por la rareza del modelo 3D o por precios muy altos dentro de la tienda. La solución es buscar estos recursos en páginas alternas de confianza. Para esto se recomienda de nuevo revisar la sección de recursos. Estas páginas requieren iniciar sesión y a cambio nos entregarán un modelo en diferentes formatos. Es importante también revisar la tabla de formatos recomendados.

Antes de descargar algún recurso de una página externa a la tienda de recursos, es necesario recordar revisar la licencia del recurso, pues no todos nos permitirán usarlo en proyectos comerciales, o pedirán que se den créditos al autor para poder utilizar en nuestro proyecto. Además, es necesario revisar el precio del recurso, pues algunas páginas son completamente gratuitas (ej. Polyhaven), mientras otras nos dan permiso a sólo una cantidad mensual de recursos gratis (ej. Poliigon). Otras, en cambio, cobran por unos recursos mientras otros son gratuitos (ej. Turbosquid).

En este caso, trataremos de buscar un alien para nuestro videojuego en la página Turbosquid. Como no se encuentra ningún alien de estilo Low-Poly en la página, se puede modificar un modelo existente para cumplir con este estilo. La guía para convertir cualquier modelo en uno Low-Poly puede encontrarse en este documento. Para este caso en específico se descargó “Spider Monster1 Model” de *konared*. Una recomendación artística es aceptar algunos recursos que puedan ser recontextualizados. Como en este

caso, que se tomó un modelo de un monstruo genérico para utilizar como modelo de alienígena.

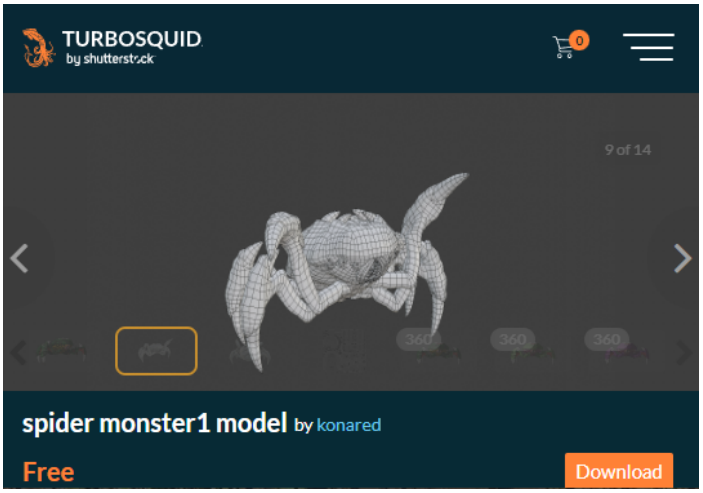


Figura 3.4.3.1. Recurso de Turbosquid.

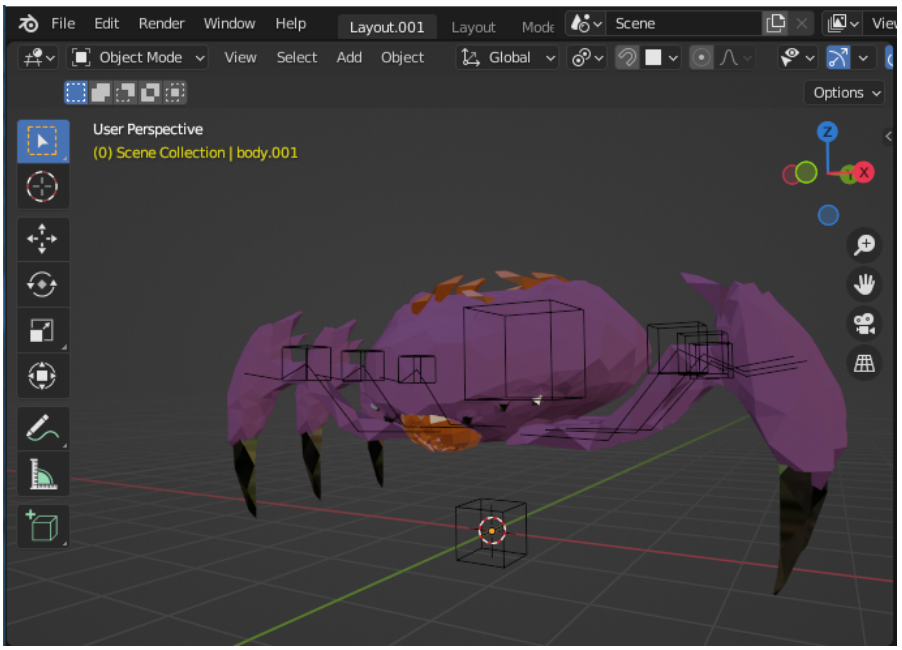


Figura 3.4.3.2. Versión modificada en Blender.

Ahora sólo debemos asegurarnos de tener nuestro recurso en el formato correcto, e importarlo a nuestra carpeta dedicada dentro del minijuego, para poder arrastrarlo a la escena.

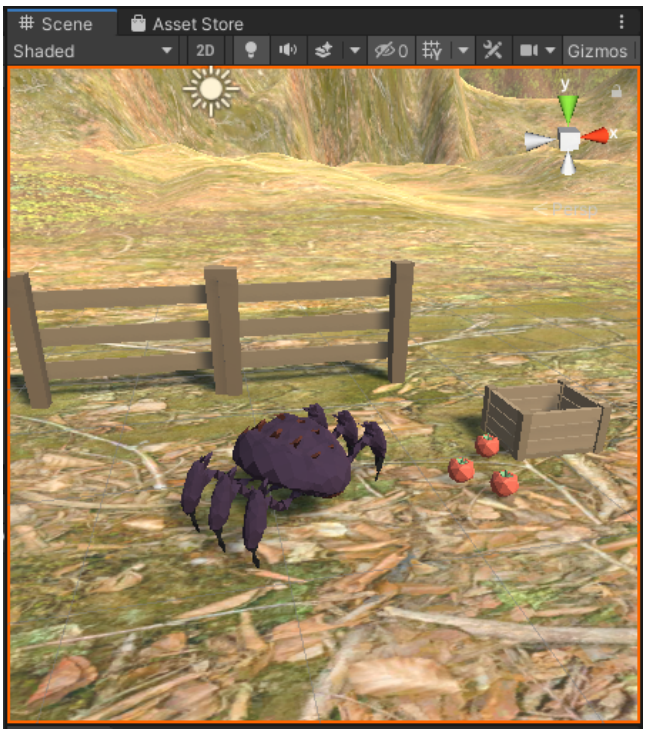


Figura 3.4.3.3. Recurso importado en la escena.

3.4.4. Creación de recursos nuevos

Esta alternativa es opcional. Sin embargo, si posee el tiempo y el conocimiento para realizar sus propios recursos, esta es una manera de que nuestra escena esté personalizada a nuestra propia visión creativa. Además, si el recurso que buscamos es demasiado escaso, puede presentarse el caso de que no lo encontremos ni en la tienda de recursos de Unity, ni en tiendas de recursos externas. Si este es el caso, podemos hacerlos utilizando software que nos facilite la creación de recursos. A continuación una lista de aplicaciones gratuitas que facilitan el proceso:

- Blender - Creación de recursos en 3D como modelos, o materiales
- Krita - Creación de recursos en 2D como texturas o dibujos
- Audacity - Creación de recursos de audio

En este caso, se utilizará Blender para crear una nave espacial para acompañar al alienígena. El software utilizado es irrelevante, mientras podamos exportar nuestro resultado final a un formato aceptado por Unity sin problemas de compatibilidad. Software como AutoCAD también sirve para la generación de archivos 3D compatibles con Unity.

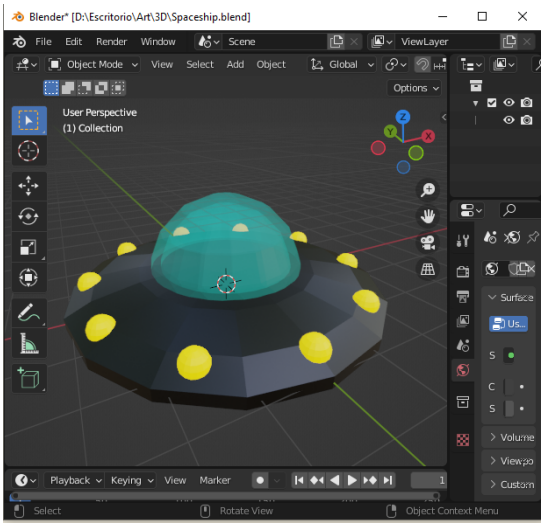


Figura 3.4.4.1. Recurso generado con Blender

Luego de exportar el recurso a un formato aceptado por Unity como fbx, como con los demás recursos, ahora sólo lo debemos arrastrar a nuestra carpeta de recursos y a nuestra escena.

Figura 3.4.4.2. Importación del recurso nuevo en la escena.

4. Jugabilidad

Esta sección está dedicada a todos los elementos necesarios para desarrollar la parte funcional de nuestro videojuego, desde la creación de scripts para lograr distintos comportamientos hasta compatibilidad e integración con los objetivos fisioterapéuticos propuestos.

4.1. Bases para desarrollo de la jugabilidad en Realidad Virtual

4.1.1. Instalación de SteamVR en el proyecto

Debido a que esta guía es de la realización de videojuegos en realidad virtual, es necesario instalar la librería que nos permite probar y jugar nuestros juegos directamente sobre la realidad virtual.

La instalación es como la de cualquier recurso. Debemos buscar SteamVR en la tienda de recursos de Unity e instalarlo. A diferencia de cualquier otro recurso, no debemos copiar la carpeta generada a la dedicada dentro de nuestro proyecto. Tampoco vamos a descartar ninguno de los archivos que vienen este recurso. Cuando la instalación termine, nos preguntará si deseamos utilizar el plugin de Legacy VR o el de Unity XR. Debemos elegir el de Legacy VR para lograr la compatibilidad con los demás minijuegos.

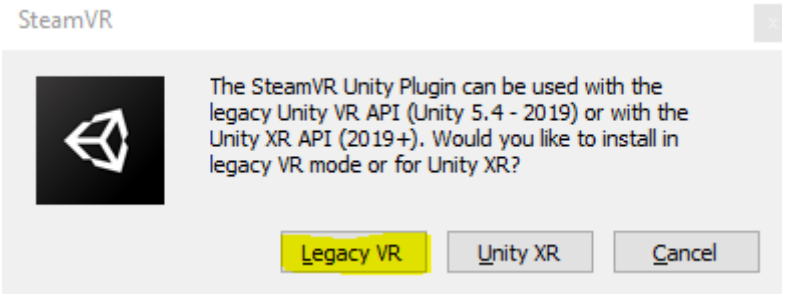


Figura 4.1.1.1. Opciones de instalación de SteamVR, Tipo de plugin.

Luego nos preguntará si deseamos activar el modo de realidad virtual, a lo que seleccionaremos que sí, puesto que este nos permitirá realizar pruebas para nuestro juego con las gafas de realidad virtual en tiempo real.

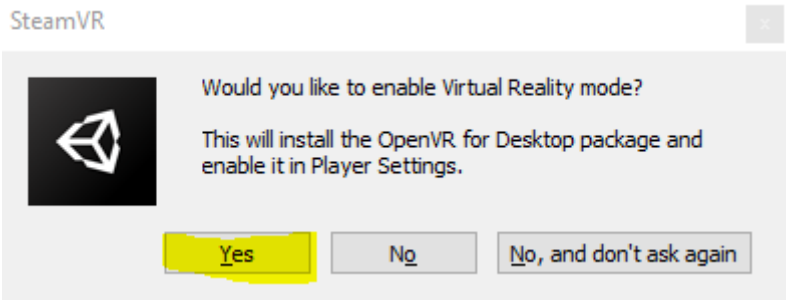


Figura 4.1.1.2. Opciones de instalación de SteamVR, instalar modo de Realidad Virtual

Por último, Steam sugerirá un par de configuraciones a cambiar en el editor. Las aceptaremos también, ya que son configuraciones visuales que harán la experiencia en realidad virtual más agradable a la vista.

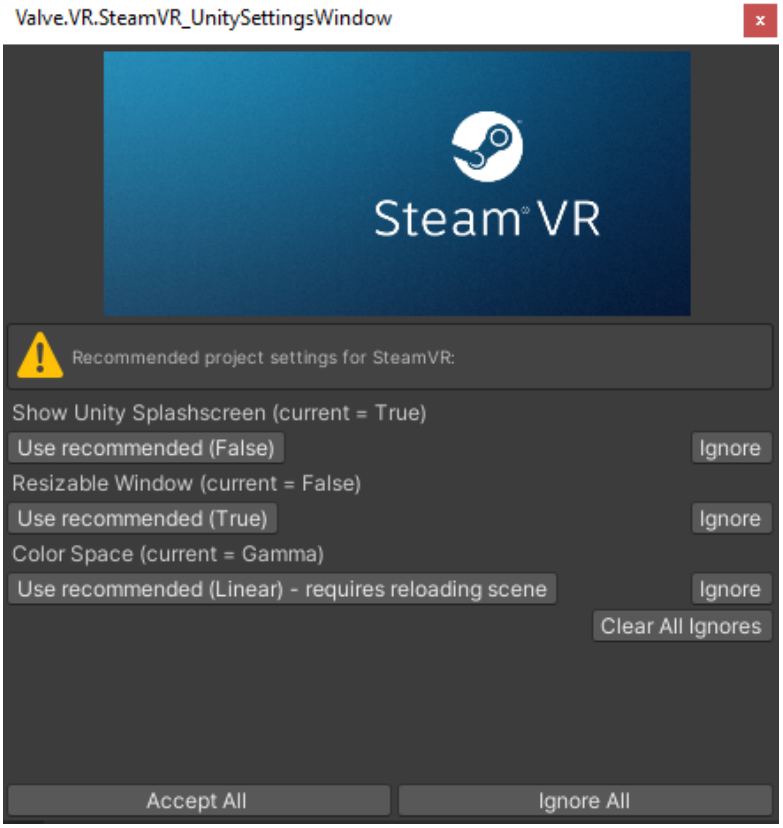


Figura 4.1.1.3. Opciones de instalación de SteamVR, instalar modo de Realidad Virtual

4.1.2. Uso de SteamVR

Con SteamVR instalado, podemos probarlo al incorporarlo a nuestra escena. Para hacer esto, debemos entrar a la carpeta de SteamVR generada en nuestro proyecto, y buscar el elemento ‘Player’ en el navegador de archivos. Este elemento es el que servirá como la versión simulada del usuario en la escena.

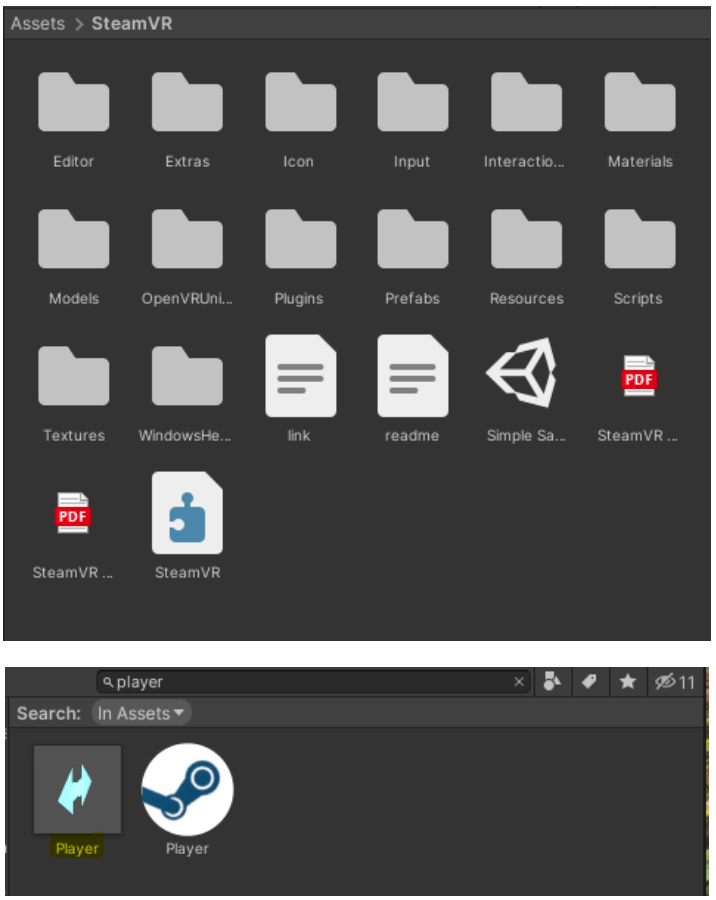


Figura 4.1.2.1. Búsqueda del elemento player en la carpeta de SteamVR.

Luego de haber arrastrado el elemento 'Player' a la escena podemos ubicarlo en la posición ideal para nuestro videojuego.

***Nota: Es importante no cambiar el tamaño o la posición en 'Y' del jugador. Si 'Player' no está bien calibrado con la escena, es mejor modificar los demás elementos. Esto será aparente al momento de probar el juego con realidad virtual.**

Figura 4.1.2.2. Elemento 'Player' ubicado en la escena.

Antes de poder probar el jugador en la pestaña de pruebas de Unity, debemos asegurarnos de eliminar la cámara creada por Unity, puesto que creará conflictos al usar el sistema de realidad virtual. Podemos hacer esto yendo al editor de elementos y seleccionando la cámara para desactivarla en el inspector.

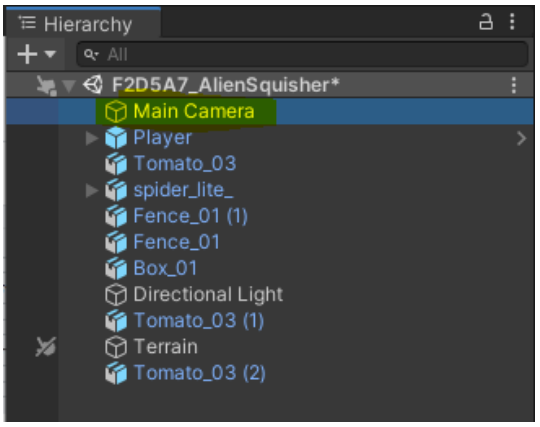


Figura 4.1.2.3. Cámara principal en el editor de elementos.

Ahora nos dirigiremos al inspector y quitaremos el cheque en la parte izquierda del nombre del objeto. Este cheque nos permitirá siempre desactivar un objeto que queramos tener en la escena sin que siga su comportamiento habitual. En este caso desactivamos la cámara porque el objeto 'Player' ya tiene una compatible con realidad virtual, y ambas cámaras entrarían en conflicto.

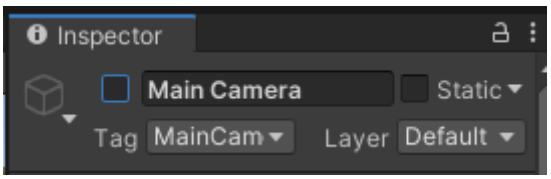


Figura 4.1.2.3. Desactivar cámara predeterminada.

Ahora podemos probar nuestro videojuego al presionar el botón de jugar en la parte superior de Unity. Podremos usar la pestaña de pruebas para navegar en la escena desde la perspectiva del jugador. Los controles son iguales a los del editor.

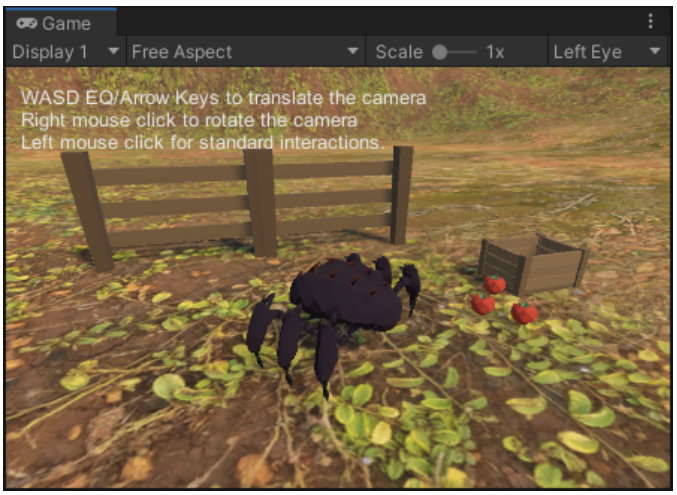


Figura 4.1.2.3. Navegación en la pestaña de pruebas de Unity.

4.1.3. Distinción entre componentes y objetos

En Unity, todos los objetos físicos pueden categorizarse como objetos, y propiedades de dichos objetos, o componentes. Una manera de entender esto fácilmente, es tomando un objeto de nuestra escena y añadiendo distintos componentes para verificar cómo cambia su comportamiento. Luego de eso podremos trabajar con distintos objetos a la vez, y ver cómo estos pueden interactuar entre sí. De esto tratarán las siguientes secciones.

4.2. Manejo de componentes

Como es establecido anteriormente, los componentes determinan la apariencia y el comportamiento de un objeto. Manejaremos distintos tipos de componentes, cada uno cumpliendo una función diferente dentro del objeto. Es importante mencionar que cada objeto tiene un script que controla su función internamente. En la sección de crear scripts podremos observar cómo acceder a otros componentes para crear nuestros propios comportamientos para un objeto.

4.2.1. Componente de colisionador y cuerpo rígido.

A pesar de que podemos ver todos los objetos en nuestra escena al añadirlos, estos no interactúan entre sí necesariamente. Para lograr que los objetos puedan interactuar con el resto de la escena, son necesarios dos componentes relacionados con el sistema de físicas. El primero es el componente de colisionador. Podemos pensar en el colisionador como el aspecto tangible de nuestro objeto. Si el modelo y el material nos permiten verlo, el colisionador nos permite tocarlo.

Para añadir un componente nuevo, podemos seleccionar un objeto (En este caso, elegí uno de los tomates al ser algo pequeño y redondo) y seleccionar “Añadir componente” en el inspector. Primero añadiremos el colisionador.

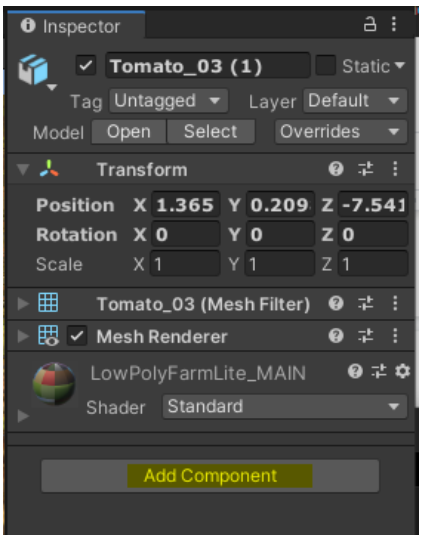


Figura 4.2.1.1. Inspector del objeto de tomate.

Al presionar el botón de añadir componente, saldrá una lista de componentes disponibles junto con un buscador. Podemos buscar 'Collider' para añadir el colisionador. Ya que el tomate tiene forma de esfera, añadiremos uno en forma de esfera.

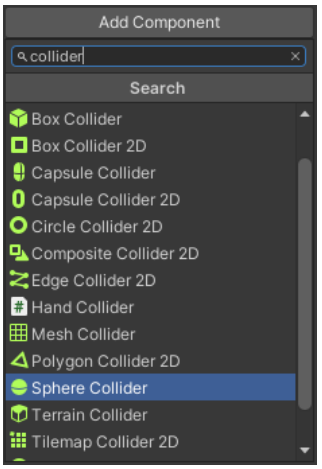


Figura 4.2.1.2. Adición de colisionador de esfera.

El componente de colisionador nos permite modificar la forma del mismo para adaptarlo más fácilmente al objeto. En nuestro caso, podemos usarlo para más fácilmente ajustar la circunferencia del colisionador a la circunferencia del tomate. Podemos hacer esto haciendo clic en la opción de "editar colisionador" en el inspector del componente.

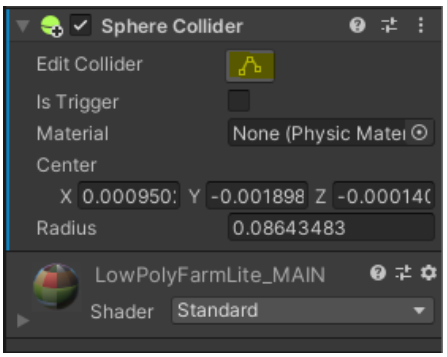


Figura 4.2.1.3. Inspector del colisionador

Luego de presionar este botón, podemos ir al editor de la escena y modificar el colisionador para ajustarse al objeto.

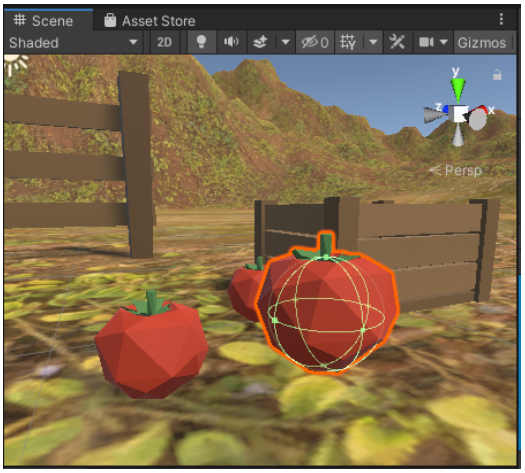


Figura 4.2.1.4. Editor de colisionador.

Luego de tener el colisionador, es importante añadir otro componente relacionado con la física del objeto, el objeto rígido. Este componente le permite al objeto reaccionar a la gravedad, así como a fuerzas provocadas por otros objetos. A pesar de que hay otros modelos de física, el de objeto rígido es el más usado y el esencial para hacer videojuegos con una física realista. Nuevamente, podemos añadirlo mediante el inspector. Además, podemos cambiar datos como la masa del objeto, la resistencia con el aire, y la resistencia angular. Si estamos seguros de los valores en la vida real correspondientes a nuestro objeto, es bueno cambiarlos. De lo contrario, debemos dejar los valores por defecto.

Esta vez, podemos dejar el tomate alto en el aire para observar cómo cae cuando ejecutemos el juego en las pruebas.

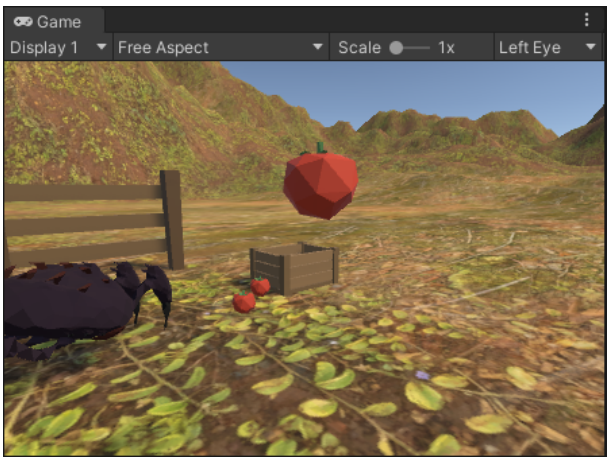


Figura 4.2.1.5. Manzana con físicas incorporadas en la ventana de pruebas.

4.2.2. Componentes interactuables de SteamVR.

Ahora que tenemos un componente que puede interactuar con el resto de la escena, podemos hacer que reaccione a nuestras interacciones como usuario. Para esto, instalaremos dos scripts de SteamVR que permiten la interacción del usuario con objetos: 'Interactable' y 'Throwable'.

Interactable permite que el objeto reconozca nuestra mano e interaccione con ella, como su nombre indica. Por otro lado, Throwable nos permite agarrar el objeto y moverlo en la escena con nuestras propias manos. Este componente es especialmente importante para juegos en los que el jugador deba agarrar algún objeto y moverlo. Podemos añadir ambos componentes de la misma manera que con el colisionador y el cuerpo rígido. Los buscaremos en el menú de añadir componente.

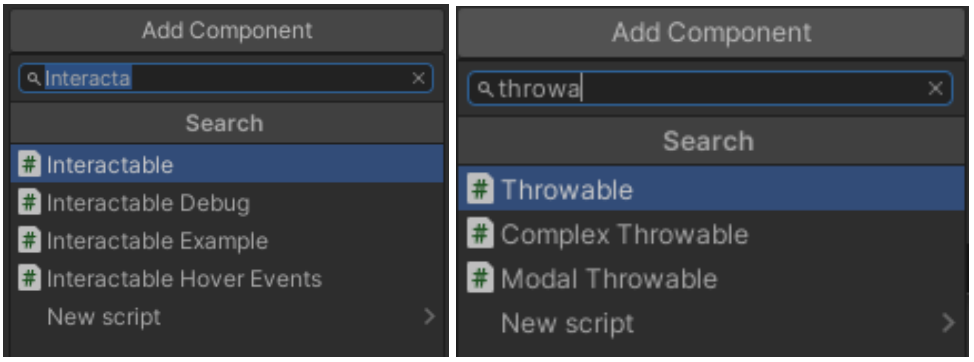


Figura 4.2.2.1. Scripts para interacción en SteamVR.

Luego de añadir estos dos componentes, es posible agarrar el objeto e interactuar con él como lo haríamos en la vida real. Esto lo podemos hacer utilizando el sistema de realidad virtual, o si no contamos con este, podemos comprobarlo en la ventana de pruebas. Ejecutaremos el juego y en la ventana de pruebas podemos agarrar el tomate con clic izquierdo.

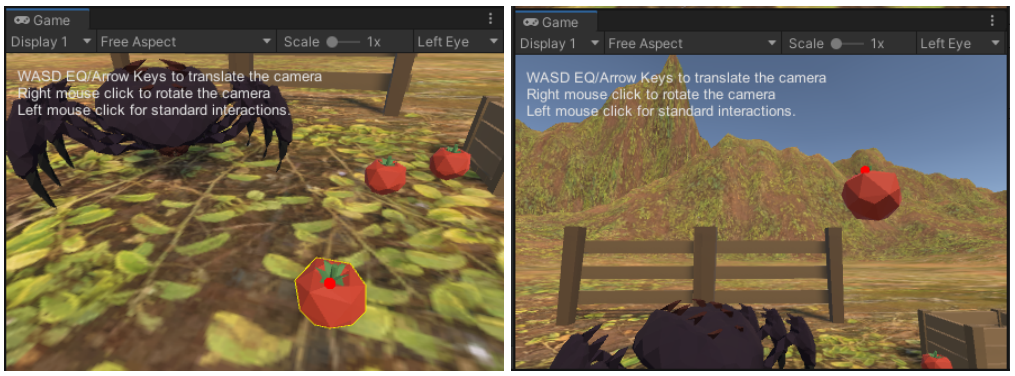


Figura 4.2.2.2. Jugador moviendo el tomate en la ventana de pruebas.

4.2.3. Componentes misceláneos

A continuación, encontrará la descripción y uso de componentes misceláneos que pueden facilitar el desarrollo de videojuegos.

"Audio source" permite que el objeto produzca sonido de cualquier tipo. Podemos añadir un archivo de audio en uno de los formatos recomendados, y cambiar detalles como el volúmen de reproducción, si queremos que se reproduzca al iniciar el juego, y si se reproducirá repetitivamente. Respecto al audio mixer, podemos utilizar uno genérico, o si se trabaja en un proyecto donde ya se haya creado uno, utilizar el existente, puesto que nos permite acoplar nuestro videojuego al sistema de configuración existente. Esto significa que si el jugador baja el volúmen global del juego, nuestro juego también obedecerá a esta configuración. Adicionalmente, puede encontrar información sobre audio y reconocimiento de voz en esta capacitación.

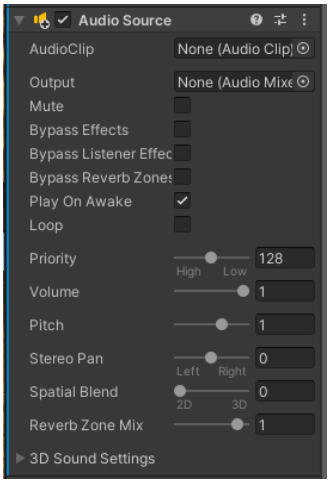


Figura 4.2.3.1. Componente de Audio Source.

Podemos utilizar el componente 'Halo' para añadir un brillo alrededor de un objeto. Esto es especialmente útil para resaltar objetos que necesitamos que el jugador vea en un momento dado. La activación y desactivación de este se realiza mediante script, por lo que nos permite activarlo siempre en el momento correcto.

Figura 4.2.3.2. Componente Halo

El sistema de partículas merece su propia guía, y requiere un poco de investigación en la documentación de Unity para dominar y utilizar a su mayor medida. Para simplificar, nos permite emitir múltiples objetos 'falsos' visuales, conocidos como partículas, a muy bajo costo computacional. Ejemplos comunes permiten crear en tiempo real: Fuego, humo, nieve, chispas, entre otros. Estas partículas se emiten del objeto al que añadimos el componente.

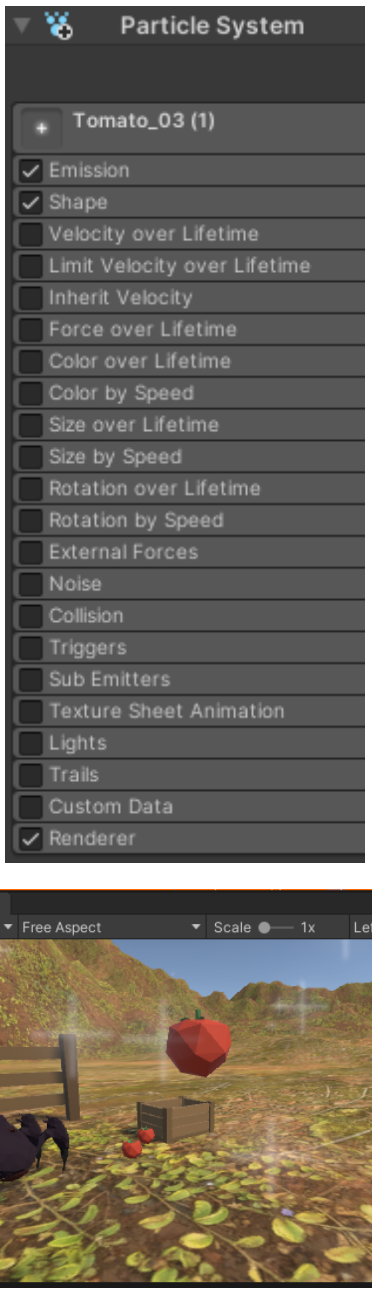


Figura 4.2.3.3. Demostración del sistema de partículas, Tomate emitiendo brillo.

Los componentes de restricciones nos permiten bloquear ciertas propiedades de algún objeto. Por ejemplo, podemos utilizar la restricción de “mirar hacia” para hacer que un objeto siempre tenga su rotación ubicada hacia otro. Esto es útil si queremos que uno de los objetos persiga a otro. Por ejemplo, podemos hacer que un vehículo siempre apunte hacia cierta meta en una pista, o que algún animal siempre esté mirando hacia el jugador. En este caso, lo usaré para que el alien que añadimos siempre mire al tomate. Para esto, debemos añadir una nueva fuente al componente, la cual contendrá al objeto del tomate. Podemos hacer esto arrastrando el objeto del editor de elementos al inspector.

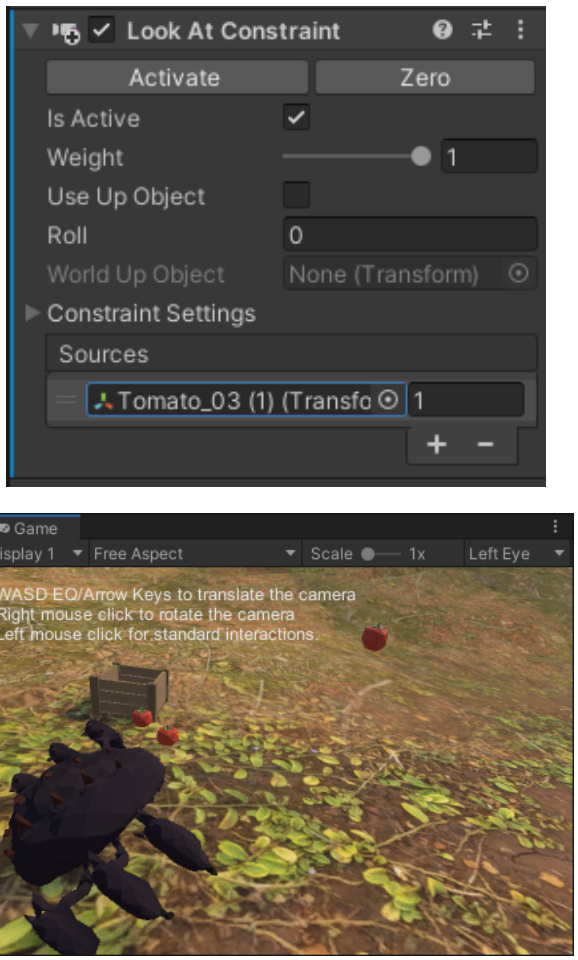


Figura 4.2.3.5. Demostración de componente “Restricción de mirar hacia objeto”

4.3. Manejo de objetos

Los objetos es la manera en la que Unity nos permite tener jerarquías en nuestro proyecto. Cada objeto debe representar un elemento de la escena, y si está compuesto por elementos más pequeños, esto debería verse reflejado en la jerarquía. Es importante mencionar que la posición de los objetos hijos quedará fija, moviéndose en relación con el padre en lugar de en relación con el mundo.

4.3.1. Emparentado entre objetos

Una de las maneras de organizar nuestros objetos, es hacer que unos objetos grandes sean padres de objetos más pequeños. Cabe aclarar que al emparentar dos objetos, los objetos ‘hijos’ adoptan todas las transformaciones del padre. La posición, la rotación y la escala. Al modificar cualquiera de estos valores en el objeto padre, los hijos también adoptarán estas cualidades. También es importante mencionar que un hijo con componentes de física como cuerpo rígido ya no funcionará como un objeto físico al crear un emparentado si este ya tiene componentes físicos. Puesto en otras palabras, las físicas

de un objeto siempre estarán determinadas por las físicas del objeto físico con mayor jerarquía.

4.3.1.1. Categorizar la escena

Si nuestra escena contiene demasiados elementos similares, lo mejor es que sean hijos de un objeto 'categoría' en la que puedan entrar. Por ejemplo, al tener muchos tomates, es muy recomendado tenerlos dentro de un objeto vacío que represente a todos los tomates de la escena. Para empezar, podemos crear un objeto vacío dando clic derecho en el editor de objetos y creando un 'GameObject' vacío. Lo llamaremos 'Tomates'. Podemos arrastrar cada uno de los tomates hacia este objeto nuevo, creando jerarquía.

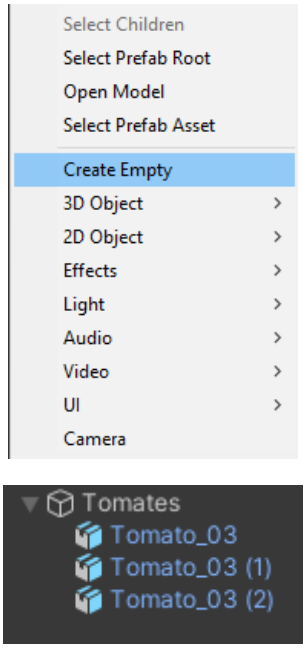


Figura 4.3.1.1.1. Emparentar objetos por categorización.

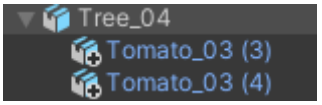
Ahora, si queremos desactivar todos los tomates de la escena, o sencillamente esconderlos del editor cuando trabajemos en algo más, podemos hacerlo al trabajar con el objeto padre 'Tomates'.

Figura 4.3.1.1.2. Objetos tomates al ser desactivados por el objeto padre.

Figura 4.3.1.1.3. Categoría 'Tomates' al ser colapsada en el editor de objetos.

4.3.1.2. Unir distintas partes de un mismo objeto.

Si un objeto tiene partes móviles como un árbol al que le podamos extraer las frutas, podemos emparentar las partes pequeñas (frutas) al árbol. De esta manera, podemos mover el árbol manteniendo la posición de las frutas en cada rama, mientras las frutas individualmente no pierden movilidad.



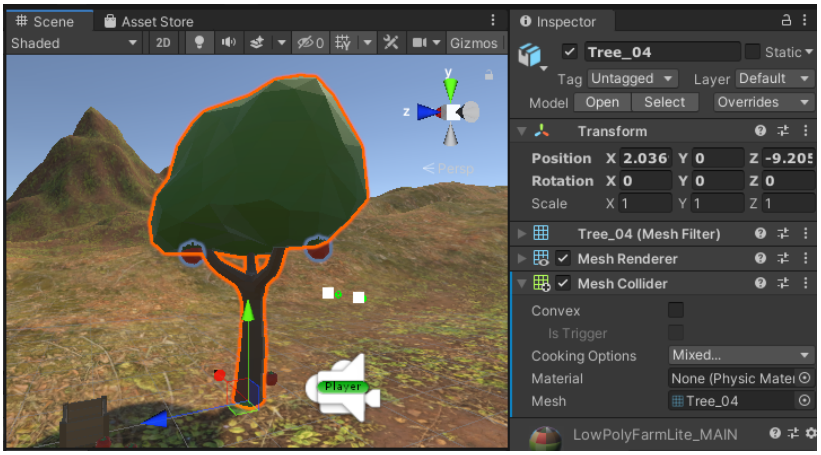


Figura 4.3.1.2.1. Objeto árbol conteniendo a dos frutas.

Figura 4.3.1.2.2. La fruta aún mantiene sus propiedades físicas y su movilidad.

4.3.1.3. Crear pertenencia de un objeto a otro

Finalmente, podemos hacer que un objeto pertenezca completamente a otro. Esto tiene múltiples usos, como darle una herramienta al jugador que no pueda soltar accidentalmente, o darle armas a un enemigo. Para lograr la pertenencia, el objeto hijo no debe tener ningún tipo de componente físico, ya que este debe depender

completamente del padre. Por ejemplo, podemos llenar una caja de tomates sin cuerpo rígido, y darle todos los componentes de física únicamente a la caja. de esta manera, cuando levantemos la caja se levantará con todos los tomates adentro, incluso si la volteamos o la lanzamos.

Figura 4.3.1.3.1. Tomates pertenecientes a una caja.

La desventaja de este método es que al no tener ningún componente físico, los tomates ahora ignoran completamente el sistema de gravedad, y no interactúan entre sí. Este problema puede mitigarse ubicando los objetos hijos en una posición 'realista' inicialmente, y usando sólo este método para objetos que deban ser completamente dependientes de sus padres.

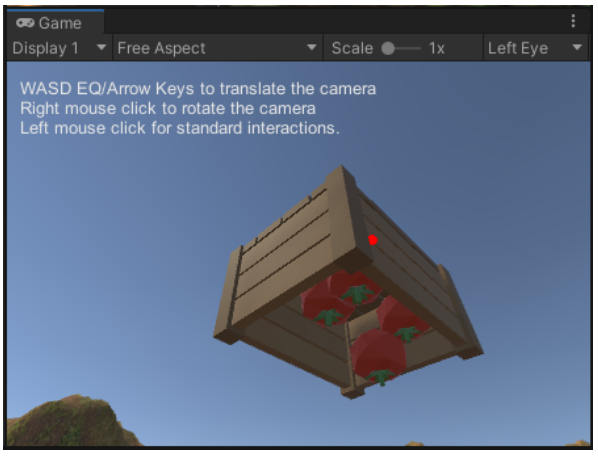


Figura 4.3.1.3.2. Los objetos ahora no dependen de la gravedad de la escena.

4.3.2. Empaquetar un objeto

Si tenemos un objeto compuesto o con componentes añadidos que queramos reutilizar, lo podemos hacer al empaquetarlo. Al empaquetarlo, toda la información sobre componentes añadidos y objetos hijos permanecerá con las demás copias del objeto. Podemos hacer esto al arrastrar nuestro objeto padre desde el editor de objetos hasta

nuestra carpeta dedicada para prefabricados. En este caso, se guardará la caja con tomates.

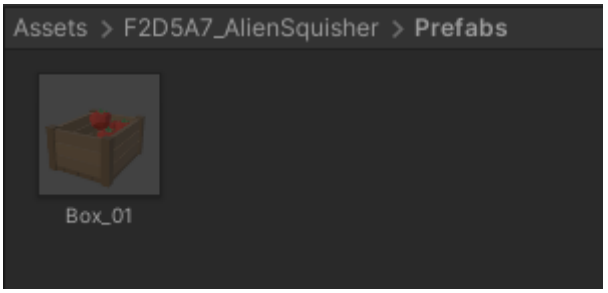


Figura 4.3.2.1. Prefabricado de caja de tomates.

Este nuevo objeto prefabricado puede ser reutilizado en la misma escena o en otras, mientras esté dentro de la carpeta de nuestro videojuego, junto con todos los elementos utilizados. En este caso, también deben estar la caja, el tomate, y todo lo que estos a su vez utilicen.

Podemos reutilizar este objeto prefabricado arrastrándolo de nuevo desde el explorador de archivos hasta la escena donde lo queramos utilizar. Para hacer la demostración, lo agregaré a una escena nueva.

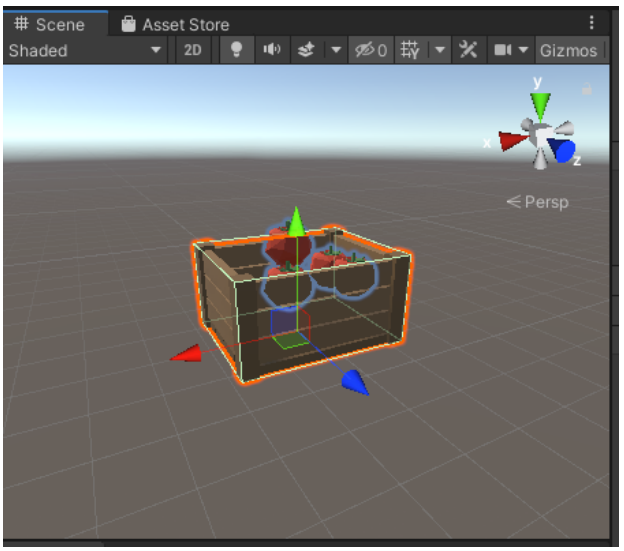


Figura 4.3.2.2. Objeto prefabricado reutilizado en otra escena.

4.3.3. Desempaquetar un objeto

Podemos seguir el proceso inverso para tomar un objeto prefabricado y dividirlo en las partes que lo conforman. Por ejemplo, si en esta nueva escena deseamos añadir más objetos, podemos hacerlo de una manera segura al desempaquetar el objeto original, hacer los cambios necesarios, y volver a empaquetarlo. En este caso, deseo añadir una naranja a la caja de tomates. Para hacer esto, es necesario desempaquetar la caja de tomates. Esto es posible al hacer clic derecho en el objeto empaquetado en el editor de objetos y seleccionar “Desempaquetar” o “Desempaquetar por completo”. Esta segunda

opción sirve para casos en los que tengamos múltiples objetos prefabricados, unos dentro de otros.

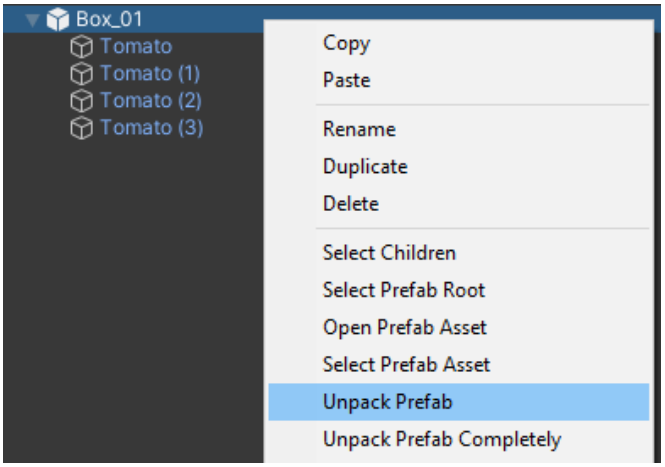


Figura 4.3.3.1. Desempaquetar un objeto.

Ahora podemos hacer los cambios necesarios a nuestro objeto. Podemos añadir, eliminar o modificar objetos del prefabricado y no van a afectar al original. Cuando terminemos, podemos volver a empaquetarlo para reutilizar esta nueva versión.

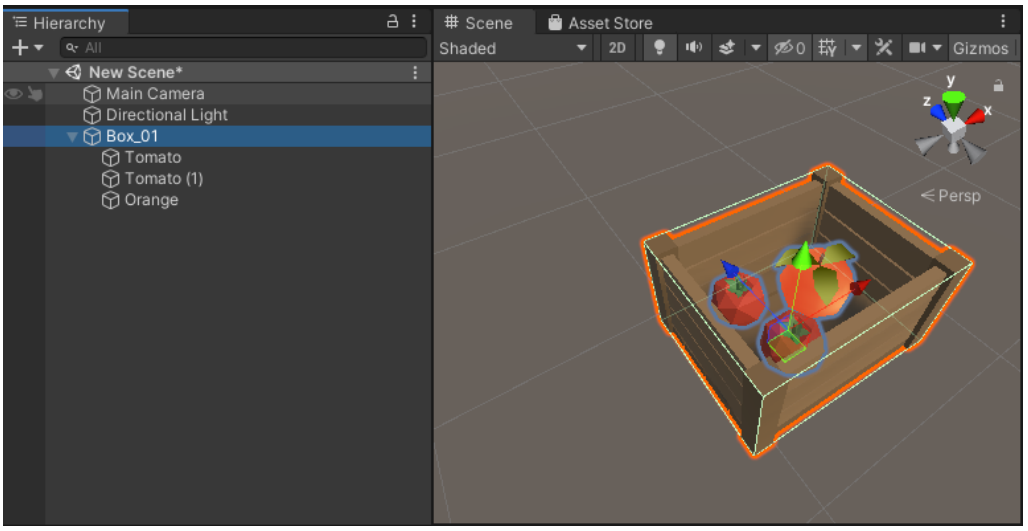


Figura 4.3.3.2. Objeto prefabricado desempaquetado y modificado

Figura 4.3.3.3. Nuevo objeto prefabricado.

5. Creación y uso de scripts

El último concepto que debemos aprender a manejar para empezar a crear juegos únicos, es el de la creación y el uso de scripts. Podemos pensar en los scripts como programas especializados en hacer funcionar cada componente de diferente manera. Podemos entonces crear scripts para cambiar el comportamiento de un objeto a nuestro antojo. Estos scripts están hechos sobre el lenguaje de programación C#, aunque no requieren un nivel avanzado de programación. Podrá encontrar la documentación para la creación de scripts en Unity en la sección de recursos.

Para esta sección, se creará un script para el alien que lo haga perseguir tomates.

5.1. Crear un script

Podemos crear un script yendo a la carpeta dedicada de nuestro proyecto para scripts, y crear uno nuevo con la opción “Crear nuevo script en C#”.

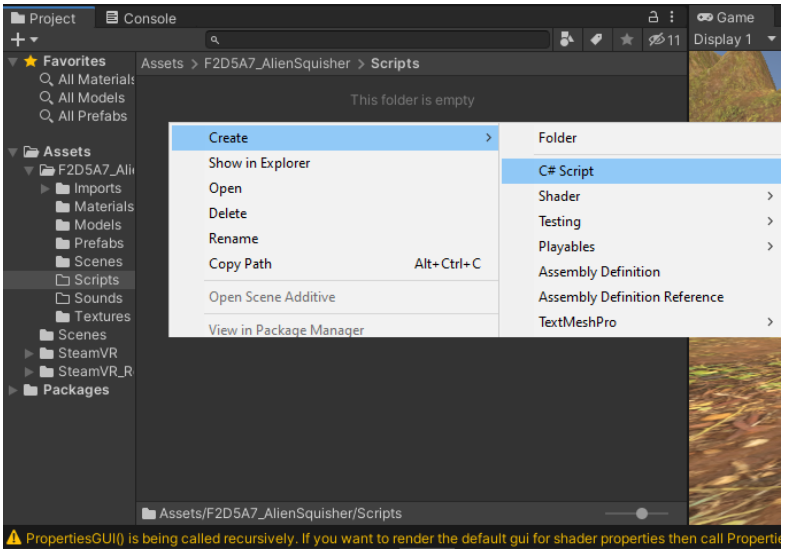


Figura 5.1.1. Creación de nuevo script.

Esto creará un nuevo script que podemos abrir y editar en nuestro software de edición de texto preferido. Se recomienda Visual Studio Code para editar el código, puesto que gracias a su modularidad tiene buen soporte de Unity. Luego de crear este archivo lo podemos añadir al objeto de nuestra preferencia seleccionándolo y agregándolo como componente, similar a como añadimos otros componentes anteriormente. Lo buscaremos en la lista de componentes por el mismo nombre que tenga el archivo de C#.

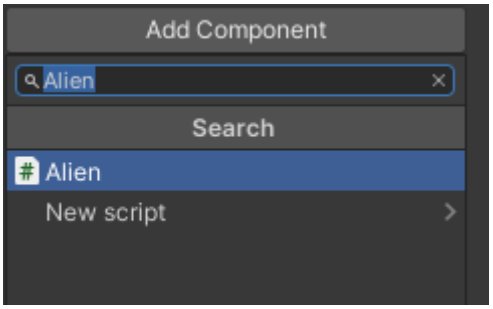


Figura 5.1.2. Adición del script como componente

En este caso, el script se llama ‘Alien’. Sin embargo, en el caso de trabajar en múltiples minijuegos, el nombre de sus scripts deben tener una manera de ser distinguidos entre sí. Esto es para evitar conflictos con el manejo de scripts de Unity. Por ende, en un caso práctico el script debería llamarse “F2D5A7_Alien”. Otros scripts como uno para naves espaciales se llamaría entonces “F2D5A7_Spaceship”, siguiendo las regulaciones sobre nombramiento en el caso de este videojuego.

A partir de ahora, cada modificación hecha al código será revisada por Unity, ya que uno de los elementos de la escena necesita el script para poder funcionar. Esto nos servirá para hallar errores de compilación.

5.2. Conceptos básicos de scripting

Al crear un script, Unity creará dos funciones básicas: Start y Update. Ambas son importantes para el uso de scripts. Se pueden diferenciar de la siguiente manera:

- Start se ejecuta una vez al iniciar la escena. Aquí deben hacerse las operaciones pesadas que puedan ejecutarse sólo una vez en el juego.
- Update se ejecuta una vez cada fotograma. Esto quiere decir que si nuestro juego se ejecuta a 60 fotogramas por segundo (Estándar para los videojuegos modernos), Update se llamará 60 veces por segundo. En esta función sólo deben haber operaciones con bajo costo computacional. Evitar ciclos y llamados recursivos aquí.

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

public class Alien : MonoBehaviour
{
    // Esta función se llama antes del primer fotograma
    void Start()
    {

    }

    // Esta función se llama cada fotograma
    void Update()
    {

    }
}
```

Figura 5.2.1. Código de script básico en Unity

Podemos también llamar eventos distintos a los de Start y Update. Por ejemplo, si nuestro objeto tiene un colisionador, podemos hacer un llamado de función cada que este choque con algo, con la función nativa de Unity OnCollisionEnter. La documentación de scripting de Unity contiene más casos para eventos a los que puede llamar funciones.

Como en otros lenguajes de programación, podemos además crear variables para la clase, así como nuestras propias funciones. Estas son las que construirán la lógica de nuestra aplicación. En este caso, crearemos una llamada Seguir. Esta función es la que contendrá la lógica mediante la cual el Alien podrá moverse. De momento, queremos que el alien siempre esté siguiendo a los tomates. Por ende, Seguir se va a llamar dentro de la función de Update.

```
    // Esta función se llama cada fotograma
    void Update()
    {
        Seguir()
    }

    // Esta función contendrá la lógica del seguimiento del alien.
    void Seguir()
    {

    }
```

Figura 5.2.2. Creación de función en C#

Ahora, vamos a darle una velocidad al Alien para que pueda moverse. Podemos hacer esto como una de las variables de la clase. Podemos asignarla como una variable pública de tipo float, ya que queremos un control preciso sobre la velocidad del Alien. Además, la podemos inicializar en un valor razonable como 1.0f. Es importante mencionar que toda variable pública que declaremos podrá ser accedida y modificada mediante el inspector.

```
public float velocidad = 1.0f;
```

Figura 5.2.3. Creación de variables en C#

*Se utiliza 1.0f en lugar de 1.0, ya que 'f' es la manera de C# de separar variables de tipo *float* a variables de tipo *double*.

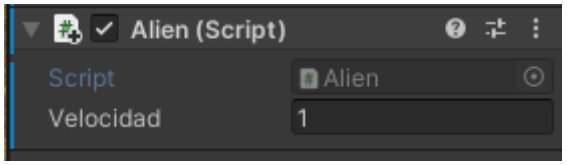


Figura 5.2.4. Opción de modificar variable desde el inspector

Teniendo una velocidad y la función creada, podemos hacer que cada que se llame, el Alien se mueva en una dirección. Para probar que sí funcione, podemos hacer que se mueva hacia arriba cada fotograma. Esta velocidad entonces debemos multiplicarla por un delta, y ponerla sobre la ubicación 'y' de nuestro Alien, haciendo un llamado hacia su componente 'Transform'. Esta suma la debemos hacer representada con un vector de 3 dimensiones, ya que así es como es representada la posición del Alien.

*Un delta es una unidad de tiempo que hace que el movimiento se mantenga constante en el videojuego sin importar si existen cambios en la tasa de fotogramas por segundo. Es importante usar esta variable cada que utilizamos la función Update o cualquier función que dependa de los fotogramas.

```
public class Alien : MonoBehaviour
{

    public float velocidad = 1.0f;

    // Esta función se llama antes del primer fotograma
    void Start()
    {

    }

    // Esta función se llama cada fotograma
    void Update()
    {
        Seguir();
    }

    // Esta función contendrá la lógica del seguimiento del alien.
    void Seguir()
    {
        //Multiplicar la velocidad por el tiempo delta
        float desplazamiento = velocidad * Time.deltaTime;
        //Añadir el desplazamiento al vector actual de posición.
        transform.position = transform.position + new Vector3(0, desplazamiento, 0);
    }
}
```

Figura 5.2.5. Creación de función para mover verticalmente un objeto.

Figura 5.2.6. Alien desplazándose verticalmente mediante el script.

5.3. Relacionar script con objetos

Podemos mover a nuestro alienígena, aunque no tendrá mucho uso si sólo puede moverse en una dirección sin ningún criterio que recree el comportamiento que deseamos. Ahora implementaremos la relación del Alien con otros objetos, específicamente con uno de los tomates. Para hacer esto, necesitamos especificar qué componente del tomate queremos utilizar. En este caso, queremos utilizar la posición, que está guardada en el componente transform del tomate. Por ende, vamos a declarar un transform público para que el Alien utilice.

Figura 5.3.1. Componente transform del tomate

```
//Componente del tomate que se utilizará para la posición
public Transform tomate;
```

Figura 5.3.2. Declaración del componente a utilizar.

Puede revisar la documentación de Unity para aprender el uso de cualquier componente desde scripting. Además, puede hacer referencia a otros scripts hechos por usted o por otros usuarios, como veremos en las siguientes secciones. Ahora debe asignar el tomate en la ventana del inspector, seleccionando el transform correcto en el script del Alien.

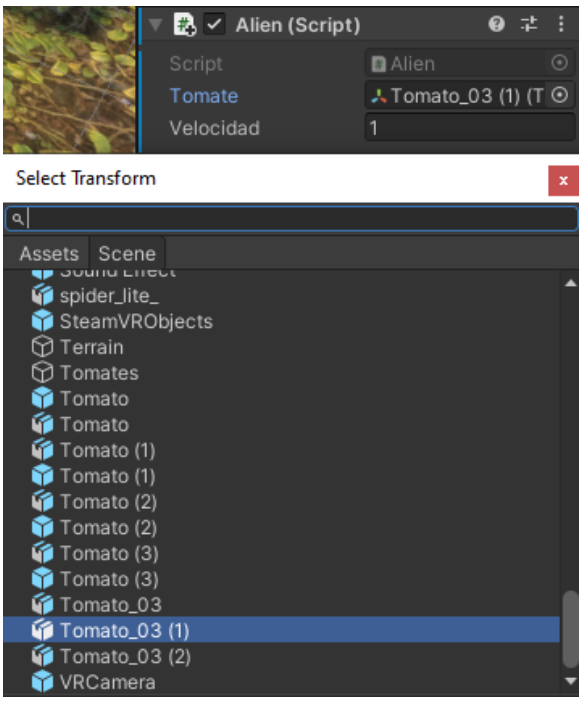


Figura 5.3.3. Designar el objeto deseado al script del Alien.

Luego de declarar el componente Transform en el script del Alien, podemos utilizarlo en nuestra función de Seguir para mover al Alien en la dirección del tomate. Afortunadamente, no necesitamos aplicar operaciones trigonométricas manualmente para cambiar esta posición. En cambio, Unity nos provee con una función que hace exactamente la transformación que necesitamos. Usaremos la función Vector3.MoveTowards, aplicando el transform del Alien, del tomate, y la velocidad como parámetros.

```
// Esta función contendrá la lógica del seguimiento del alien.
void Seguir()
{
    //Multiplicar la velocidad por el tiempo delta
    float desplazamiento = velocidad * Time.deltaTime;
    //Movel el Alien hacia la dirección del tomate, con un desplazamiento de velocidad * delta
    transform.position = Vector3.MoveTowards(transform.position, tomate.position, desplazamiento);
}
```

Figura 5.3.4. Modificación de la función para hacer que el objeto siga a otro.

Al ejecutar el videojuego, el Alien debería moverse lentamente hacia el tomate. Puede mover el tomate con el simulador de realidad virtual para comprobar que el Alien siempre se moverá en la dirección del tomate.

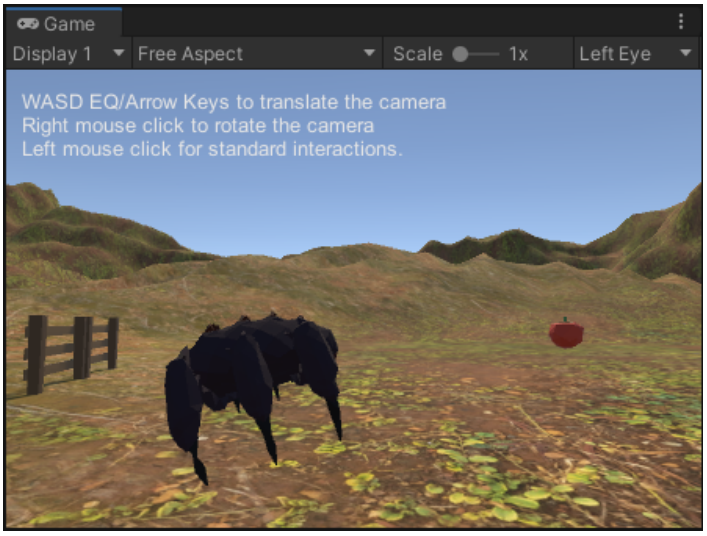


Figura 5.3.5. Demostración de comportamiento de Alien

5.4. Generar prefabricados

Ya tenemos un Alien que seguirá al tomate en todo momento. Sin embargo, no queremos repetir este proceso para generar más aliens en el futuro. Lo ideal es tener un sistema que genere Aliens periódicamente con el mismo comportamiento. Para esto utilizaremos una técnica llamada instanciación. Primero, necesitaremos empaquetar nuestro Alien como un prefabricado. Debemos hacer esto luego de haber implementado el script para el comportamiento del Alien.

Figura 5.4.1. Alien como prefabricado.

A continuación, elegiremos un objeto mediante el cual se generarán los nuevos alienígenas. En este caso, ya contamos con una nave espacial, así que la utilizaremos para la instanciación. Crearemos un script nuevo para la nave espacial y se lo asignaremos en la escena.

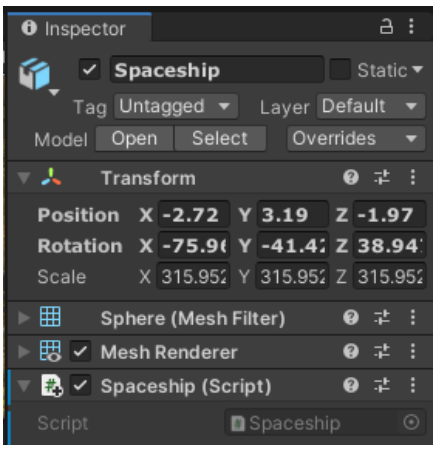


Figura 5.4.2. Script añadido a la nave espacial.

Inicialmente, intentaremos generar un alien en la función Start. Es decir, deseamos que al iniciar el juego se cree un alien que empiece a seguir al tomate. El primer paso es declarar un objeto prefabricado en el script de la nave espacial. Todos los objetos del juego pertenecen a la clase en C# de GameObject.

```
//Referencia al alien que se generará
public GameObject AlienPrefab;
```

Figura 5.4.3. Declaración de prefabricado en C#.

Adicionalmente, podemos tomar una ubicación en la escena como hicimos con el tomate. Esta ubicación servirá como el punto donde los aliens aparecerán.

```
//Lugar donde los aliens se generarán
public Transform ubicacionInicial;
```

Figura 5.4.4. Declaración de la posición de donde los objetos aparecerán.

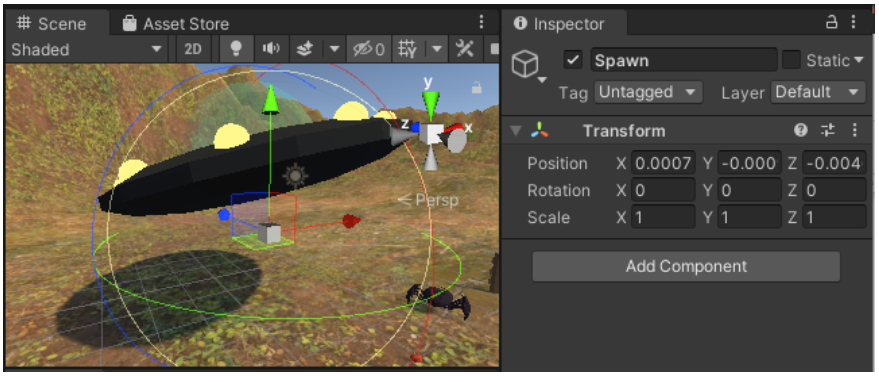


Figura 5.4.5. Ubicación para la generación de aliens.

Ahora podemos utilizar la función dedicada de Unity para instanciar objetos, asignándole como parámetros la ubicación de inicio y el prefabricado del Alien. Además vamos a agregar un parámetro de rotación. Este último sólo debe cambiarse en caso de que queramos que el alien se inicie en una rotación en específico.

```
// Esta función se llama antes del primer fotograma
void Start()
{
    //Instanciar alien en una ubicación inicial, sin ninguna rotaicón
    Instantiate(AlienPrefab, ubicacionInicial.position, Quaternion.identity);
}
```

Figura 5.4.6. Código para instanciar un objeto prefabricado.

Ahora podemos asignar el prefabricado del Alien arrastrándolo del navegador de archivos al inspector. También debemos asignar la ubicación inicial y ejecutar el código para comprobar la creación del alien.

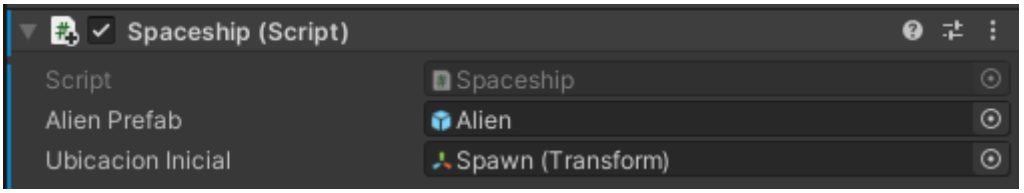


Figura 5.4.7. Asignación de objetos para la generación del Alien

Figura 5.4.8. Correcta instanciación del Alien

A pesar de que el alien se generó correctamente, este no tiene el comportamiento esperado. La consola de Unity nos da una pista de por qué.

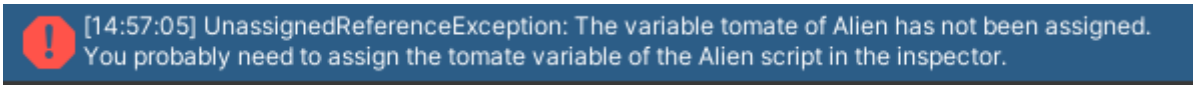


Figura 4.4.4.9. Error por no asignar la variable del tomate a los aliens instanciados.

A pesar de que asignamos el tomate al Alien original, los aliens nuevos que la nave espacial genere no van a tener esta información. Para esto, debemos asignarle el tomate a

cada alien al momento de crearlo. Una posible solución es asignar el tomate a la nave espacial, y dejar que esta se lo asigne a cada alien al momento de instanciación. Para esto, declaramos el tomate en el script de la nave espacial. Para generalizar, lo llamaremos ‘objetivo’.

```
//Lugar al que atacarán los aliens
public Transform objetivo;
```

Figura 5.4.10. Declaración de variable tomate en la nave espacial.

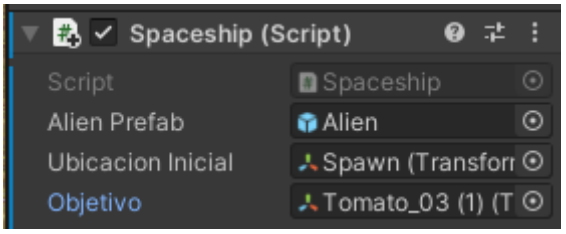


Figura 5.4.11. Declaración del tomate en la nave espacial.

Ya que necesitamos hacer más cosas al momento de instanciar, podemos crear una función llamada “Crear Alien” en la que además asignemos el tomate a los componentes necesarios del alien creado. Para esto, guardaremos el alien instanciado en una variable, y accederemos a cada uno de los componentes necesarios mediante la función de Unity GetComponent. A esta función le agregaremos el nombre del componente que buscamos.

```
Alien alienscript = alien.GetComponent<Alien>();
```

Figura 5.4.12. Declarar el componente de script de los aliens instanciados.

Para la restricción de mirar hacia el objeto, debemos implementar la librería de Unity de animaciones. En casos como los de este componente, es bueno revisar la documentación para verificar las dependencias de cada componente.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine.Animations;
4   using UnityEngine;
```

Figura 5.4.13. Declaración de librería de animaciones.

```
LookAtConstraint restriccionAlien = alien.GetComponent<LookAtConstraint>();
```

Figura 5.4.14. Declaración del componente de restricción.

Ahora podemos asignarle el tomate al componente correspondiente. Haremos esto accediendo las propiedades del componente como atributos de una clase.

```
// Esta función se llama antes del primer fotograma
void Start()
{
    CrearAlien();
}

//Crear alien en una ubicación con un objetivo
void CrearAlien(){
    //Instanciar alien en una ubicación inicial, sin ninguna rotaicón
    GameObject alien = Instantiate(AlienPrefab, ubicacionInicial.position, Quaternion.identity);
    //Componentes del alien modificables para cada instancia

    //Añadir tomate a script de alien
    Alien alienScript = alien.GetComponent<Alien>();
    alienScript.tomate = objetivo;

    //Añadir nueva fuente para la restricción de mirar del alien
    LookAtConstraint restriccionAlien = alien.GetComponent<LookAtConstraint>();
    //Crear una fuente para la restricción
    ConstraintSource fuenteRestriccion = new ConstraintSource();
    fuenteRestriccion.sourceTransform = objetivo;
    fuenteRestriccion.weight = 1;
    restriccionAlien.AddSource(fuenteRestriccion);
}
```

Figura 5.4.15. Función para crear aliens instanciados con el error corregido

Podemos verificar el comportamiento del alien generado desactivando el original en la escena y moviendo nuevamente el tomate. Ahora el Alien debería moverse en la dirección del tomate desde el momento en el que aparece.

Figura 5.4.16. Alien generado con el comportamiento correcto.

5.5. Temporizadores

Luego de haber generado un Alien, queremos que este no sea el único que aparezca. Después de todo, el juego trata sobre una invasión de aliens. No podemos

invocar uno nuevo en cada fotograma con Update, pues esto generaría 60 aliens por segundo en promedio, y computacionalmente sería demasiado caro.

La solución es crear un temporizador en la nave espacial que cree un Alien nuevo cada 10 segundos. Para esto, crearemos una variable contador en el script de la nave espacial. Además de esto, crearemos una función para descontarle tiempo al temporizador, y crear un alien nuevo cuando este llegue a cero. Además, luego de crear un Alien, este temporizador debe reiniciarse. Podemos descontar el tiempo utilizando el deltaTime descrito anteriormente.

```
//Temporizador para la creación de Aliens
public float temporizador = 10.0f;
```

Figura 5.5.1. Declaración e inicialización del temporizador.

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

//Descontar tiempo del temporizador, crear un alien y reiniciarlo cuando este llegue a cero.
void descontarTiempo(){
    if (temporizador> 0){
        temporizador -= Time.deltaTime;
    }
    else{
        temporizador = 10.0f;
        CrearAlien();
    }
}
```

Figura 5.5.2. Función para descontar tiempo del cronómetro o reiniciarlo

El juego ahora debería generar un Alien al inicio, además de uno adicional cada 10 segundos.

Figura 5.5.3. Múltiples aliens siendo creados.

Finalmente, queremos que la frecuencia de aliens también sea una variable que podamos controlar, quizás aumentando a medida que la dificultad aumente.

```
//Temporizador para la creación de Aliens
public float frecuenciaDeAliens = 10.0f;
float temporizador;
```

Figura 5.5.4. Declaración de variable para la frecuencia de aliens.

```
// Esta función se llama antes del primer fotograma
void Start()
{
    temporizador = frecuenciaDeAliens;
    CrearAlien();
}
```

Figura 5.5.5. Asignación de la variable al temporizador.

```
else{
    temporizador = frecuenciaDeAliens;
    CrearAlien();
}
```

Figura 5.5.6. Asignación de la variable cuando el temporizador reinicia.

5.6. Accionadores y etiquetas

5.6.1. Creación del accionador

Luego de haberle asignado todo el comportamiento a los aliens, ahora debemos darle al jugador la oportunidad de defenderse. De lo contrario, pronto el juego se llenará de aliens y el jugador sólo podrá huir. Ahora, implementaremos una bota que el jugador podrá utilizar para pisar a los Aliens. Esta tendrá los mismos componentes de cuerpo rígido, Interactable, y Throwable, para que el jugador pueda utilizarla para aplastar los aliens.

La bota además tendrá una particularidad, pues le añadiremos dos colisionadores. Uno de ellos tendrá la forma de la bota, mientras el otro será un cubo, al que le daremos la propiedad “Is trigger” o “Es accionador”. Este accionador debe ser un poco más grande que el objeto.

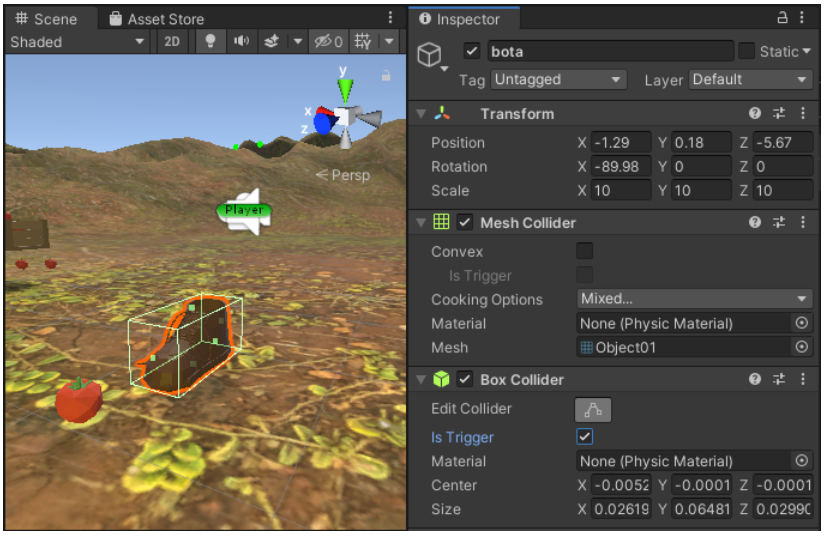


Figura 5.6.1.1. Bota con dos colisionadores, uno de ellos es accionador.

Un accionador se diferencia de un colisionador en que nunca va a impedir el paso de otros objetos físicos. Esto es importante, pues sólo queremos que la bota elimine los aliens, no que bloquee otros objetos.

5.6.2. Asignación de etiquetas

Ya que sólo queremos que las botas interactúen con los aliens, debemos diferenciar a los aliens de otros objetos en la escena. Vamos a hacer esto mediante el uso de etiquetas. Podemos hacer esto modificando el prefabricado de Alien, dando doble clic a su archivo en el navegador de archivos. Esto nos llevará al editor de prefabricados, el cual es igual al editor de escena, pero sólo aplica al prefabricado que seleccionemos.

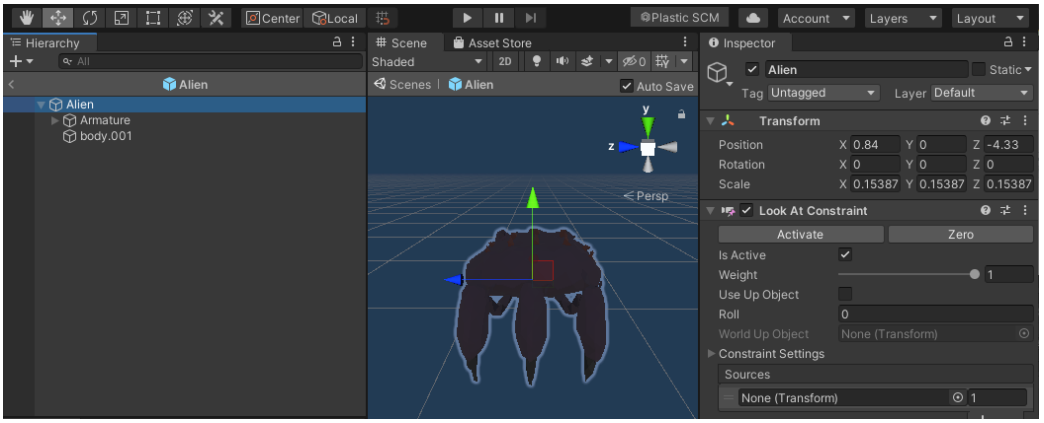


Figura 5.6.2.1. Editor de prefabricados.

En este editor podemos cambiar la etiqueta del Alien. Podemos crear una nueva en el inspector, abriendo la pestaña de tab debajo del nombre del objeto, y seleccionando “Añadir etiqueta”.

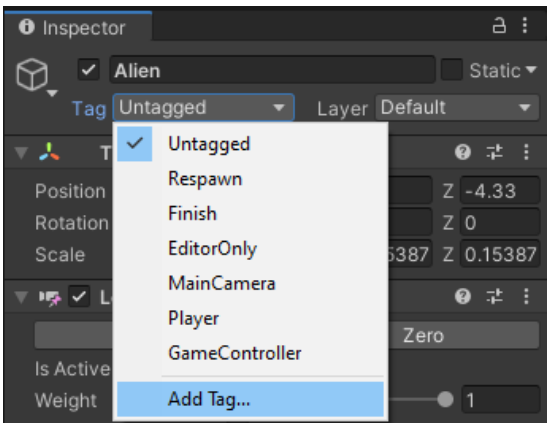


Figura 5.6.2.2. Cambiar etiqueta de objeto.

Esto abrirá el menú de etiquetas, donde podemos crear una nueva acorde al elemento en nuestro videojuego. Es importante aclarar que esta lista se comparte en todo el proyecto. Por ende, es de vital importancia crear la etiqueta con el código de nuestro videojuego y el elemento que representa, para no mezclar etiquetas entre distintos videojuegos.

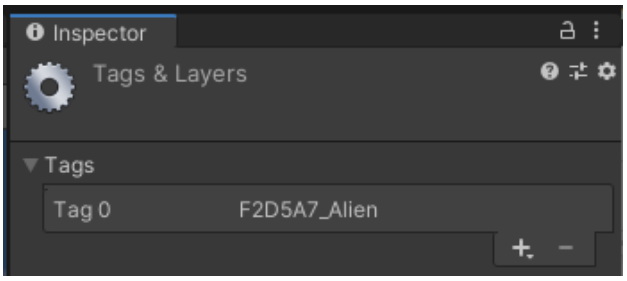


Figura 5.6.2.3. Creación de etiqueta para Alien.

Ahora podemos volver al Alien y asignarle la etiqueta guardada. Acto seguido guardaremos el prefabricado y regresaremos a la escena. Adicionalmente, le asignaremos un colisionador al alien para que la bota pueda interactuar con cada alien.

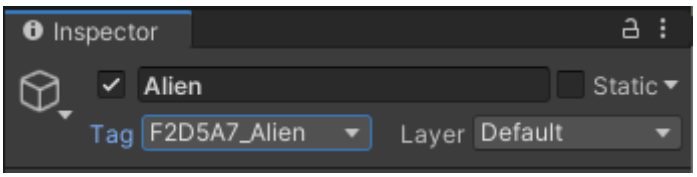


Figura 5.6.2.4. Asignación de etiqueta.

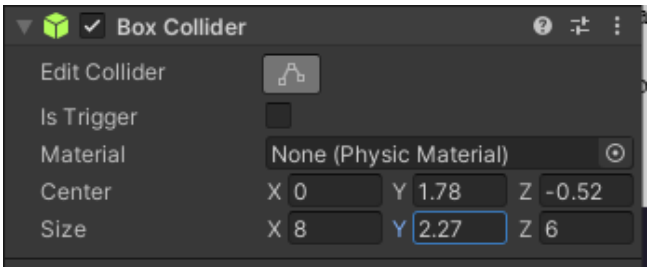


Figura 5.6.2.5. Adición de colisionador

5.6.3. Relación entre accionadores y etiquetas

Por último, haremos que los aliens 'mueran' al entrar en contacto con la bota. Para esto, crearemos un script para la bota. Este script hará uso del accionador para reconocer cuando entra en contacto con un alien y eliminarlo. Haremos uso entonces de la función para accionadores 'OnTriggerEnter'.

```
//Esta función se llamará cada que la bota interactúe con otro objeto
private void OnTriggerEnter(Collider other){

}
```

Figura 5.6.3.1. Función predeterminada para usar triggers.

En esta función, 'other' hace referencia al objeto con el cual la bota interactúa en cualquier momento dado. Queremos que cuando este objeto sea un alien, este sea eliminado. Sin embargo, no podemos simplemente eliminarlo, pues eliminaría otros objetos que no aliens. Podemos comprobar esto haciendo que la consola nos muestre con qué objeto está interactuando la bota en cada momento.

```
//Esta función se llamará cada que la bota interactúe con otro objeto
private void OnTriggerEnter(Collider other){
    Debug.Log(other);
}
```

Figura 5.6.3.2. Código para imprimir el objeto que interacciona en consola.

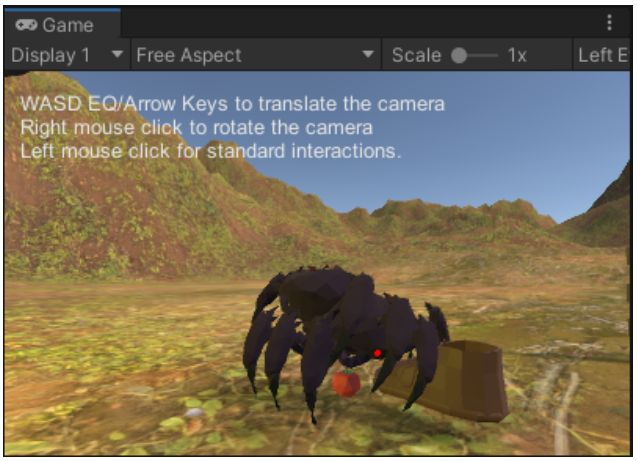


Figura 5.6.3.3 Interacción de la bota con otros objetos en la escena

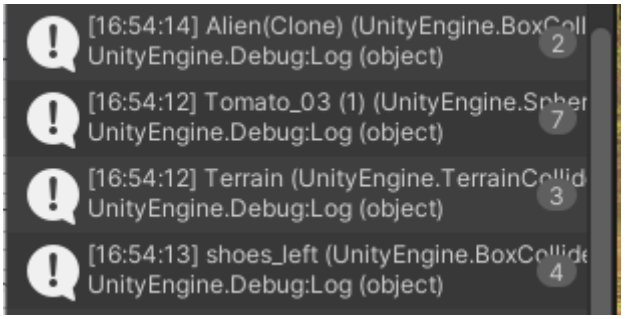


Figura 5.6.3.4. Lista de interacciones de la bota.

Para solucionar esto, filtraremos sólo objetos con la etiqueta que le agregamos a los aliens. De esta manera, todo el comportamiento de la bota se enfocará exclusivamente en aliens y otros objetos de esta misma categoría. Esto es fácil mediante la función de Unity para comprobar etiquetas. Es importante escribir la etiqueta exactamente como la hayamos creado. Cualquier diferencia entre la etiqueta asignada al objeto y la etiqueta escrita en código hará que el comportamiento no se active. Además, recuerde que estamos utilizando un colisionador. Debemos referenciar el objeto al que pertenece este colisionador.

```
//Esta función se llamará cada que la bota interactúe con otro objeto
private void OnTriggerEnter(Collider other){
    if (other.gameObject.CompareTag("F2D5A7_Alien"))
    {
        Debug.Log ("Alien!");
    }
}
```

Figura 5.6.3.5. Modificar función para aceptar sólo aliens.

Finalmente, utilizaremos la función de Destroy para destruir al Alien cuando este interactúa con la bota.

```
//Esta función se llamará cada que la bota interactúe con otro objeto
private void OnTriggerEnter(Collider other){
    if (other.gameObject.CompareTag("F2D5A7_Alien"))
    {
        Destroy(other.gameObject);
    }
}
```

Figura 5.6.3.6. Modificar función para destruir los aliens en entrada.

Ahora podemos ejecutar nuestro juego, el cual ya debería estar completo al tener los aliens persiguiendo los vegetales y la opción de aplastarlos con la bota.

Figura 4.4.6.3.7. Aliens antes de interactuar con la bota.

Figura 5.6.3.8. Aliens siendo destruidos por la bota.

6. Elementos audiovisuales

Con las mecánicas del videojuego listas, ahora podemos añadir más componentes para ayudar al jugador a entender cómo jugar el juego más fácilmente. La jugabilidad es importante, pero también lo es crear la escena de manera que los objetivos del juego sean claros para el usuario. Añadir texto, instrucciones por audio, y retroalimentación puede hacer que el usuario aprenda rápidamente a jugar y complete sus objetivos con mayor facilidad.

6.1. Incorporación de texto en 3D

Al trabajar con juegos minimalistas, se recomienda no utilizar mucho texto, ya que puede confundir al jugador en actividades que deberían ser sencillas. Adicionalmente, algunos juegos se realizan con los ojos cerrados, por lo que usar texto no es una opción viable para dar indicativos al jugador. Sin embargo, en los casos en los que sí se deba usar texto, es importante que este sea claro y se pueda leer a la distancia adecuada en realidad virtual. La función predeterminada de Unity para añadir texto no es útil para este propósito, al no interactuar bien con la profundidad de otros objetos en 3D. Aquí podemos ver el problema mejor ilustrado. A pesar de que ubicamos el texto detrás de la nave espacial, aún es visible desde cualquier ángulo. Lo ideal sería poder ver el texto sólo cuando este se encuentre en frente del jugador.

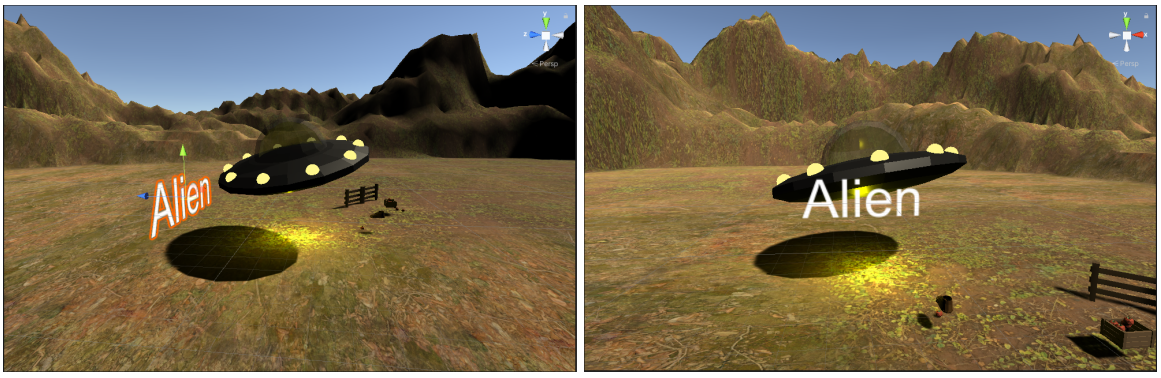


Figura 6.1.1. Texto en 3D visible desde cualquier ángulo.

6.1.1. Uso de librería TextMeshPro

Para solucionar esto, haremos uso de una librería llamada *TextMeshPro*. Afortunadamente, esta librería está preinstalada en Unity y sólo debemos importarla a nuestro proyecto abriendo el menú *Window* y seleccionar las opciones “Importar TMP essentials” dentro de la sección *TextMeshPro:*

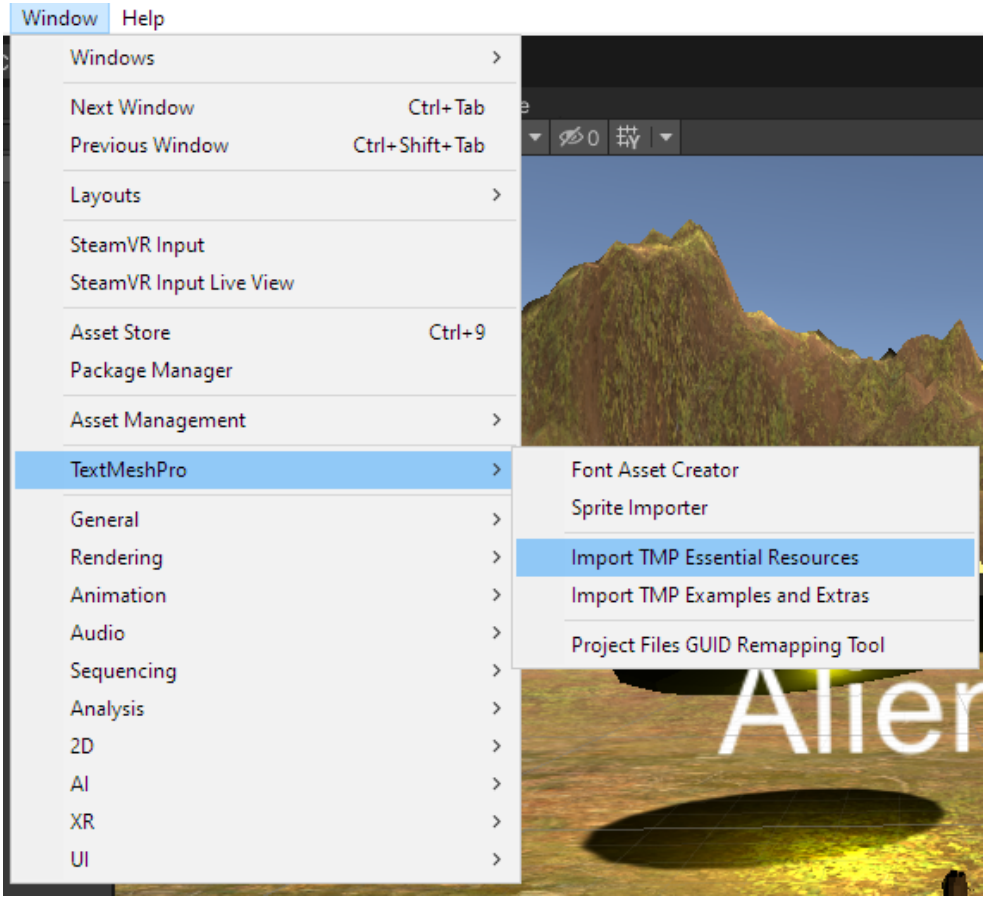


Figura 6.1.1.2. Importar librería TMP Essential

Hacer esto creará el recurso de TextMeshPro dentro de nuestro proyecto. Ahora podremos crear un objeto de texto TextMeshPro en la ventana de objetos 3D primitivos, como lo hemos hecho anteriormente.

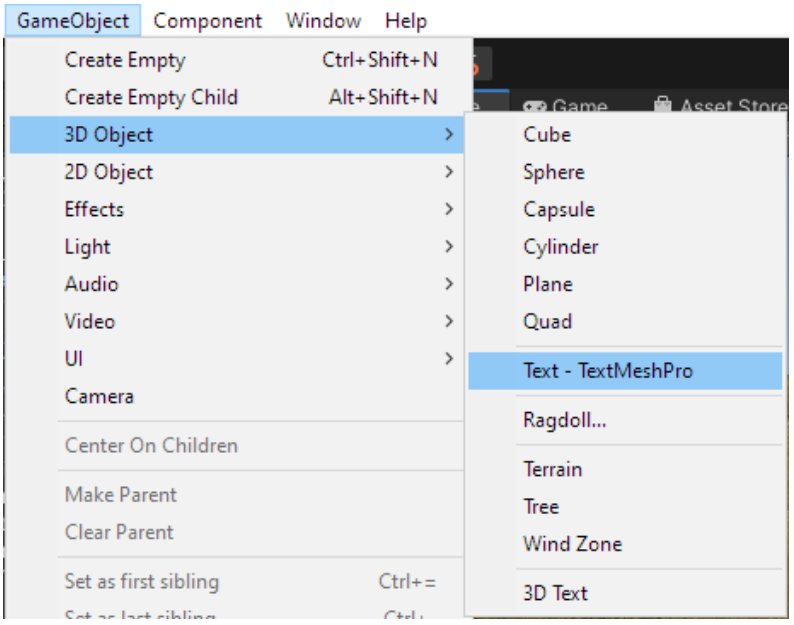


Figura 6.1.1.3. Creación de texto TextMeshPro

El objeto creado se comporta de manera similar al texto 3D tradicional, con la diferencia de que reacciona apropiadamente con la profundidad de otros objetos, como podemos observar a continuación:

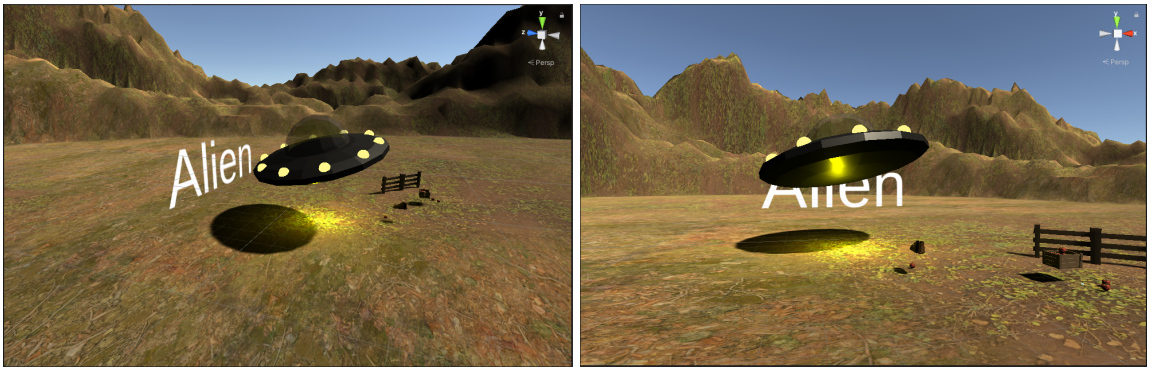


Figura 6.1.1.4. Texto TMP reaccionando apropiadamente en un entorno 3D.

Este objeto de texto tiene dos componentes particularmente importantes, que nos permiten modificar cómo se ve en el ambiente virtual. El primero es el de transformar rectángulo, que nos permite modificar el espacio que el texto va a ocupar en el juego con coordenadas, rotación y tamaño. De esta manera, funciona como un objeto 3D primitivo.

Figura 6.1.1.5. Modificar rectángulo en texto TMP.

El segundo componente es el de texto TMP. Este es el más importante, ya que nos permite modificar el contenido y el estilo del texto. Podemos modificar el estilo, la fuente, el espaciado y la alineación del texto, entre otras opciones. Este componente también es el que usaremos si queremos modificar el texto desde código. Puede apreciar estas opciones a continuación.

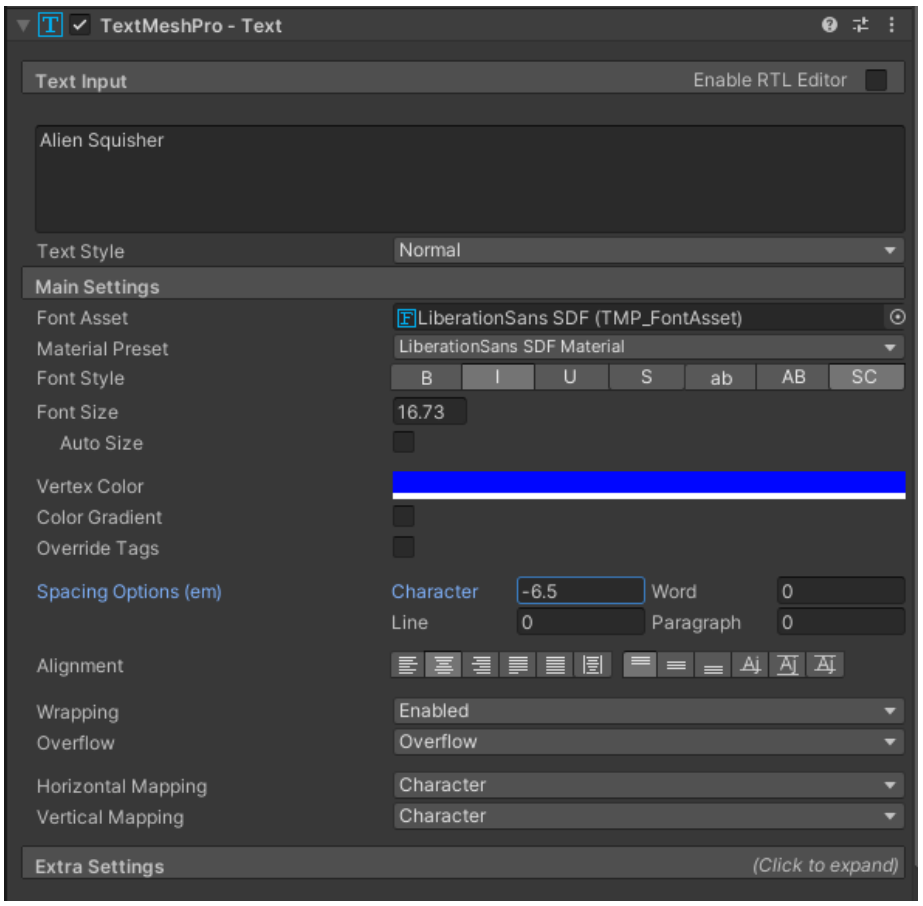


Figura 7.1.1.6. Opciones de texto TMP.

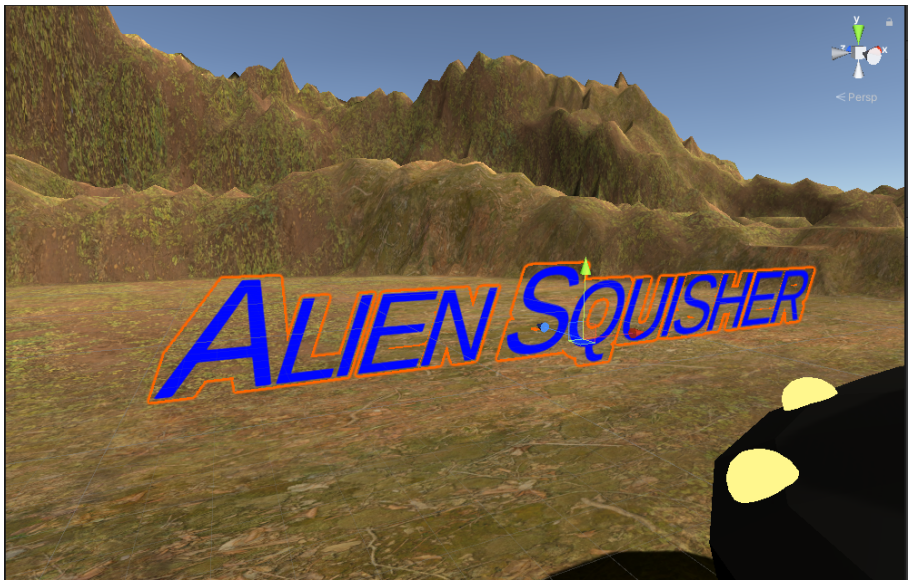


Figura 6.1.1.7. Texto TMP modificado en el entorno 3D.

7.1.2. Modificar texto TMP desde código

Si queremos expresar información específica del juego mediante texto, podemos hacerlo haciendo uso del componente de texto TMP, al vincularlo con el código de nuestro juego. En este caso, queremos que el texto cambie para indicar cuando aplastamos un alien con nuestra bota. Para hacer esto, podemos declarar el texto en nuestro script de

bota y modificarlo cuando interaccione con un alien. Podemos hacer esto asignando un string al atributo *text* del componente.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
```

Figura 6.1.2.1. Importar librería TMPro.

```
//Componente de texto
0 references
public TextMeshPro texto;
```

Figura 6.1.2.2. Declarar texto en el script

```
//Esta función se llamará cada que la bota interactúe con otro objeto
0 references
private void OnTriggerEnter(Collider other){
    if (other.gameObject.CompareTag("F2D5A7_Alien"))
    {
        texto.text = "¡Alien destruído!";
        Destroy(other.gameObject);
    }
}
```

Figura 6.1.2.3. Función modificada para cambiar el texto en colisiones

Ahora agregaremos el objeto de texto a la bota en el editor, y probaremos cómo se cambia al interactuar con aliens.

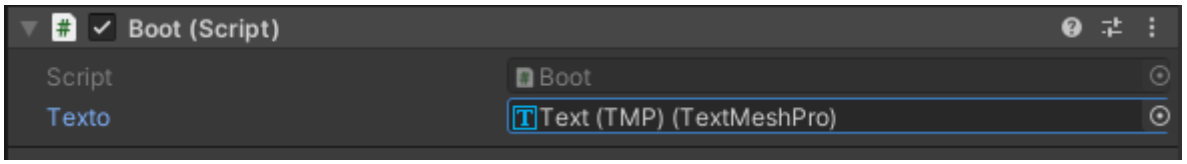


Figura 6.1.2.4. Asignar texto en el componente de la bota.

Figura 6.1.2.5. Texto antes de ser cambiado por el script.

Figura 6.1.2.6. Texto luego de ser cambiado por la interacción en el script.

6.2. Emisores y receptores de audio

Otra pista que podemos darle al jugador de las interacciones es a través de audio. Unity maneja audio en 3D mediante emisores y receptores de audio. Los emisores de audio emiten sonido, como su nombre lo indica, mientras que el receptor de audio simula los oídos del jugador. Esto puede apreciarse al utilizar audífonos, ya que notaremos que los sonidos expresados en el ambiente virtual son simulados como si estuviesen en el ambiente real.

Para añadir emisores de audio, sólo debemos añadirlo en el menú de componentes como se demostró anteriormente. El nombre del componente es *Audio Source*. En este caso, lo añadiremos a la bota para que reproduzca un sonido de aplastar cada que aplaste un alien. No añadiremos los emisores de audio a los aliens, debido a que los estamos eliminando en contacto con la bota. Esto significa que el emisor de audio también desaparecerá y no podrá reproducir el audio completo.

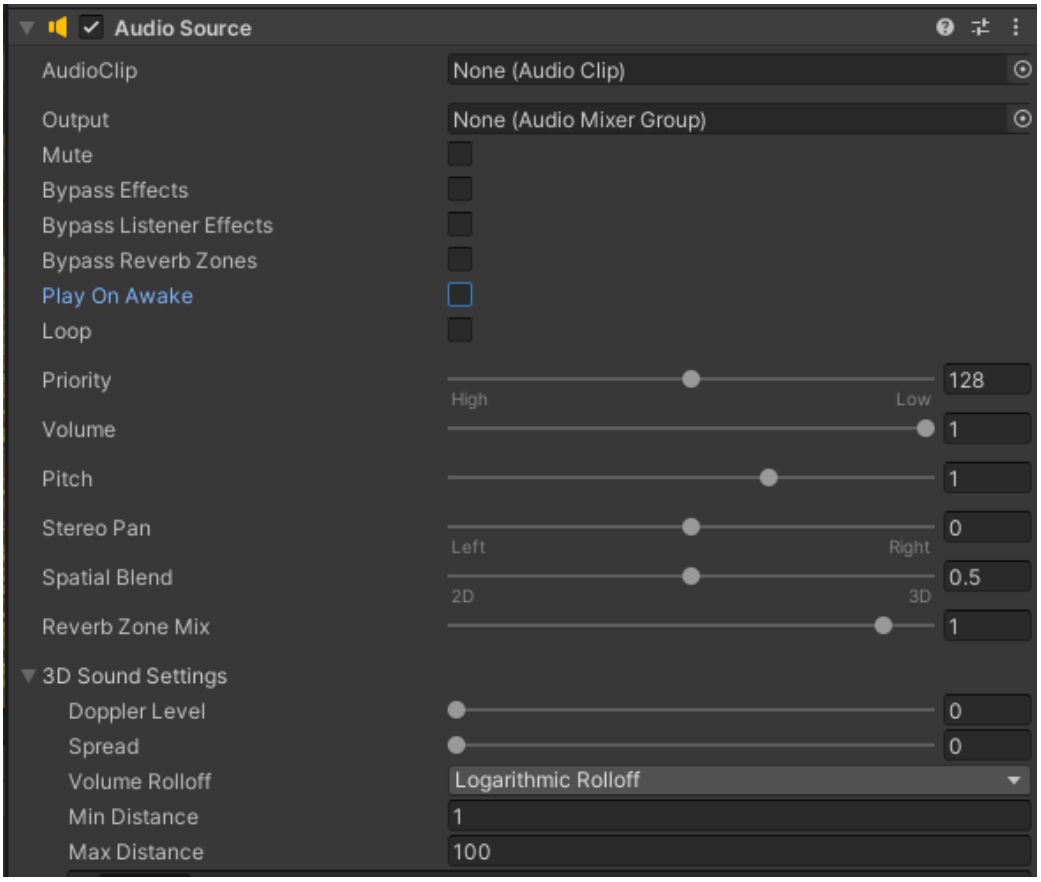


Figura 6.2.1. Opciones de configuración de componente *Audio Source*.

Teniendo este componente, podemos modificar diferentes aspectos del audio, los más importantes son:

- *AudioClip* será el archivo de audio que decidamos reproducir
- *Output* es el canal de audio por el que saldrá el archivo. Se recomienda dejar el canal por defecto.
- Al activar *Mute*, el emisor será silenciado sin importar cuándo reproduzcamos el audio.
- *Play on Awake* determina si el audio se reproducirá cuando el objeto aparezca en la escena.
- *Loop* determina si el audio será reproducido indefinidamente en ciclo, hasta que decidamos detenerlo manualmente.
- *Volume* cambia el volumen del audio.
- *Pitch* cambia la agudeza del audio.
- *Stereo Pan* determina si el audio se reproducirá más en el oído izquierdo o en el derecho. Se recomienda dejar la opción predeterminada.
- *Spatial Blend* determina qué tanto se tratará el emisor de audio como un emisor en 3D. Se recomienda cambiarlo a 0.6 para obtener el efecto de escuchar los sonidos en un ambiente real, sin dejarlo de escuchar por completo al alejarnos de él.

- Bajo la sección *3D Sound settings:*
    - *Doppler Level* nos permite añadir el efecto doppler de sonido al emisor. Este efecto hace que los sonidos suenen más agudos mientras se alejan. Se recomienda cambiarlo a 0.
    - *Min Distance* determina la distancia mínima desde la que podamos escuchar el sonido a su máximo volumen. Se recomienda dejar la configuración predeterminada.
    - *Max Distance* determina la distancia máxima desde la que podamos escuchar el sonido. Podemos cambiarla si queremos que el sonido desaparezca pronto y no cause problemas al alejarse de la escena.

Como queremos que el audio se reproduzca cada vez que la bota haga contacto con algo, es importante que desactivemos *Play On Awake*. De esta manera, el audio sólo se reproducirá cuando queramos. Para el audio, descargamos uno de los recursos al final de este documento. Usaremos uno que suena similar a algo siendo aplastado. Podemos guardar todos nuestros audios en la sección *Sounds* en la carpeta de nuestro juego. Se recomienda renombrarlos al código del minijuego, con un nombre que caracterice qué representa cada audio. En este caso, se utilizará una onomatopeya: **F2D5A7_Squish**.

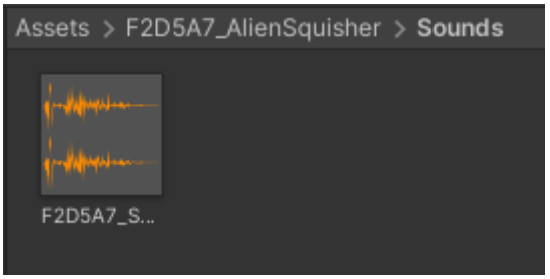


Figura 6.2.2. Representación del archivo de audio en el explorador de archivos.

Ahora, sólo falta asignar el archivo de audio al componente de emisor, y podremos empezar a codificar el comportamiento para que el audio se reproduzca.

Figura 6.2.3. Audio asignado al emisor de audio

6.2.1. Controlar emisor de audio desde script

Ya que el componente de audio también se encuentra en la bota, podemos acceder a él mediante código sin tener que declararlo. Podemos usar el método *GetComponent*<> para utilizarlo:

```
21      //Esta función se llamará cada que la bota interactúe con otro objeto
22      private void OnTriggerEnter(Collider other){
23          if (other.gameObject.CompareTag("F2D5A7_Alien"))
24          {
25              //Activación de componente de audio
26              gameObject.GetComponent<AudioSource>().Play();
27              //Activación de componente de texto
28              texto.text = "¡Alien destruído!";
29              //Destruir alien
30              Destroy(other.gameObject);
31          }
32      }
33  }
```

Figura 6.2.1.1. Modificación de la función para activar componente de audio.

6.4. Desarrollo de la ambientación

6.4.1. ¿Qué es el valle inquietante?

Al ser descubierto muy recientemente, uno de los conceptos más difíciles de comprender durante la creación de videojuegos es el de la liminalidad en las escenas, que producen un efecto de “valle inquietante”. Este efecto indica la emoción negativa de terror o de incomodidad que puede producirse, en este caso, al observar ambientes virtuales que se acercan a uno pero no logran convencer al usuario de que lo son. Esto se debe al poseer distorsiones o desviaciones que lo hacen diferente a un ambiente que se encontraría en la vida real.[3] Debido al enfoque de salud de los videojuegos debemos remover estas distorsiones, ya que hacerlo hará los videojuegos más acogedores y menos incómodos para los usuarios. Estas distorsiones son catalogadas como: Falta, repetición, ubicación inusual o tamaño inusual de los objetos, falta de control sobre la escena (acceso a salidas visibles o higiene del lugar) y la carencia de personas. Esta última depende de si nuestra escena es al aire libre o a puertas cerradas.[4]

6.4.2. Cómo eliminar la sensación de valle inquietante

Para eliminar la sensación de incomodidad en nuestra escena, es importante que añadamos elementos de decoración adicionales luego de finalizar la parte funcional de nuestro videojuego y cambiemos parte de la estética del minijuego. Actualmente, la escena para Alien Squisher se ve de la siguiente manera:

---

[3] Diel, A., & Lewis, M. (2022). Structural deviations drive an uncanny valley of physical places. Journal of Environmental Psychology, 82, 101844. https://www.sciencedirect.com/science/article/pii/S0272494422000895

[4] Diel, A., & Lewis, M. (2022). Structural deviations drive an uncanny valley of physical places. Journal of Environmental Psychology, 82, 101844. https://www.sciencedirect.com/science/article/pii/S0272494422000895

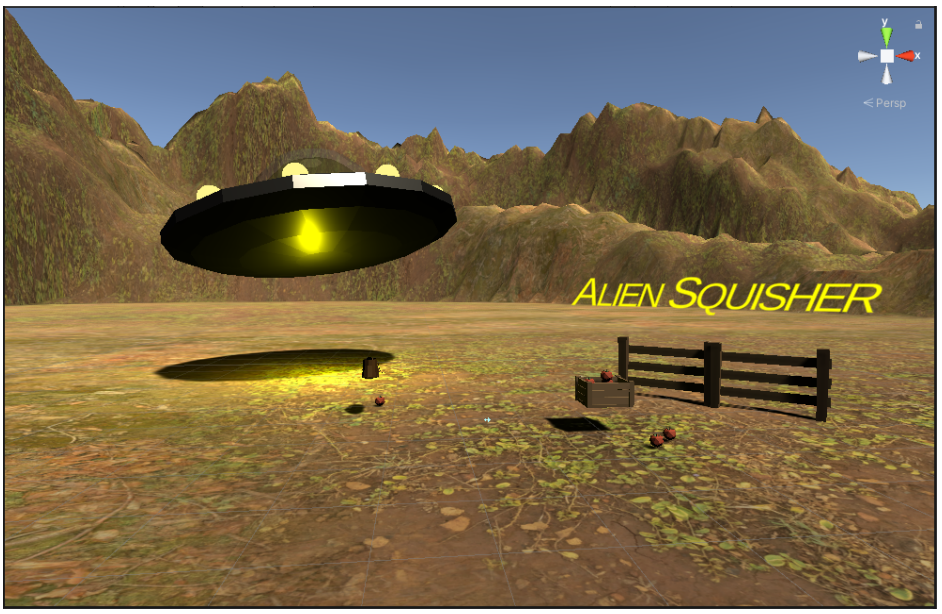


Figura 6.4.2.1. Escena de Alien Squisher.

El cambio de la escena se dará en los siguientes pasos.

6.4.2.1. Añadir un mejor fondo para la escena

El fondo está compuesto del terreno y el cielo. El terreno que utilizamos en la sección anterior es uno generado por Unity, que contrasta con el resto de recursos Low Poly que hemos utilizado para los demás objetos. Lo primero para eliminar este contraste será cambiarlo por un recurso Low Poly que contenga terrenos, o generar un terreno Low Poly desde 0. En este caso, se utilizará el terreno contenido en el paquete "*Low Poly Simple Nature Pack*" en la tienda de assets de Unity.

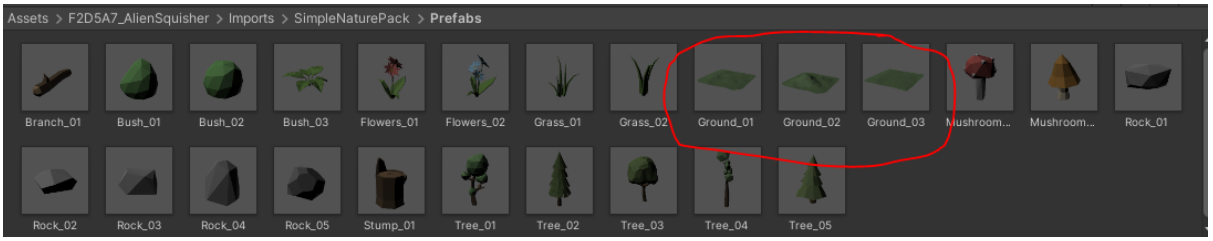


Figura 6.4.2.1.1. Assets de terreno Low Poly.

Así se ven luego de haberlos acomodado y ajustado en términos de tamaño en la escena:

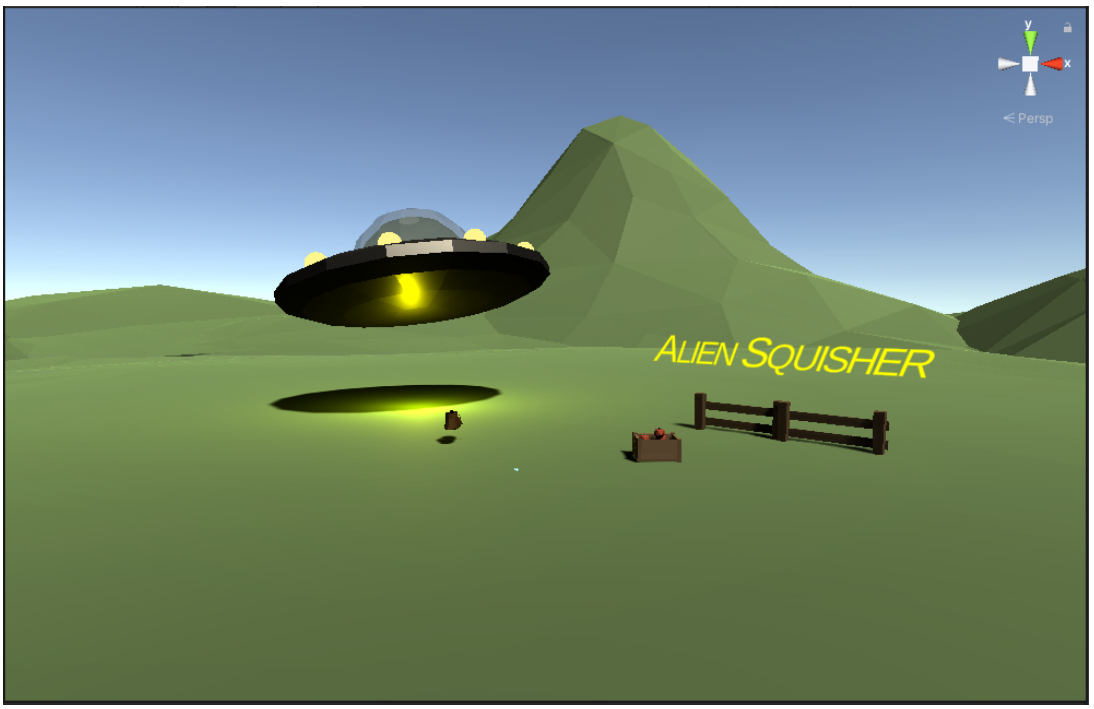


Figura 6.4.2.1.2. Nuevo terreno Low Poly en la escena.

El siguiente paso es el cielo, que al ser generado automáticamente por Unity, siempre tiene la misma gradiente de colores. Si queremos dar un ambiente más acogedor a la escena, podemos cambiarlo por un cielo que encaje mejor con la temática de la escena. En este caso se utilizará el recurso "*Customizable Skybox*" en la tienda de recursos, ya que nos da control sobre cómo queremos que se vea el cielo en nuestra escena.

Figura 6.4.2.1.3. Ejemplos de materiales posibles para reemplazar el cielo en la escena.

Como con cualquier material, podemos arrastrar el material deseado y soltarlo en el cielo actual. Ahora la escena se ve de la siguiente manera:

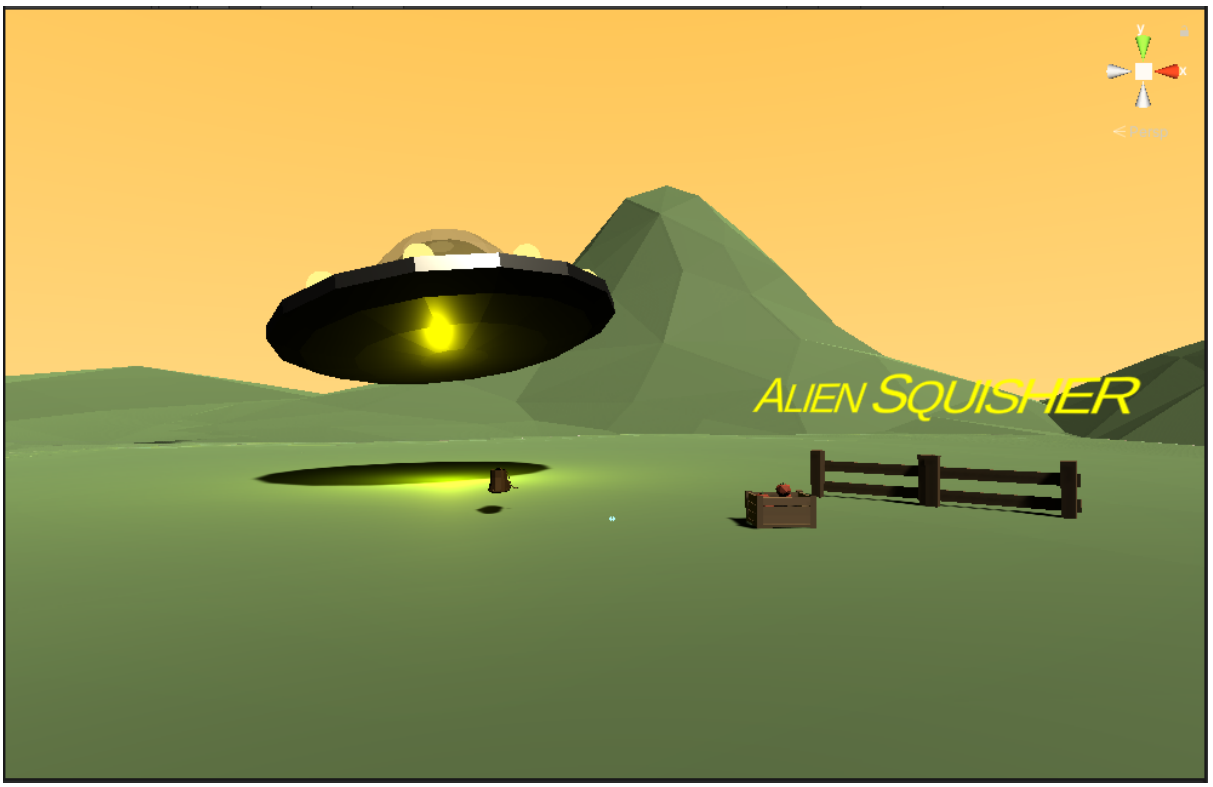


Figura 6.4.2.1.4. Escena luego de modificar el cielo.

6.4.2.2. Añadir decoraciones

Luego de haber completado el fondo del juego, ahora hace falta añadir elementos visuales que complementen la escena y ayuden a que no se sienta tan vacía. Todos estos elementos son decorativos.

Estas decoraciones incluyen objetos que podamos encontrar en la escena naturalmente. Por ejemplo, ya que Alien Squisher toma lugar en una granja, podemos agregar árboles y vallas en el fondo. Podemos usar estos elementos de los recursos que hemos descargado previamente, y organizarlos alrededor de la escena:

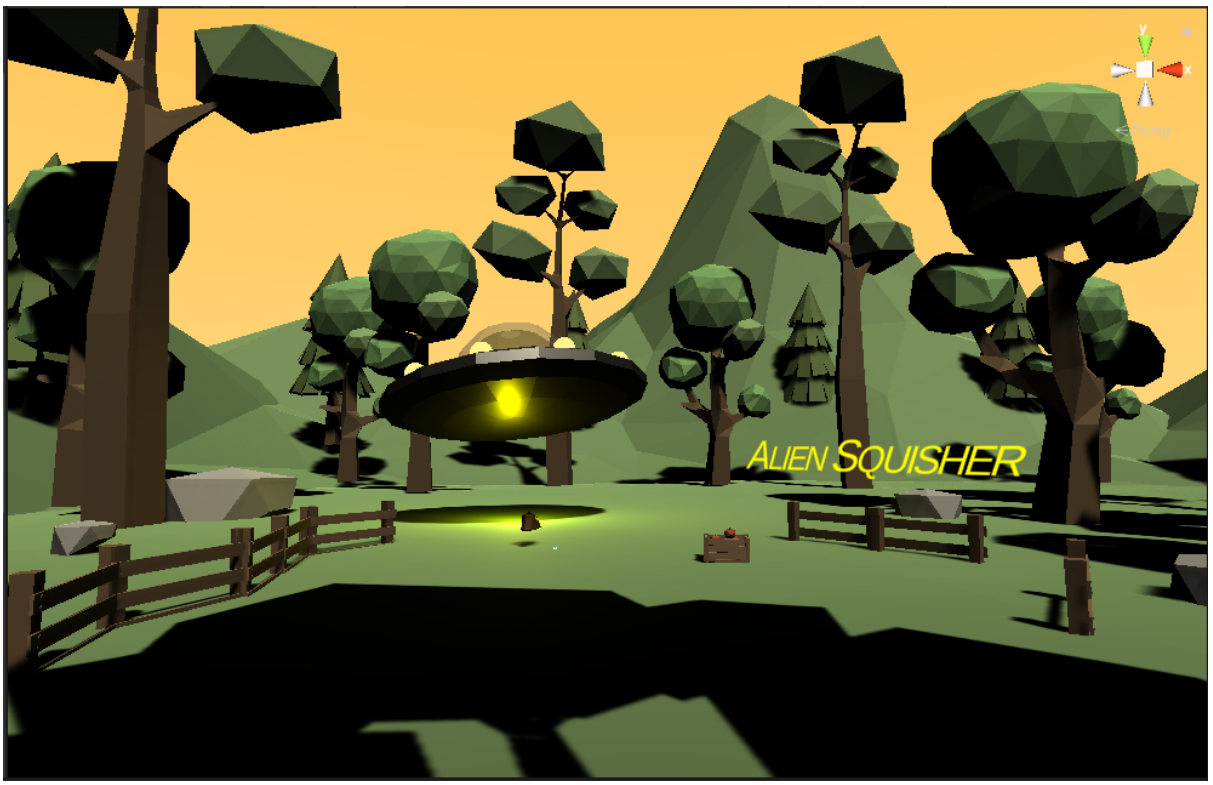


Figura 6.4.2.2.1. Adición de elementos decorativos

Otro de los efectos negativos que podemos encontrarnos referentes al valle inquietante, es la aparición de objetos de tamaños inusuales o repetitivos. Por ende es importante eliminar elementos que parezcan muy idénticos entre sí, y ajustar el tamaño de los que parezcan muy grandes o muy pequeños:

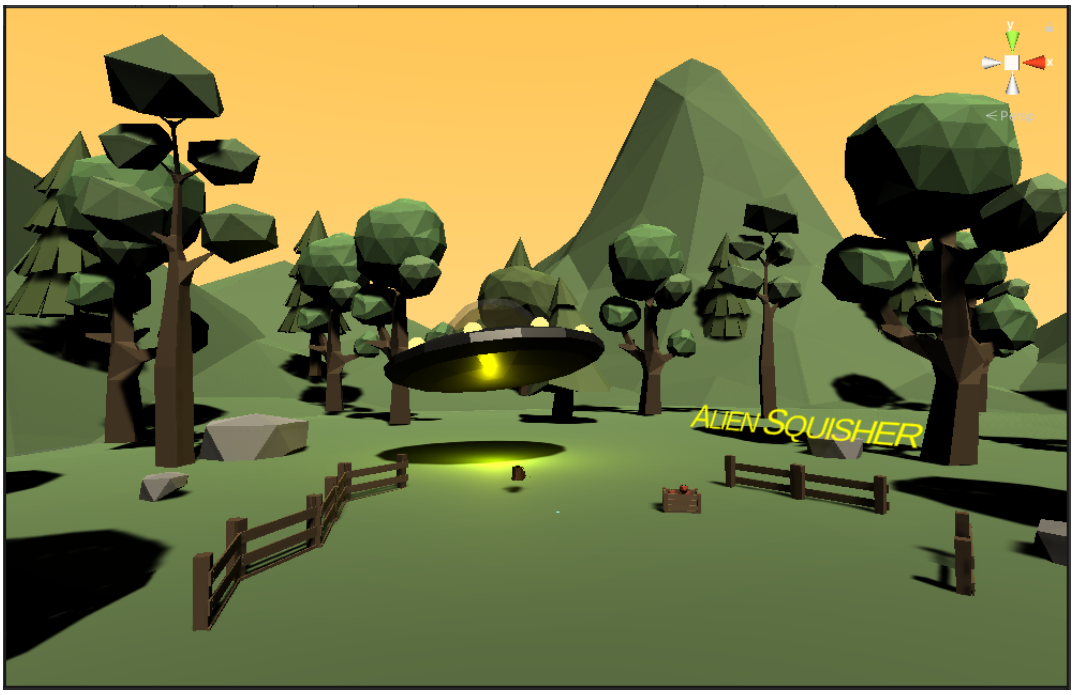


Figura 6.4.2.2.2. Modificación del tamaño de los objetos decorativos.

Finalmente, para editar luego los demás objetos decorativos de la escena, es buena práctica hacer uso de la jerarquía de los objetos para categorizarlos y esconderlos cuando haya falta trabajar con ellos:

Figura 6.4.2.2.4. Organización de objetos decorativos.

6.4.2.3. Añadir una iluminación natural

Unity presenta un problema en el que al añadir sólo una fuente de luz, se generan sombras fuertes que no se encuentran en la vida real. Como puede verse en la escena anterior, la nave espacial y los árboles oscurecen completamente el césped debajo. Buscamos que la iluminación tenga tres puntos de luz, para que todos los objetos se encuentren perfectamente iluminados.

El primer punto de luz representaría entonces la luz del sol, y por ende será el más fuerte. Esta luz es la que incluye Unity por defecto, pero podemos cambiarla para representar un ambiente de atardecer, un ambiente nocturno o cualquier otro escenario.

Para modificar la luz, podemos seleccionar la luz a modificar y editar sus propiedades en el inspector.

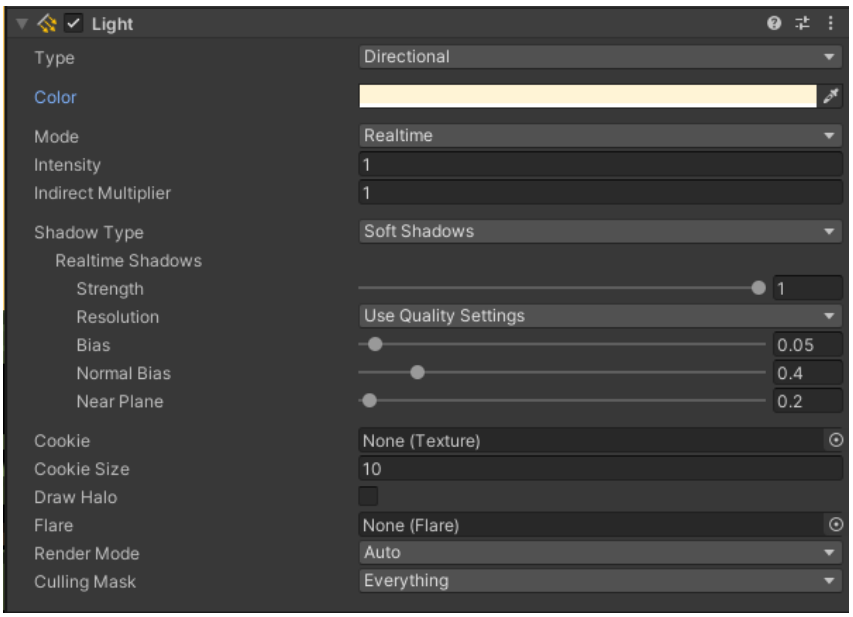


Figura 6.4.2.3.1. Propiedades de componente Luz

La luz primaria debe tener un color que se asemeje al del cielo, por lo que podemos elegir cambiar el color por defecto seleccionando la propiedad de color y presionando el botón del gotero que está a la derecha del color para elegir el color del cielo:

Figura 6.4.2.3.2. Elección de nuevo color para la luz primaria

Así se ve la escena con el cambio a nuestra luz primaria:

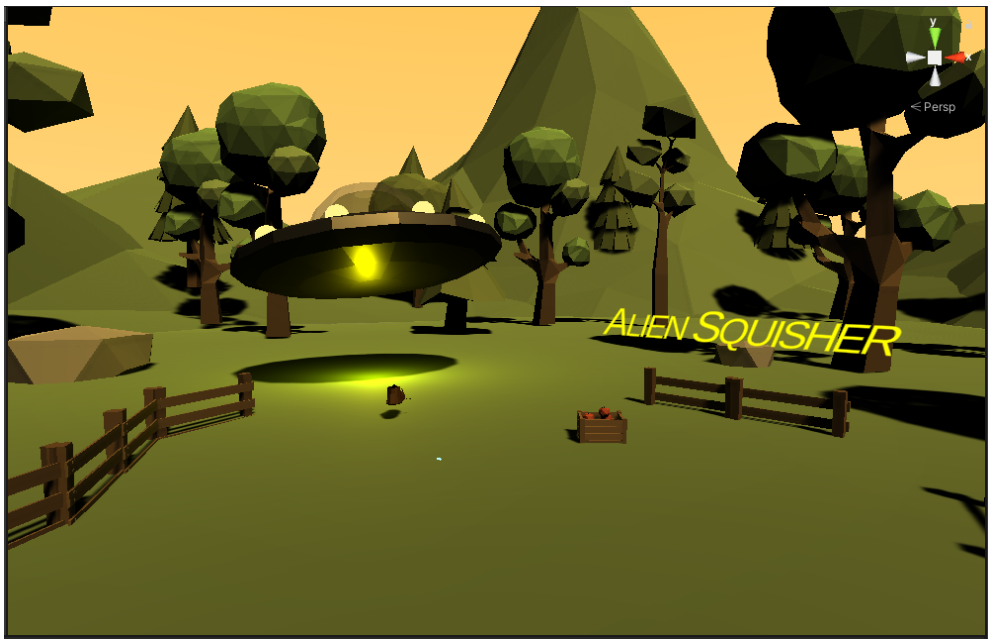


Figura 6.4.2.3.3. Escena luego del cambio de luz.

El segundo punto de luz vendría del reflejo del sol con otros objetos como ventanas o edificios. Para crearla, podemos dar clic derecho a la luz original en nuestra escena y elegir la opción de duplicar:

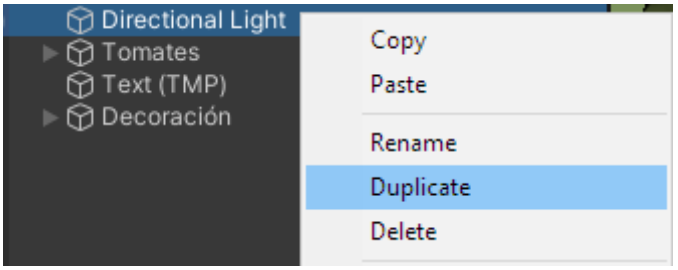


Figura 6.4.2.3.4. Duplicar luz creada por Unity.

Para la luz secundaria, se debe rotar en Y en la dirección contraria a la del sol. Podemos hacer esto fácilmente con la herramienta de rotar, ya que se verá un semicírculo cuando lleguemos a la dirección opuesta:

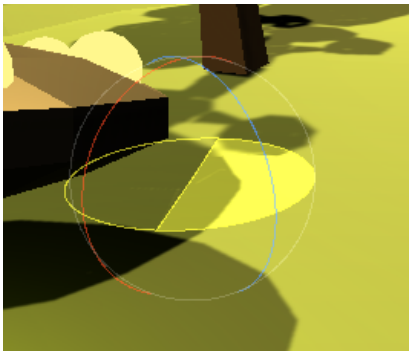


Figura 6.4.2.3.5. Rotación de la luz secundaria.

Esta luz no puede ser exactamente igual a la primaria, así que modificaremos los valores de intensidad para que sea la mitad de la luz primaria. Adicionalmente, vamos a

remover la opción de generar sombras, ya que no son deseadas para este caso y generan complejidad computacional innecesaria.

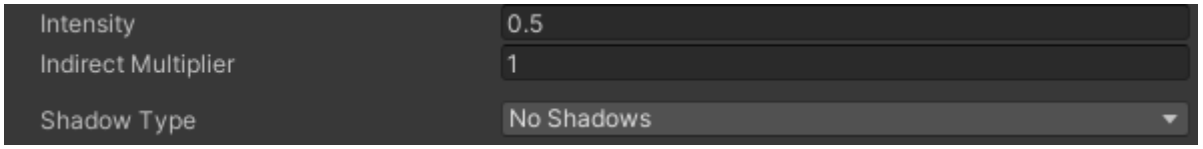


Figura 6.4.2.3.6. Modificación de la luz secundaria.

Para la luz terciaria, sólo debemos ponerla en la posición opuesta en el eje Z, de manera que esté direccionada hacia arriba. La intensidad también debería disminuir, siendo sólo la mitad de la luz secundaria. Además, es recomendable cambiar el color a uno más frío, ya que es el que se opone a la luz del sol en la vida real. Para simplificar, podemos elegir un púrpura, ya que naturalmente contrasta la luz del sol. Por último, debemos cambiar el modo de renderizado (*Render Mode)* a "No importante". Esto se debe a que no queremos que la luz terciaria cambie cómo se ve la parte ya iluminada de la escena, sólo las sombras.

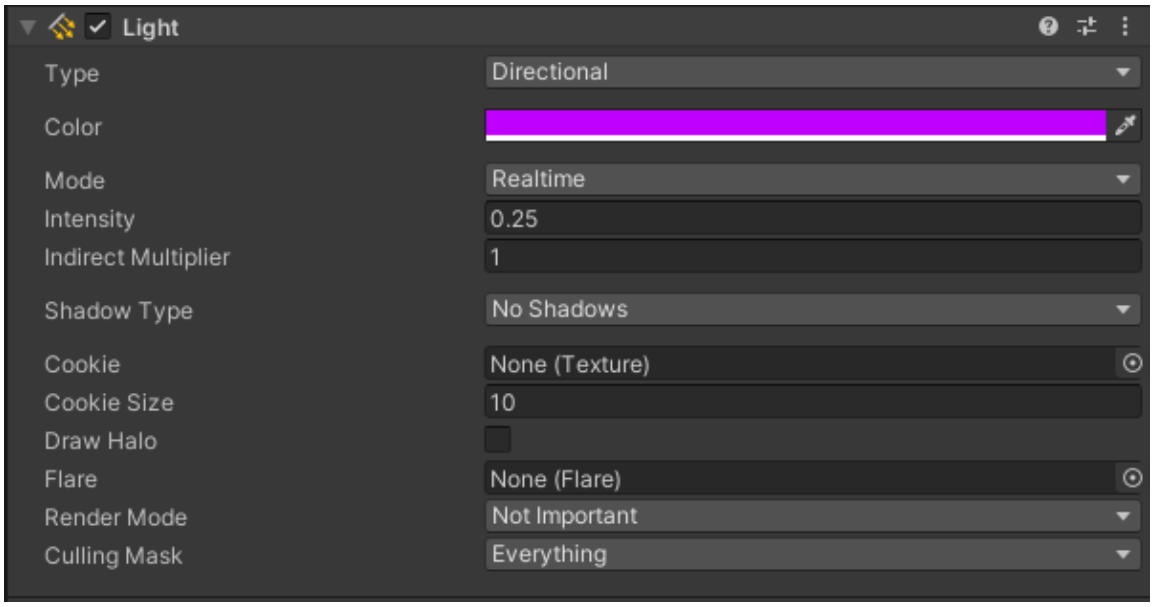


Figura 6.4.2.3.7. Propiedades de luz terciaria.

Así se ve la escena luego de añadir las dos luces adicionales, eliminando las sombras fuertes y los cambios drásticos de color:

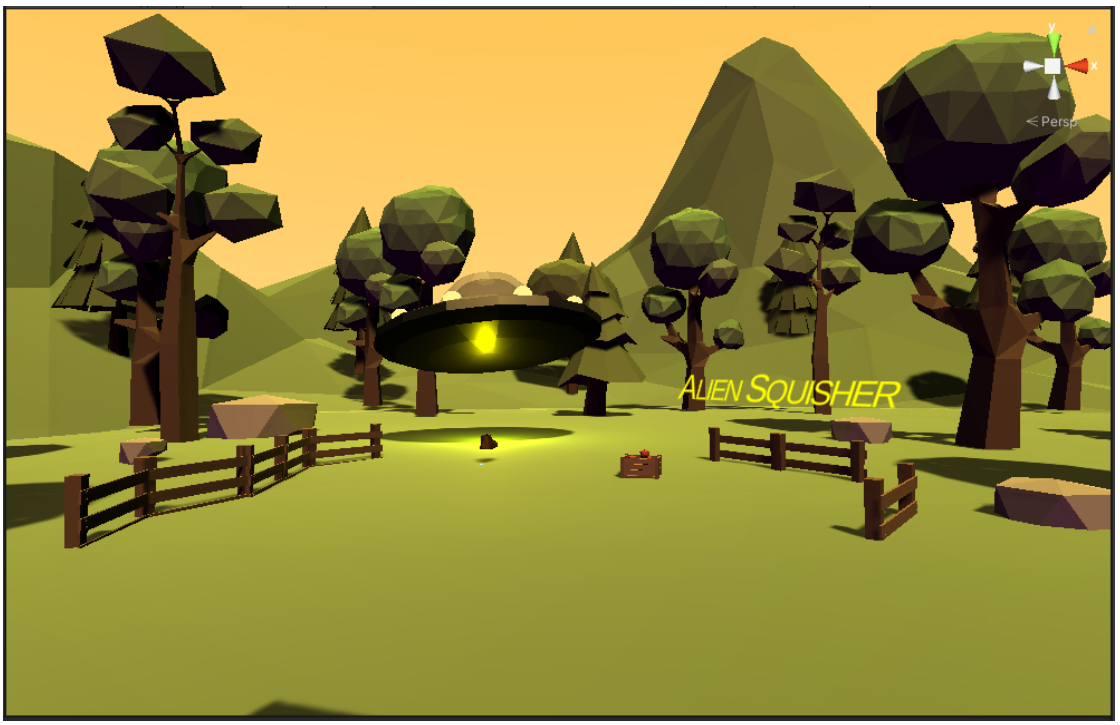


Figura 6.4.2.3.8. Escena con los cambios en la iluminación

6.4.2.4. Añadir 'NPCs'

Los NPCs o "*Non-Playable Characters*" son personajes que acompañan la escena pero que el jugador no puede controlar. Los NPCs ayudan a sentir la escena más poblada y menos peligrosa, aunque es necesario elegir cuándo añadirlos a las escenas y en qué medida. Por recomendación general, los NPCs deberían estar en escenas al aire libre, y deberían estar haciendo una actividad característica de la escena en la que se encuentran. Por ejemplo, ya que Alien Squisher toma lugar en una granja, podemos incorporar a un granjero en la escena que acompañe al jugador. Para este caso, se utilizará el recurso "*Distant Lands Free Characters*" en la tienda de recursos de Unity.

*Al descargar personajes de la tienda de recursos, se recomienda buscar personajes que tengan un esqueleto, ya que podremos modificar su movimiento más adelante. Podemos encontrarlos con el atributo en inglés '*Rigged*'.

Figura 6.4.2.4.1. NPC incorporado en la escena.

Como podemos ver, este personaje no se ve natural en la escena. Esto se debe a que aún no tiene una posición natural, y parece rígido. Para solucionar esto, podemos acudir a una propiedad de algunos recursos de personajes en Unity, los esqueletos. Podemos identificar el esqueleto de un personaje al observar los objetos en su jerarquía. En este caso, el NPC posee un objeto llamado 'Male' que contiene otros objetos describiendo sus partes del cuerpo. Cada uno de estos es un 'hueso' que controla una parte del personaje, y conforma un esqueleto que podemos modificar para controlar la pose del personaje.

Figura 6.4.2.4.2. Esqueleto del NPC.

Como en este caso el NPC no es muy importante en la escena, podemos modificar el esqueleto manualmente para que esté en la posición que queramos. En este caso, podemos hacer que esté cruzado de manos. Para hacer esto, podemos expandir el esqueleto para rotar los huesos que queramos:

Figura 6.4.2.4.3. Rotación de los huesos del NPC

Podemos repetir este proceso con los demás huesos del NPC para conseguir una posición natural:

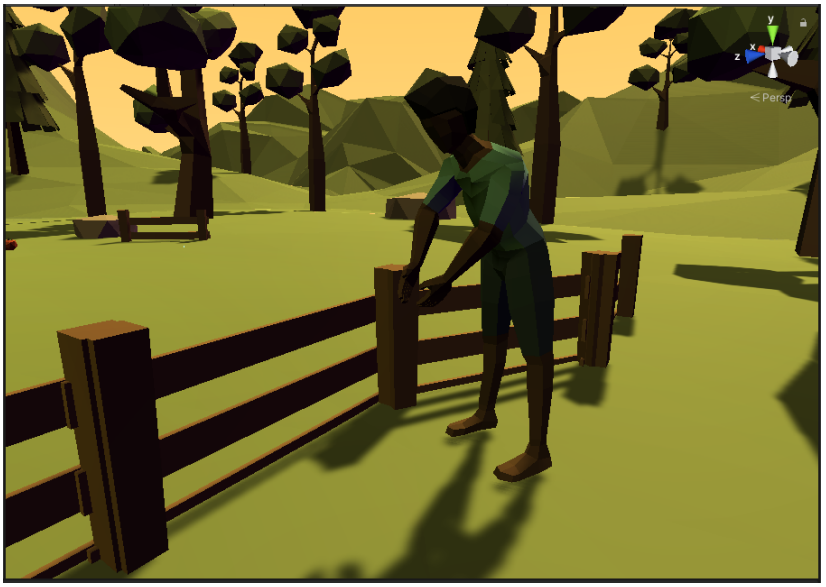


Figura 6.4.2.4.4. NPC con posición de granjero.

Podemos complementar el NPC añadiendo elementos característicos importándolos de otros recursos, como lo puede ser una azada en sus manos o un sombrero. Como vimos anteriormente, podemos usar la jerarquía de objetos en Unity para hacer que la azada y el sombrero se queden en su manos y cabeza respectivamente. Podemos hacer esto arrastrando los objetos agregados al hueso de la parte del cuerpo deseada, y luego modificándolo para que se vea natural:

Figura 6.4.2.4.5. Adición de objetos adicionales al NPC.

Figura 6.4.2.4.6. NPC con el objeto adicional en sus manos.

6.4.2.5. Añadir animaciones a personajes

Si queremos obtener más control sobre la pose de un personaje, podemos descargar animaciones adicionales de la tienda de recursos. Estas animaciones permiten que los personajes puedan moverse independientemente, sin que nosotros tengamos que alterar manualmente su esqueleto en la escena. De esta manera, el NPC siempre está moviéndose en la escena, haciéndolo parecer más natural y con vida.

Para verificar si nuestro personaje acepta animaciones personalizadas, podemos ir al objeto prefabricado y verificar que cuente con el componente de animador '*Animator*'. A este componente le podremos agregar un controlador de animaciones que será el que controle el comportamiento del NPC.

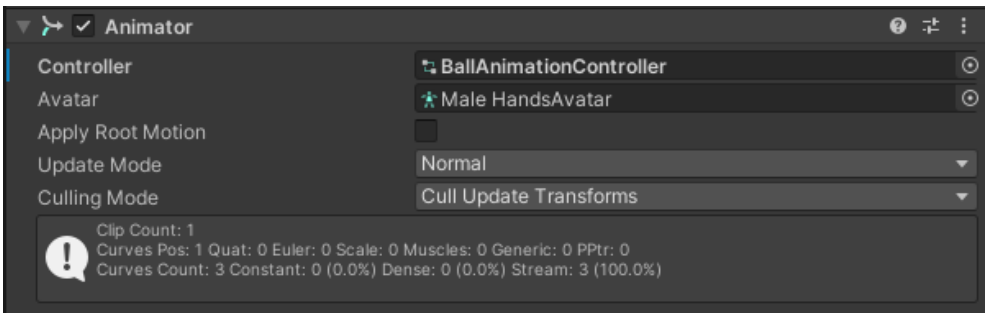


Figura 6.4.2.5.1. Componente de animador en el objeto de NPC.

Podemos reemplazar el atributo controlador '*Controller*' al descargar uno adicional de la tienda de recursos. Para este caso, se descargó el recurso "*Villager Animations FREE*" de la tienda de Unity. Ahora podemos cambiar el controlador para asignar distintos comportamientos al NPC.

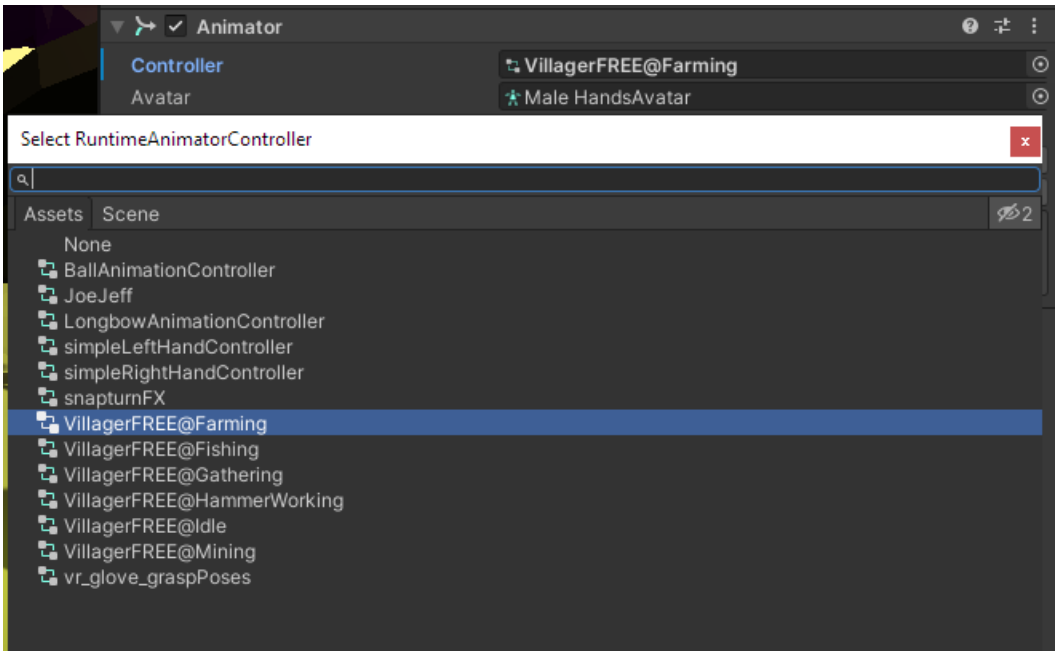


Figura 6.4.2.5.2. Cambio de controlador de animaciones en el personaje.

Ahora podemos ver al NPC moviéndose independientemente cuando iniciamos el minijuego.

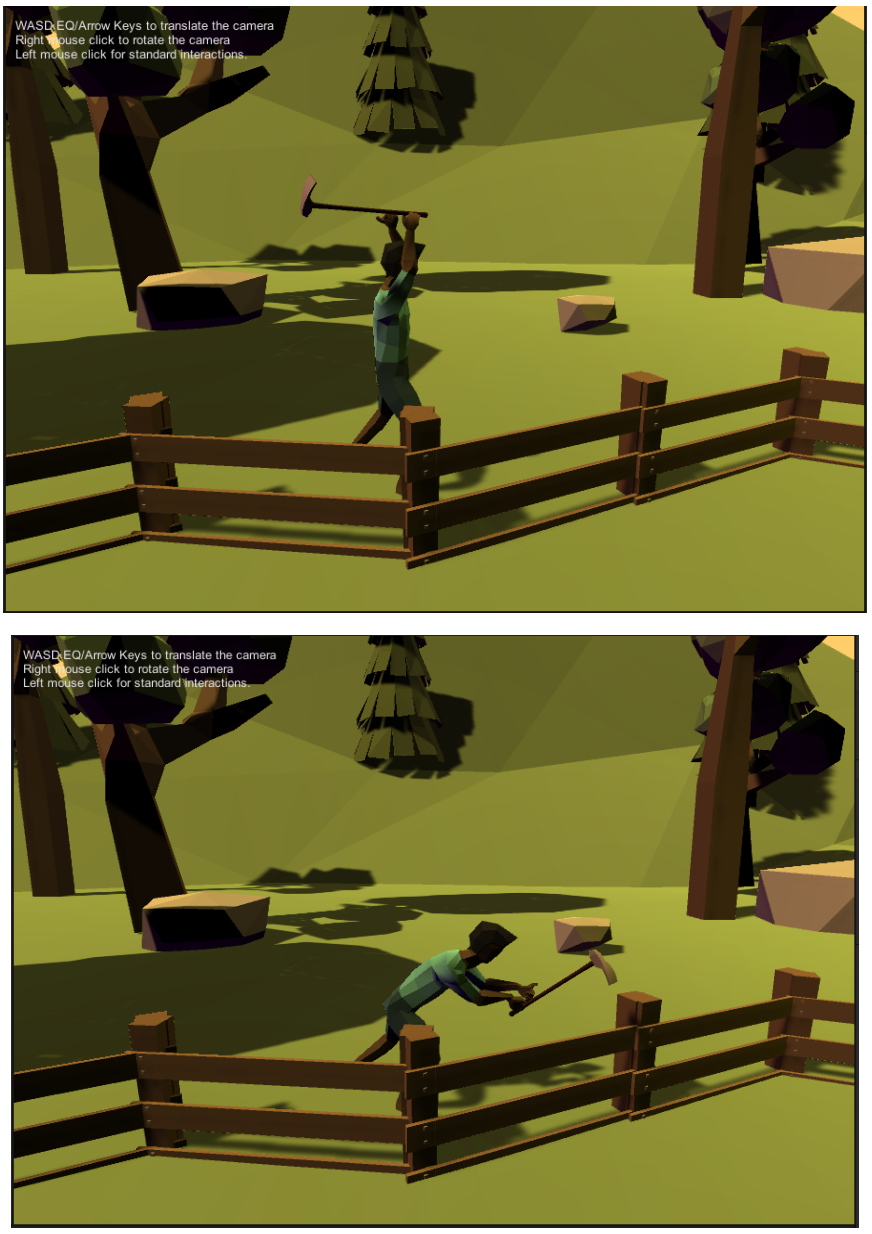


Figura 6.4.2.5.3. NPC moviéndose independientemente en la escena.

6.4.3. Añadir sonidos de ambientación.

Finalmente, podemos usar un emisor de audio para hacer la escena menos vacía auditivamente. Se recomienda buscar audios que tengan una duración de más de 30 segundos, pero que no tengan mucho peso, puesto que harán nuestro juego más pesado. Lo ideal es encontrar un audio que represente bien la escena. En este caso se buscará un audio con sonidos de pájaros, viento y otros detalles que harán sentir al jugador más cómodo en la escena. Este paso tiene la ventaja adicional de que el sonido del viento hará sentir al jugador como si estuviese al aire libre en lugar de en un espacio cerrado. Se descargó el audio "*Forest Ambient Loop*" por *rolandasb*.

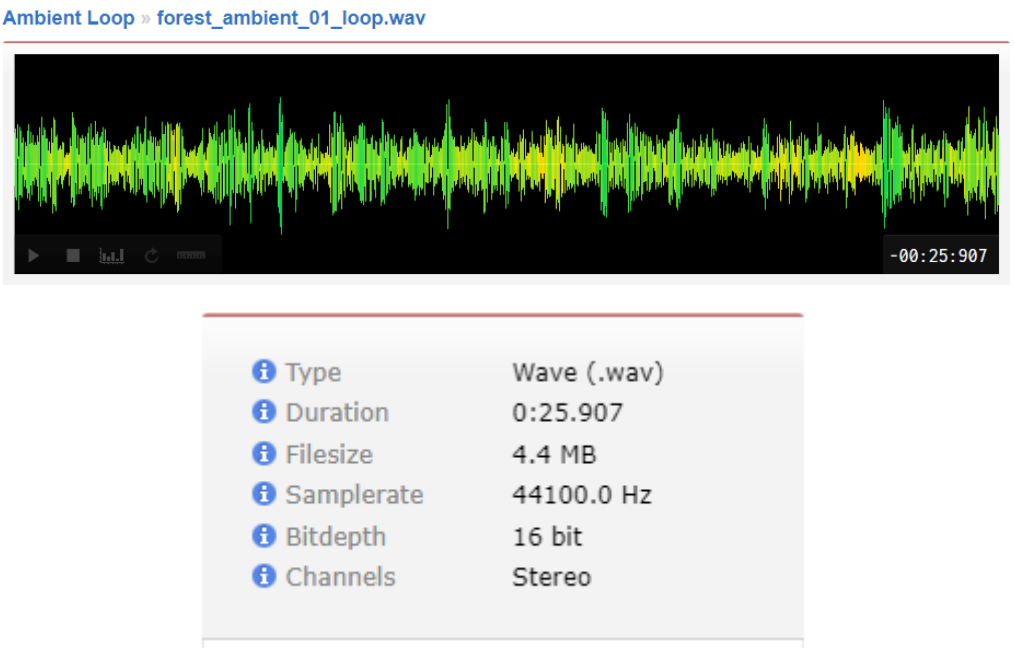


Figura 6.5.3.1. Sonido seleccionado para el ambiente.

Para incorporarlo en la escena, sólo hace falta crear un objeto vacío que represente la ambientación de la escena. Como vimos anteriormente, añadiremos un componente de audio que contenga el archivo que descargamos. La diferencia está en que como el sonido proviene de todos lados, desactivaremos la opción de 3D del emisor de audio. De esta manera el jugador siempre podrá escuchar la ambientación a igual volumen. Adicionalmente, reduciremos el volumen para que no interrumpa otros sonidos importantes del videojuego, y activaremos *Loop* y *Play on Awake* para que la ambientación sea persistente durante todo el juego.

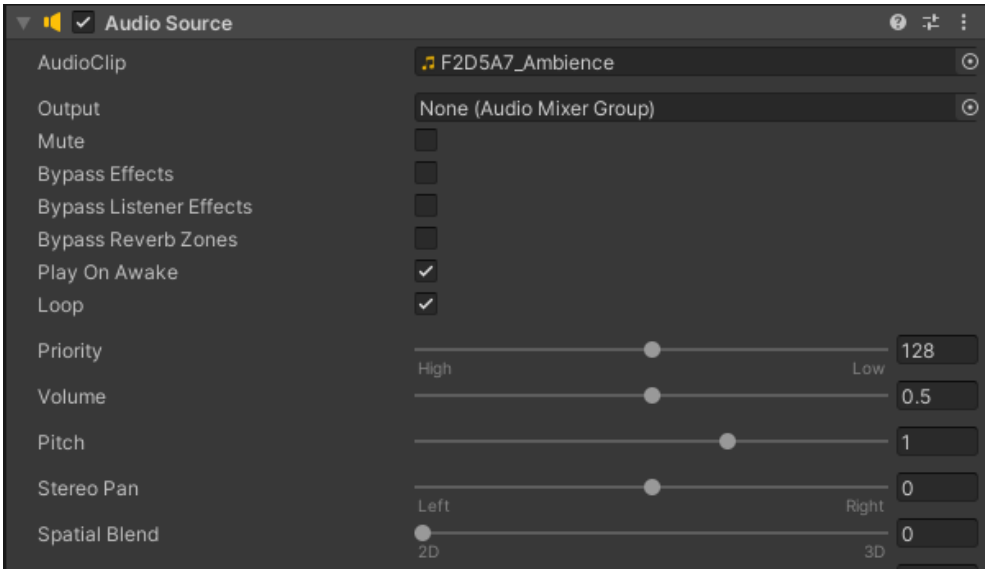


Figura 6.4.3.2. Configuración de emisor de audio de ambientación.

7. Sistemas de puntuación

Para dar un sentido de progresión en el minijuego, es importante que exista un sistema que lleve cuenta de cuántas veces el jugador ha realizado la actividad, cuántas veces ha fallado y cuántas veces la ha realizado correctamente. Por ende, es importante implementar un sistema de puntuación en nuestros minijuegos. Con el sistema de puntuación, podemos definir en qué momento el jugador gana o pierde el minijuego, o cómo se compara contra sus demás intentos. A continuación podrá ver distintos sistemas de puntuación.[5] Alien Squisher utilizará una combinación de contador de puntos con contador de vidas, pero también podrá leer acerca de otros modelos de puntuación para elegir el que mejor se adapte al minijuego pensado.

7.1. Contador de puntos

Este sistema de puntuación es el más sencillo, verificando cuando el jugador realiza una actividad al agregar puntos a un contador, y quitando puntos cuando realiza un error. El minijuego se gana o se pierde cuando el jugador llega a un número de puntos. Por ejemplo, el jugador gana al llegar a 10 puntos o pierde al llegar a 0 puntos.

Esta puntuación también puede representarse con un porcentaje, una barra de progreso, u otros medios visuales. También puede convertirse en un sistema de puntos con recolección de objetos, en el que el jugador tenga que tomar distintos objetos de sus actividades para completar el juego.

7.1.1. Preparación de otros elementos del juego

Antes de empezar a implementar el contador de puntos, será necesario añadir un par más de elementos a la escena para poder tener todos los elementos que la ficha técnica promete. El primero es añadir dos naves espaciales adicionales, junto con dos tomates adicionales, para que los aliens puedan aparecer desde múltiples direcciones y atacar a los distintos tomates. Duplicar los elementos basta para conseguir este paso:

[5] MakeUseOf (2022). 11 Ways Progression Works in Video Games. https://www.makeuseof.com/video-game-progression-ways/.

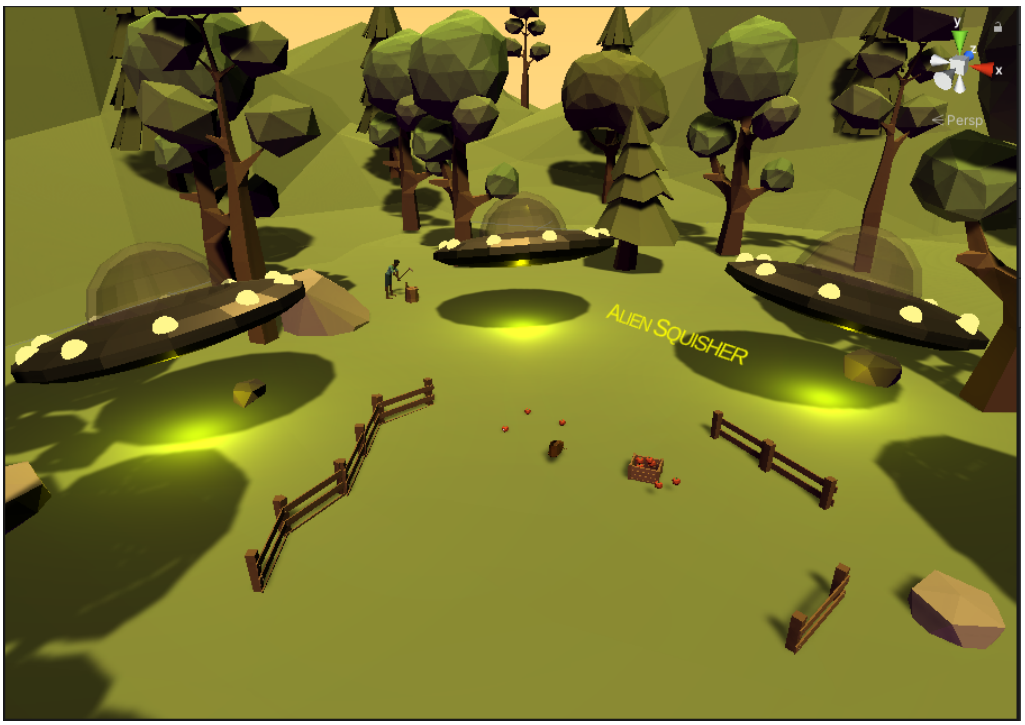


Figura 7.1.1.1. Escena con los tomates y las naves espaciales duplicadas

Teniendo varias naves espaciales, podemos modificar los tiempos de cada una para variar la dificultad del juego, y no tener a los aliens completamente sincronizados. Además, podremos adicionar una variable booleana para determinar si las naves seguirán creando aliens o no.

```
//Temporizador para la creación de Aliens
1 reference
public float frecuenciaDeAliens = 10.0f;
1 reference
public float tiempoDeEspera = 0.0f;
4 references
float temporizador;
//Determina si la nave continúa creando aliens
0 references
public bool continua;
```

Figura 7.1.1.2. Creación de variables para el control de aliens en el script de nave espacial

En este caso, la *frecuencia de aliens* controla el tiempo que pasa luego de crear un alien hasta crear el siguiente. El *tiempo de espera* determina el tiempo que dura cada nave al iniciar el juego sin crear un alien. Y *continúa* determina si la nave sigue creando aliens. Siguiendo estas variables, vamos a remover la creación inicial de alien y en cambio asignar el tiempo de espera al temporizador.

```
// Esta función se llama antes del primer fotograma
0 references
void Start()
{
    temporizador = tiempoDeEspera;
}
```

Figura 7.1.1.3. Cambio a la función de inicio.

Adicionalmente, añadiremos un condicional que sólo descuente tiempo del temporizador mientras la nave continúe generando aliens. De esta manera, al desactivar la condición *continúa*, la nave dejará de crear aliens.

```
// Esta función se llama cada fotograma
0 references
void Update()
{
    if(continua) descontarTiempo();
}
```

Figura 7.1.1.4. Cambio a la función de actualizar.

Teniendo estos cambios, podremos configurar las naves espaciales para que se alternen en la generación de aliens e inicien con la opción de continuar habilitada.

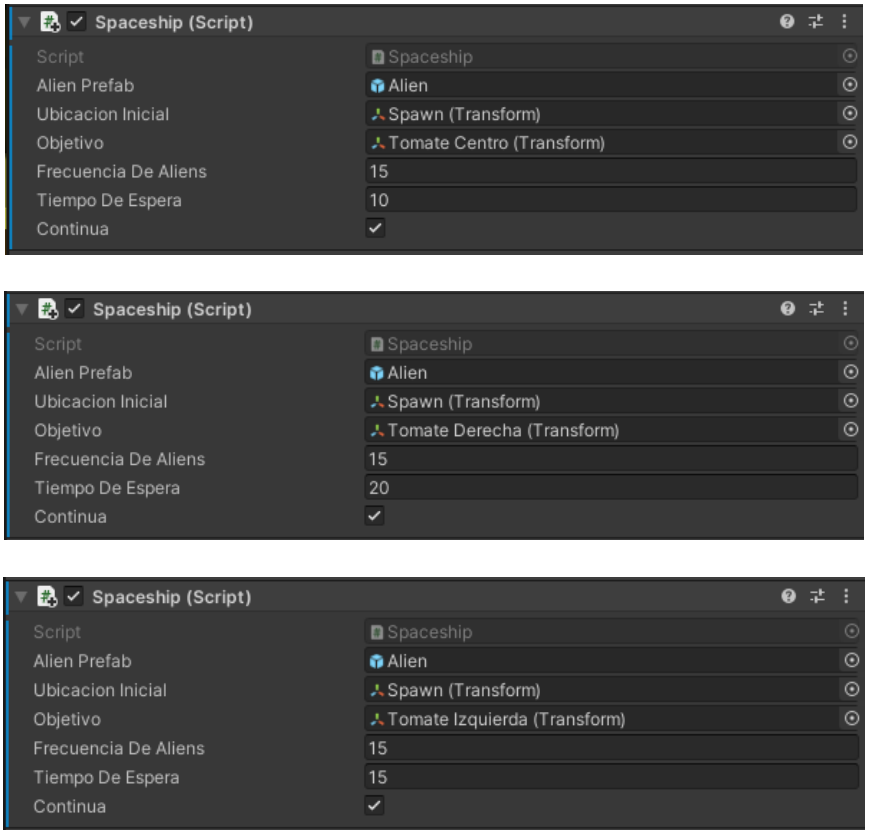


Figura 7.1.1.5. Configuración para los Aliens

7.1.2. Creación de un contador de puntos

Lo primero que haremos para implementar el sistema de puntuación en el videojuego será crear un componente vacío que contenga el script para manejar los puntos. Como este objeto sólo maneja aspectos abstractos, no es necesario que tenga otros componentes.

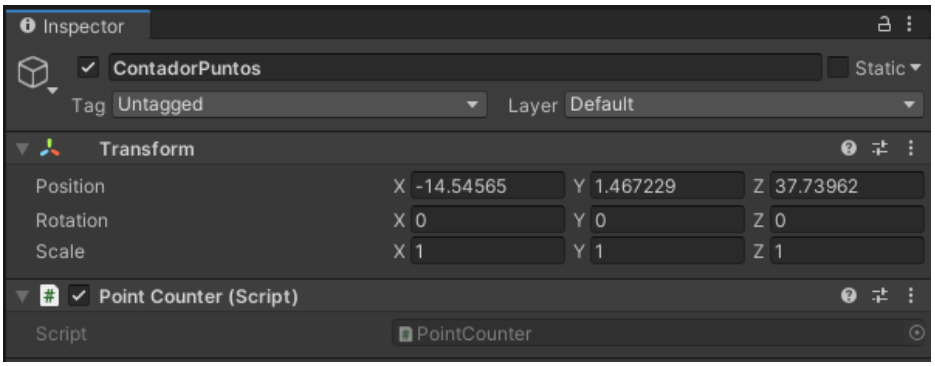


Figura 7.1.2.1. Objeto de contador de puntos.

Dentro de este script, añadiremos una variable encargada de manejar los puntos, a la que añadiremos a medida que los aliens sean destruídos. Adicionalmente, añadiremos una función que podamos llamar desde otros objetos para actualizar los puntos.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
    0 references
5   public class PointCounter : MonoBehaviour
6   {
        1 reference
7       private int puntos = 0;
8
        0 references
9       public void ActualizarPunto(int cantidad){
10          puntos += cantidad;
11      }
12  }
13
```

Figura 7.1.2.2. Clase de contador de puntos.

Ahora podemos añadir un llamado a esta clase desde la función de la bota que se encarga de destruir los aliens. Cuando un alien sea destruído, el contador de puntos sumará un punto.

```
//Componente que actualiza los puntos
0 references
public PointCounter contador;
```

```
//Esta función se llamará cada que la bota interactúe con otro objeto
0 references
private void OnTriggerEnter(Collider other){
    if (other.gameObject.CompareTag("F2D5A7_Alien"))
    {
        //Activación de componente de audio
        gameObject.GetComponent<AudioSource>().Play();
        //Activación de componente de texto
        texto.text = "¡Alien destruído!";
        contador.ActualizarPunto(1);
        //Destruir alien
        Destroy(other.gameObject);
    }
}
```

Figura 7.1.2.3. Cambios a la clase de Bota.

Al haber añadido la referencia al contador, vincularemos el objeto a la bota desde el inspector.

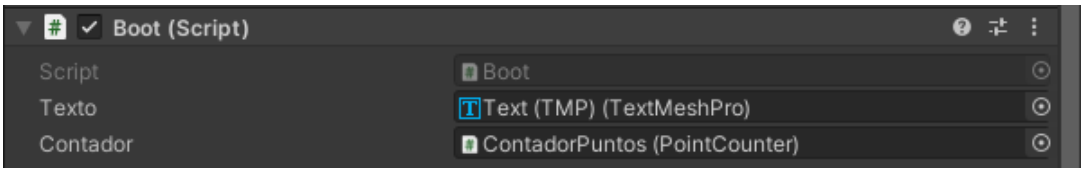


Figura 7.1.2.4. Adición del contador de puntos al objeto bota.

A partir de ahora, el contador de puntos sumará puntos cada que el jugador aplaste un alien. Sin embargo, aún falta un par de cosas para tener un sistema de puntuación funcional. Lo siguiente es añadir una condición de victoria que nos permita verificar cuando el jugador tenga suficientes puntos, y mostrarlo explícitamente en el juego. En la función de actualizar puntos en el contador, añadiremos una condición que cambie el texto en la escena para indicarle al jugador que ya ganó. Adicionalmente, detendrá a las demás naves espaciales de seguir generando aliens. Podemos hacer esto haciendo llamado al objeto texto y al conjunto de naves espaciales, agregándolos a la clase del contador. Además, añadiremos un componente de audio que reproduce un sonido de victoria cuando el jugador gane el juego.

```
//Naves a desactivar al ganar el juego
2 references
public Spaceship[] naves;

//Indicativo para
1 reference
public TextMeshPro texto;
```

Figura 7.1.2.5. Llamado a los objetos a modificar cuando se gane el juego.

Figura 7.1.2.6. Componente de audio en el contador de puntos.

```
1 reference
public void ActualizarPunto(int cantidad){
    puntos += cantidad;

    if(puntos >= 10){
        texto.text = "¡Ganaste!";
        //Activación de componente de audio
        gameObject.GetComponent<AudioSource>().Play();
        for(int i = 0; i< naves.Length; i++){
            naves[i].continua = false;
        }
    }
}
```

Figura 7.1.2.7. Condición de victoria cuando el jugador alcance 10 puntos o más.

Como antes, debemos vincular el texto y las naves espaciales al objeto de contador de puntos.

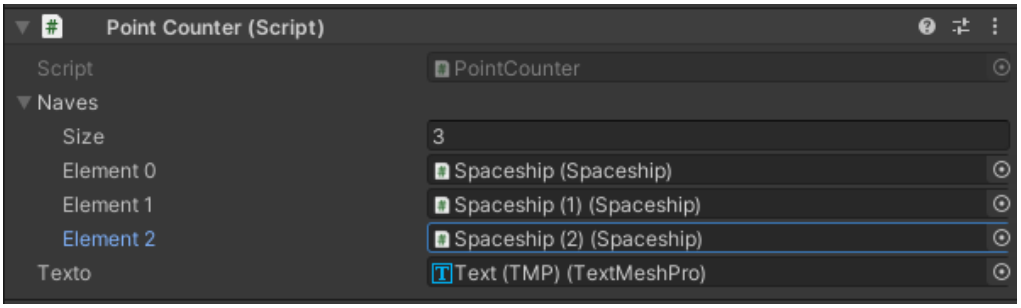


Figura 7.1.2.8. Asignación de objetos en el objeto de contador de puntos.

Ahora el jugador puede saber en qué momento gana el juego, con las indicaciones visuales y auditivas. Sin embargo, ahora necesitamos una condición que cuente los errores del jugador y active una pérdida cuando llegue a cierto número de errores. Podemos empezar haciendo modificaciones en el contador de puntos para crear una función que cuente las pérdidas, de esta manera una sola clase se encargará de manejar la condición de pérdida y de ganancia.

```
2 references
private int puntos = 0;
2 references
private int perdidas = 0;
```

Figura 7.1.2.9. Variables de puntos y pérdidas.

De igual manera, agregaremos una función que actualice las pérdidas, y active la condición de pérdida cuando llegue a cierto número de errores.

```
0 references
public void ActualizarPerdidas(int cantidad){
    perdidas += cantidad;
    if(perdidas >= 5){
        texto.text = "¡Perdiste!";
        //Activación de componente de audio
        gameObject.GetComponent<AudioSource>().Play();
        for(int i = 0; i< naves.Length; i++){
            naves[i].continua = false;
        }
    }
}
```

Figura 7.1.2.10. Función para actualizar las pérdidas.

Como ahora la misma clase maneja dos eventos pero sólo tiene un componente de audio, vamos a agregar dos variables de clips de audio. De esta manera, podremos reproducir sonidos diferentes dependiendo de si el jugador gana o pierde el juego. Añadiremos estas variables a la clase.

```
0 references
public AudioClip ganancia;
0 references
public AudioClip perdida;
```

Figura 7.1.2.11. Variables de clip de audio.

Ahora podemos asignar un clip de audio antes de reproducirlo.

```
//Activación de componente de audio
gameObject.GetComponent<AudioSource>().clip = ganancia;
gameObject.GetComponent<AudioSource>().Play();
```

Figura 7.1.2.12. Asignación de audio de ganancia.

Podemos hacer lo mismo con ambas condiciones, para tener un audio distinto en ambos casos.

```
0 references
public void ActualizarPerdidas(int cantidad){
    perdidas += cantidad;
    if(perdidas >= 5){
        texto.text = "¡Perdiste!";
        //Activación de componente de audio
        gameObject.GetComponent<AudioSource>().clip = perdida;
        gameObject.GetComponent<AudioSource>().Play();
        for(int i = 0; i< naves.Length; i++){
            naves[i].continua = false;
        }
    }
}
```

Figura 7.1.2.13. Cambio a función de actualizar pérdidas.

Haremos las asignaciones respectivas de archivos de audio al objeto de contador de puntos.

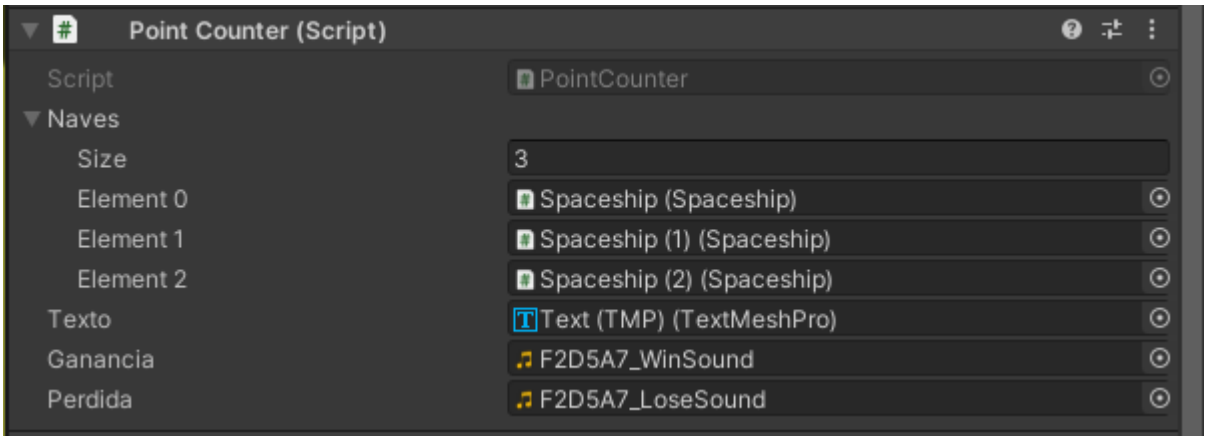


Figura 7.1.2.14. Asignación de sonidos al contador de puntos.

Ahora necesitamos crear la condición que permita aumentar las pérdidas. Como lo dice la ficha técnica, el juego contará una pérdida cuando el alien llegue a la fruta objetivo. Para lograr la colisión entre las frutas y los aliens, debemos modificar componentes en la fruta. Haremos que el colisionador sea un trigger para que pueda reconocer cuando un alien llega hasta ella, y deshabilitaremos las físicas en el componente rigidbody para que se quede en su lugar cuando colisione con un alien (Desactivar gravedad y activar propiedad cinemática). Adicionalmente, agregaremos un script a cada Tomate que se encargará de reconocer las colisiones con los aliens y contar las pérdidas.

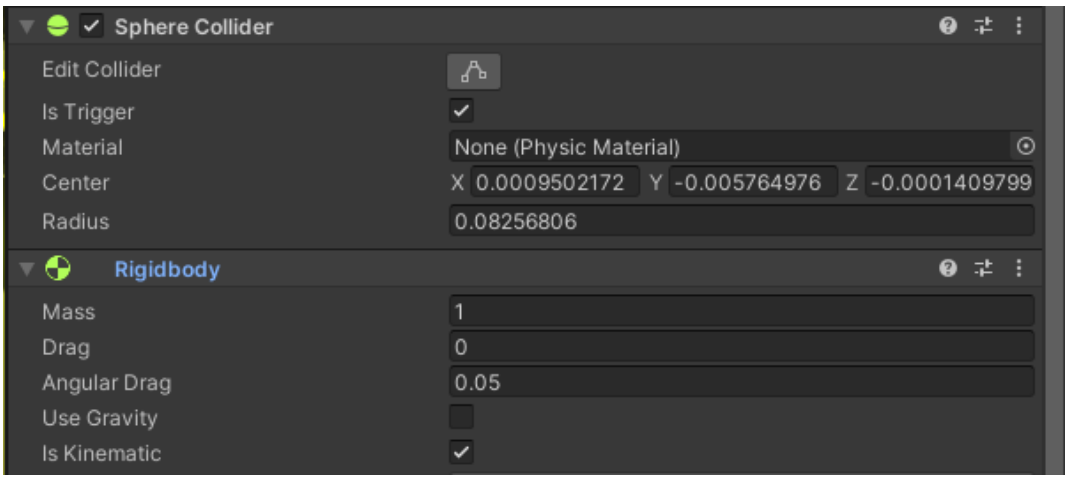


Figura 7.1.2.15. Componentes de física para el tomate

El script de tomate funciona similar al script de la bota. Reconoce una colisión con aliens, y reacciona modificando el texto, llamando al contador de puntos y reproduciendo un sonido. El primer paso es entonces añadir un componente de texto y otro de contador de puntos al script.

```
//Componente que actualiza las pérdidas
1 reference
public PointCounter contador;

//Componente de texto
1 reference
public TextMeshPro texto;
```

Figura 7.1.2.16. Variables en la clase de tomate.

Luego de tener estas variables, tendremos otra función de activación de los triggers, en la que llamaremos la función del contador de puntos.

```
// Update is called once per frame
0 references
void OnTriggerEnter(Collider other){
    if (other.gameObject.CompareTag("F2D5A7_Alien")){
        //Activación de componente de audio
        gameObject.GetComponent<AudioSource>().Play();
        //Activación de componente de texto
        texto.text = "¡Dejaste al Alien pasar!";

        contador.ActualizarPerdidas(1);
    }
}
```

Figura 7.1.2.17. Función para los triggers de los tomates.

De esta manera, asignaremos el objeto de contador de punto y de texto a cada tomate, y agregaremos el componente de sonido para indicar cuando el alien llega hasta el tomate. también deberemos hacer el mismo procedimiento para los demás tomates.

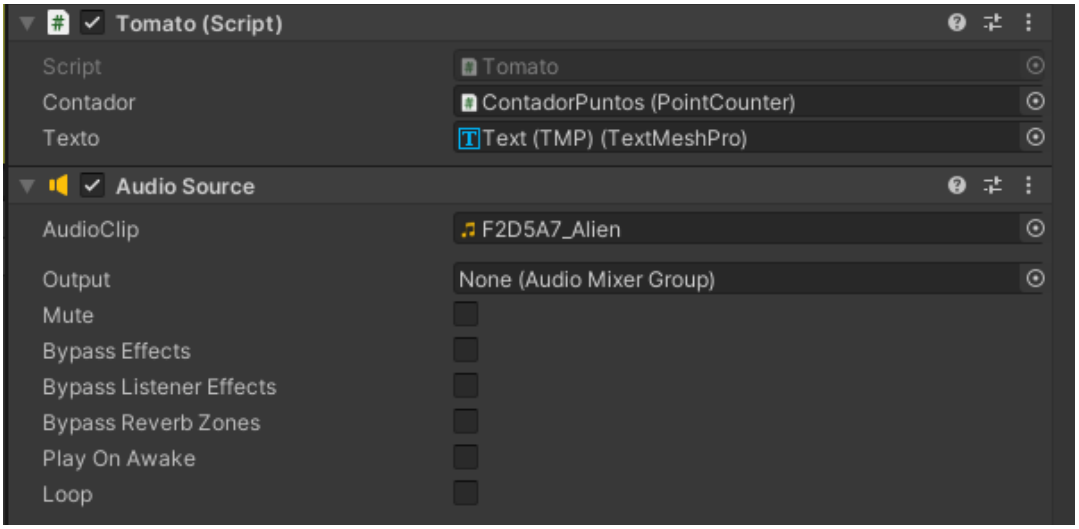


Figura 7.1.2.18. Asignación de componentes en el tomate.

Ahora nuestro juego indicará cuando ganemos o cuando perdamos el juego.

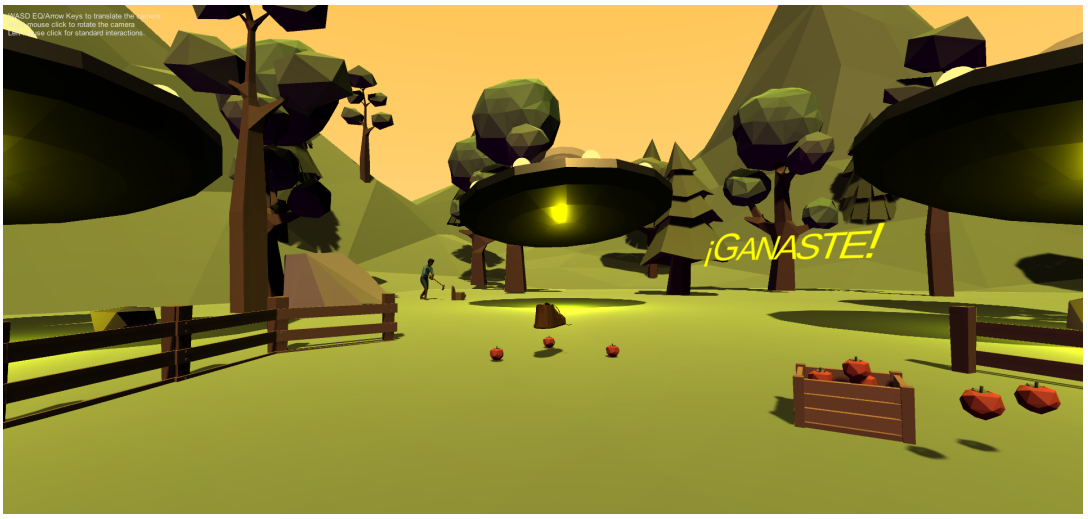


Figura 7.1.2.19. Estado de victoria del videojuego.

Figura 7.1.2.20. Estado de pérdida del juego.

A continuación, podrá encontrar descripciones para otros modos de juego que puede implementar en su videojuego, como alternativas al contador de puntos.

7.3.2. Contador de vidas

Este sistema de puntuación cuenta puntos mientras el jugador cuenta con una cantidad limitada de vidas. Estas vidas representan errores que el jugador puede cometer, agotándose cada vez que el jugador se equivoca. El objetivo es aumentar la dificultad del videojuego con el paso del tiempo para que el jugador se vea motivado a concentrarse más en la actividad. Se puede definir una cantidad de puntos para ganar, o el juego puede seguir indefinidamente, aumentando constantemente la dificultad. De cualquier manera, el juego se pierde cuando el jugador comete una cantidad específica de errores. Al final del juego, se cuentan los puntos que se consiguieron antes de perder las vidas, o si el juego tiene condición de victoria, la cantidad de vidas que se perdieron antes de ganar.

7.3.3. Contratiempo

Este sistema de puntuación pide al jugador realizar la actividad un número de veces mientras un temporizador le quita tiempo. El objetivo es ver cuántas veces el jugador puede realizar la actividad hasta que se le acabe el tiempo, pudiendo fácilmente compararlo con otros intentos o con demás jugadores. En otras variaciones de este sistema de puntos, el jugador puede ganar tiempo al realizar actividades particularmente difíciles o al no cometer errores.

7.3.4. Contador de estrellas o de objetivos

Es una variación del juego a contratiempo, en el que el intento es categorizado en distintas estrellas dependiendo de la destreza del jugador. Por ejemplo, en un juego donde la actividad dure 30 segundos, pueden darse:

- 3 estrellas por obtener 15 puntos
- 2 estrellas por obtener 10 puntos
- 1 estrella por obtener 5 puntos
- No se dan estrellas si se consiguen menos de 5 puntos

Este modo de juego es especialmente útil en juegos que contienen diferentes niveles o diferentes minijuegos, ya que las estrellas se pueden usar como una moneda en la parte externa a los minijuegos.

7.3.5. Zen

Este sistema de puntuación sólo agrega puntos, y no los quita ni penaliza al jugador de ninguna manera. El videojuego no tiene condiciones para ganar o para perder. El objetivo es el de quitar presión del jugador, al no castigarlo por cometer errores y no apresurarlo a ganar. De esta manera el jugador únicamente se enfoca en realizar la actividad hasta perder interés. Este sistema de puntuación no se recomienda en videojuegos serios, debido a que se necesita que el jugador complete un número de objetivos en un tiempo determinado, por lo que es necesario premiar o castigar el jugador para evitar que pase mucho tiempo en una sola actividad.

8. Realización de pruebas y depuración de los videojuegos en realidad virtual.

En esta sección encontrará pautas para poder probar sus videojuegos en el sistema de realidad virtual, así como soluciones a errores comunes y tips para perfeccionar detalles en los minijuegos. No es necesario tener un sistema de realidad virtual para hacer videojuegos serios en realidad virtual. Sin embargo, tener acceso a uno permite reconocer y arreglar errores difíciles de encontrar desde la perspectiva del monitor. En este caso, necesitará lo siguiente:

- Un espacio dedicado a la realidad virtual, sin obstáculos y donde pueda moverse libremente.
- Steam instalado en su equipo
- Gafas de realidad virtual (HTC Vive o Oculus Quest recomendadas)

Esta parte se recomienda realizar con una persona extra, que esté a cargo del computador mientras la otra utiliza las gafas de realidad virtual. Sin embargo, una sola persona puede realizar todo perfectamente, con la organización apropiada.

8.1. Instalación de Steam VR en el equipo.

Para completar este paso, debe tener Steam instalado en su ordenador. Puede adquirir Steam desde su página oficial. Además, debe iniciar sesión en su cuenta de Steam para poder acceder a la tienda. Una vez tenga la aplicación de Steam en su computador, podrá acceder a opciones de Steam desde la barra de herramientas de su sistema operativo.

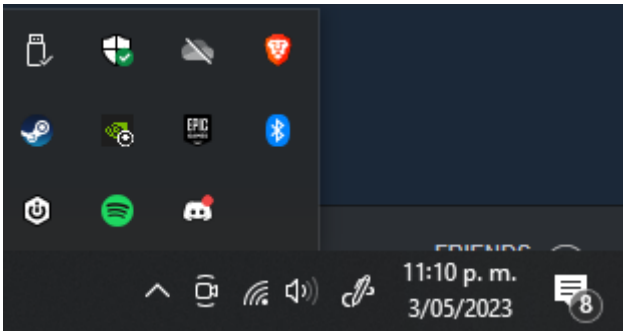


Figura 8.1.1. Steam en la barra de herramientas de Windows 10.

Podemos darle clic al ícono de Steam para acceder a sus opciones. Allí seleccionaremos “SteamVR”, la cual iniciará el proceso de instalación en el computador.

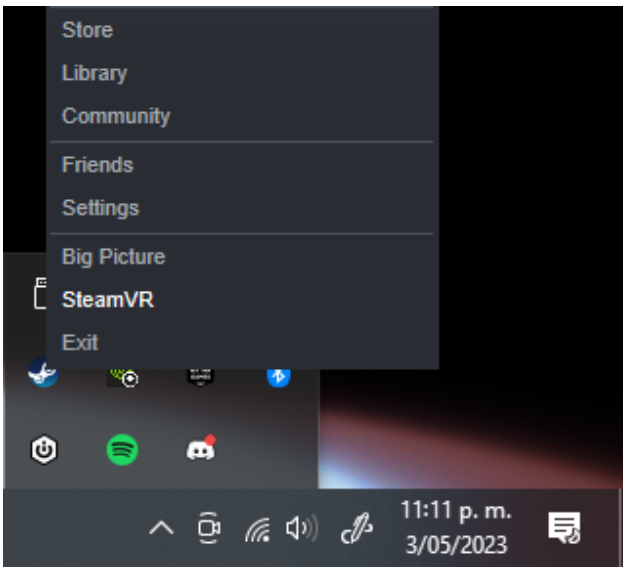


Figura 8.1.2. Opciones de Steam en la barra de herramientas.

Lo único que se debe verificar es la capacidad del disco duro antes de instalar SteamVR. Si se cuenta con el suficiente espacio, sólo es necesario presionar continuar. Adicionalmente podemos crear un acceso directo en el menú de inicio o en el escritorio.

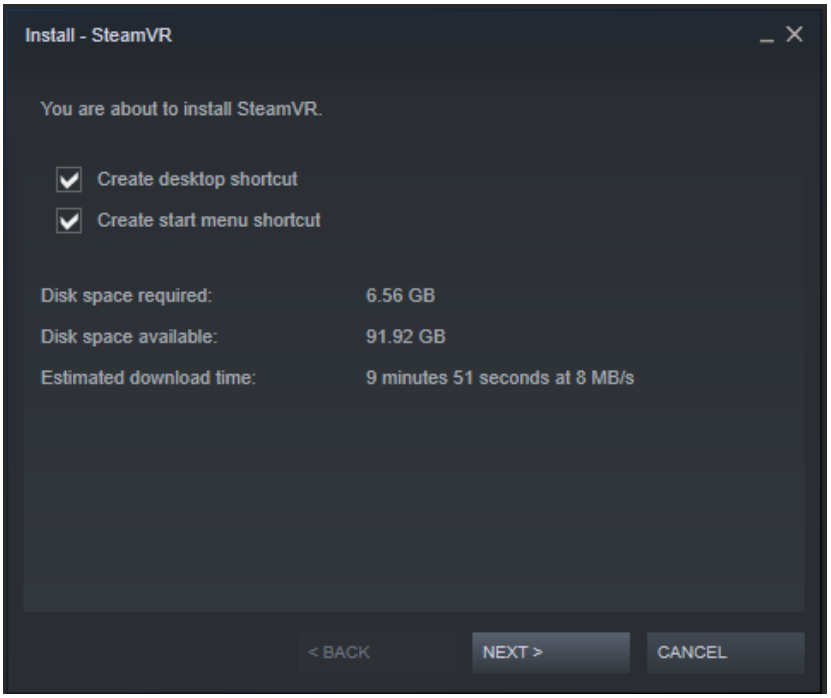


Figura 8.1.3. Ventana de instalación de SteamVR.

A partir de ahora, SteamVR empezará a descargarse en segundo plano. Podemos acceder a la ventana de descargas en Steam para verificar cuando la descarga finalice. Deberíamos ver a SteamVR bajo la pestaña de elementos completados al finalizar:

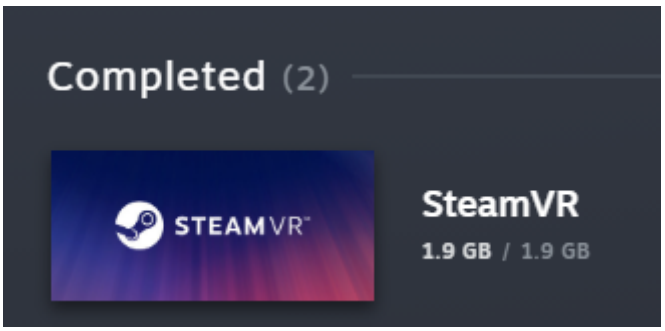


Figura 8.1.4. Pestaña de objetos descargados en Steam.

8.2. Montaje del sistema de realidad virtual.

Una vez instalado SteamVR, se puede realizar el montaje del sistema para probar los videojuegos, empezando por la preparación para el espacio dedicado.

8.2.1. Delimitación del área de juego física

Como se mencionó anteriormente, es buena idea definir un espacio dedicado a la realidad virtual. Este espacio estará libre de obstáculos y servirá para moverse libremente sin el riesgo de chocar o tropezar con objetos en el mundo real. A continuación podrá observar un ejemplo de un área delimitada para realidad virtual.

Figura 8.2.1.1. Delimitación del área de juego utilizando cinta de enmascarar.

8.2.2. Lugar dedicado para el casco

Cuando no se esté usando el sistema de realidad virtual, es buena idea preparar una silla o mesa fuera del área de juego, en la que se pueda poner el casco y los controles mientras no se estén usando. De esta manera, se evita tirarlos al piso accidentalmente mientras se esté trabajando en algo más.

Figura 8.2.2.1. Sistema de realidad virtual en su área de descanso.

8.2.3. Instalación del sistema en el computador

Para instalar el sistema de realidad virtual, puede leer el manual de su sistema, o mirar la guía para algunos sistemas de realidad virtual populares en la sección de recursos. Una vez instalado, sólo es necesario conectarlo. En el caso de HTC VIVE, esto también significa la conexión correcta del dispositivo de RV a su *Link Box*.

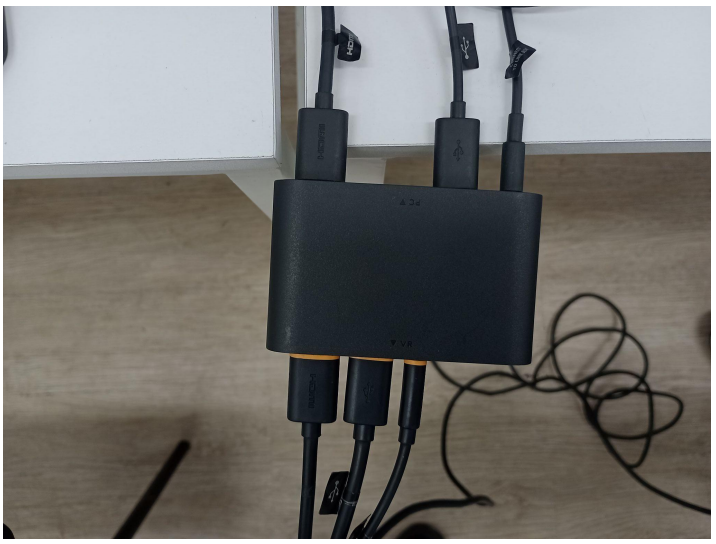


Figura 8.2.3.1. Conexión de las gafas de realidad virtual a la *Link Box.*

Figura 8.2.3.2. Sistema de realidad virtual conectado con el equipo

8.2.4. Conexión con SteamVR

Al tener instalado el sistema de realidad virtual, podemos realizar la conexión de los cables indicados con el computador. Ahora simplemente accederemos a la aplicación de SteamVR y se podrá observar el cambio en los íconos, la cual representa los elementos activos del sistema de RV.

Figura 8.2.4.1. SteamVR luego de conectar el sistema de RV al computador

Una vez encienda los controles y coloque el casco en su cabeza, podrá ver cómo estos elementos se encienden en la aplicación.

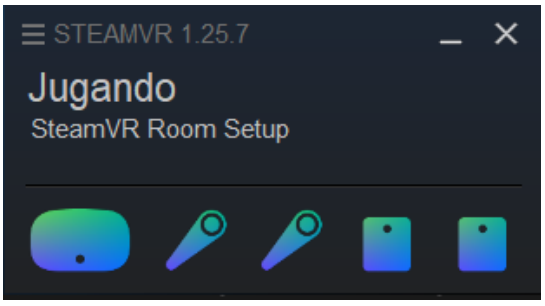


Figura 8.2.4.2. SteamVR luego de ponerse el caso y encender los controles

8.2.5. Delimitación del área de juego virtual

Al ponerse el casco por primera vez, SteamVR empezará a pedir que delimite su área virtual.

Figura 8.2.5.1. Pantalla inicial de configuración de SteamVR.

Para esta guía, se eligió la opción “A escala de Habitación”, ya que se cuenta con el espacio suficiente para moverse libremente. Elija la opción que mejor se ajuste a su área de juego. Una vez lo haga, sólo será necesario seguir los pasos que le indique el programa para establecer el área de juego virtual. El primer paso requiere preparar el área de juego física, lo cual ya se realizó pasos anteriores en esta guía.

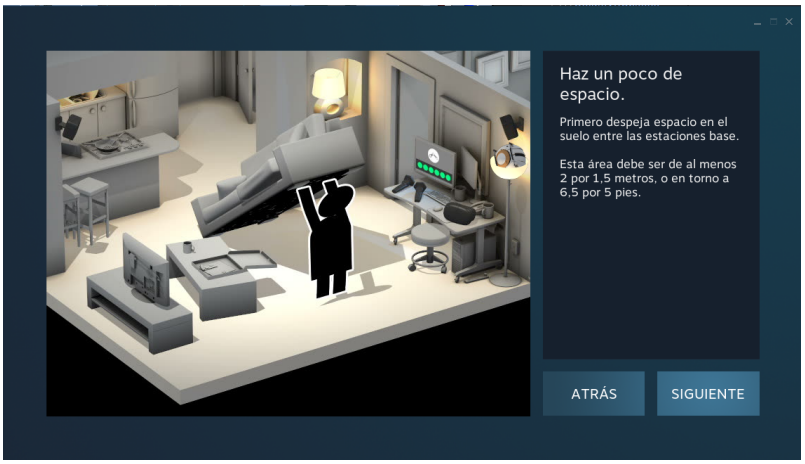


Figura 8.2.5.2. Primer paso en delimitación de área virtual.

A continuación, podrá ver una serie de figuras demostrando el proceso de delimitación de área virtual tanto en la pantalla del monitor, como lo que se debe hacer al tener el sistema de realidad virtual puesto.

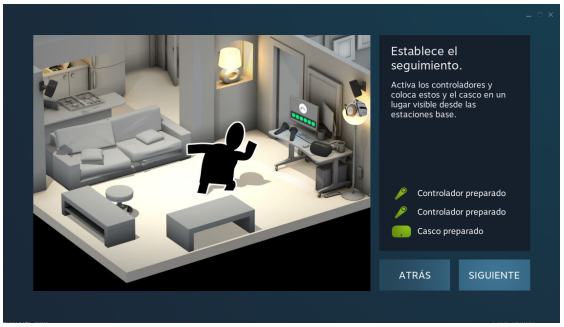


Figura 8.2.5.3. Ubicar controles y casco en un punto visible para las cámaras rastreadoras.

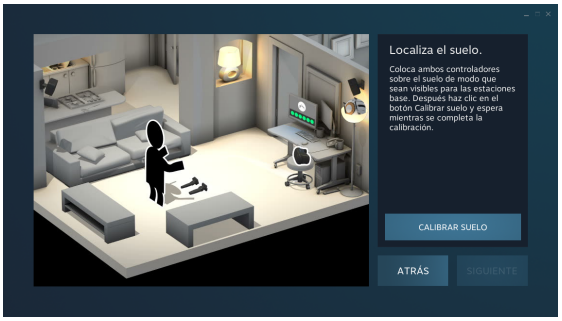


Figura 8.2.5.4. Ubicar los controles en el suelo para alinearlo con el suelo virtual.

Figura 8.2.5.5. Trazar el espacio de juego virtual con el uso de los controles.

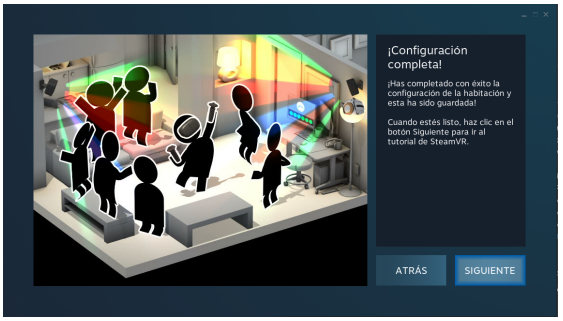


Figura 8.2.5.6. Ventana de SteamVR luego de que el área de juego haya sido definida.

Luego de haber configurado la sala, SteamVR lo enviará al escenario virtual de VR Home. Siéntase libre de explorar este ambiente para familiarizarse con la realidad virtual antes de realizar pruebas.

Figura 8.2.5.7. Steam Home desde la perspectiva del jugador.

8.3. Ejecución del videojuego en realidad virtual

Una vez haya realizado el montaje del sistema de realidad virtual, podrá ingresar a Unity, como si fuese a trabajar en el proyecto de manera normal. Con SteamVR encendido, sólo debe presionar jugar en el editor como lo haría normalmente.

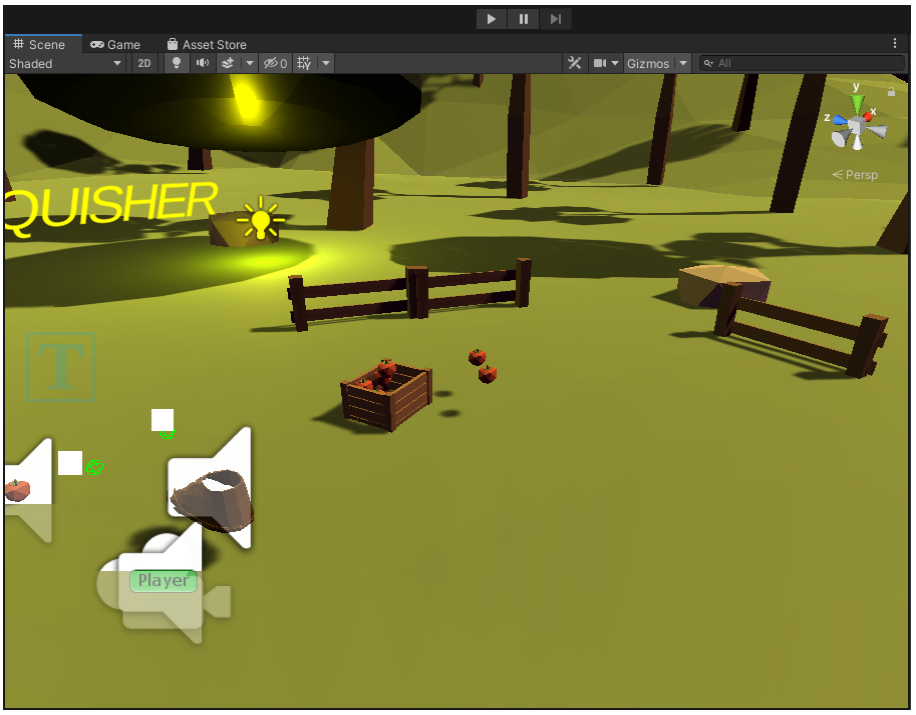


Figura 8.3.1. Editor de Unity, aún con SteamVR encendido

Con los controles encendidos y una vez iniciado el juego, podrá ver cómo los controles se convierten en manos.

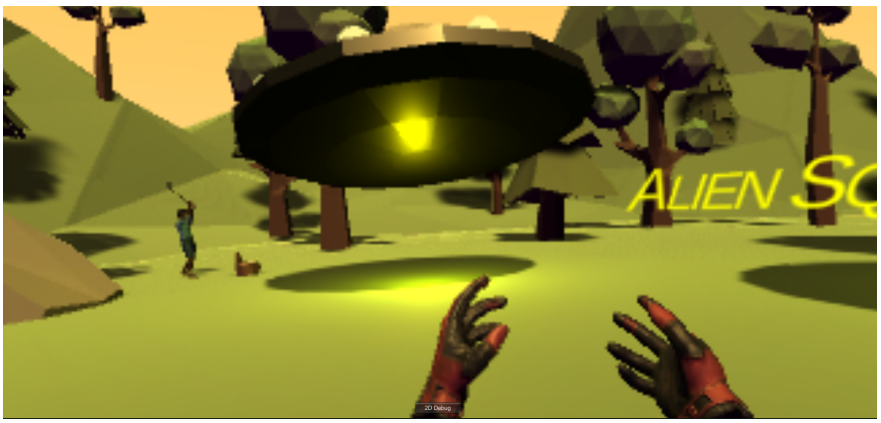


Figura 8.3.2. Perspectiva del juego desde realidad virtual.

Estas manos son muy importantes, pues serán las que utilizaremos para interactuar con todos los elementos interactuables de nuestro juego. Por ejemplo, podemos utilizar el botón dedicado en el control para agarrar los objetos. Este botón puede variar dependiendo del sistema de realidad virtual, por lo que es importante experimentar las funciones de los distintos botones y leer la guía del sistema que se esté utilizando. En Unity, los objetos tendrán un brillo amarillo a su alrededor para indicarnos que podemos interactuar con ellos.

Figura 8.3.3. Bota indicando que puede ser levantada con el control.

Con estos conceptos claros, podemos comenzar a probar el videojuego con el sistema de realidad virtual.

8.4. Depuración

A continuación podrá ver un par de consejos para agilizar el proceso de pruebas y solución de errores en realidad virtual.

8.4.1. Exploración del menú virtual

Dependiendo del sistema de realidad virtual que esté utilizando, deberá presionar un botón en específico para acceder al menú virtual. Este menú abrirá una pestaña con su actividad actual, junto con una barra de herramientas debajo, como se muestra en la siguiente figura.

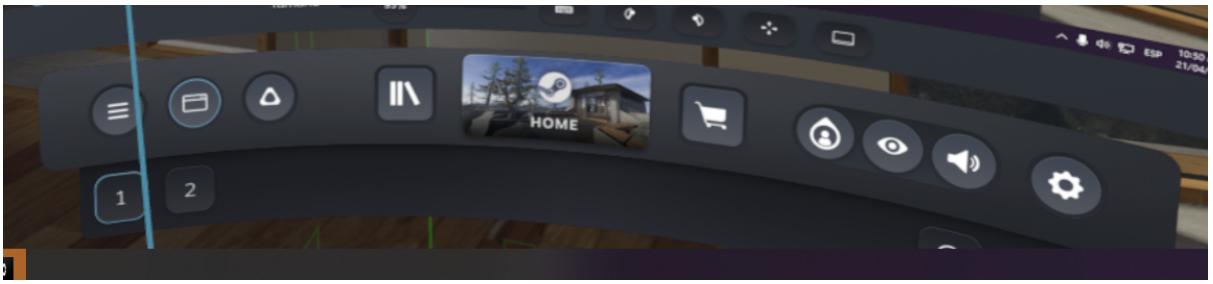


Figura 8.4.1.1. Barra de herramientas de menú virtual

Con este menú abierto, podrá utilizar el puntero de sus controles para presionar los distintos botones virtuales. En este particular caso, presionaremos el segundo botón de la barra de herramientas para acceder al monitor virtual. Esto nos permitirá ver lo que esté en la pantalla del computador mientras tengamos las gafas de RV puestas.

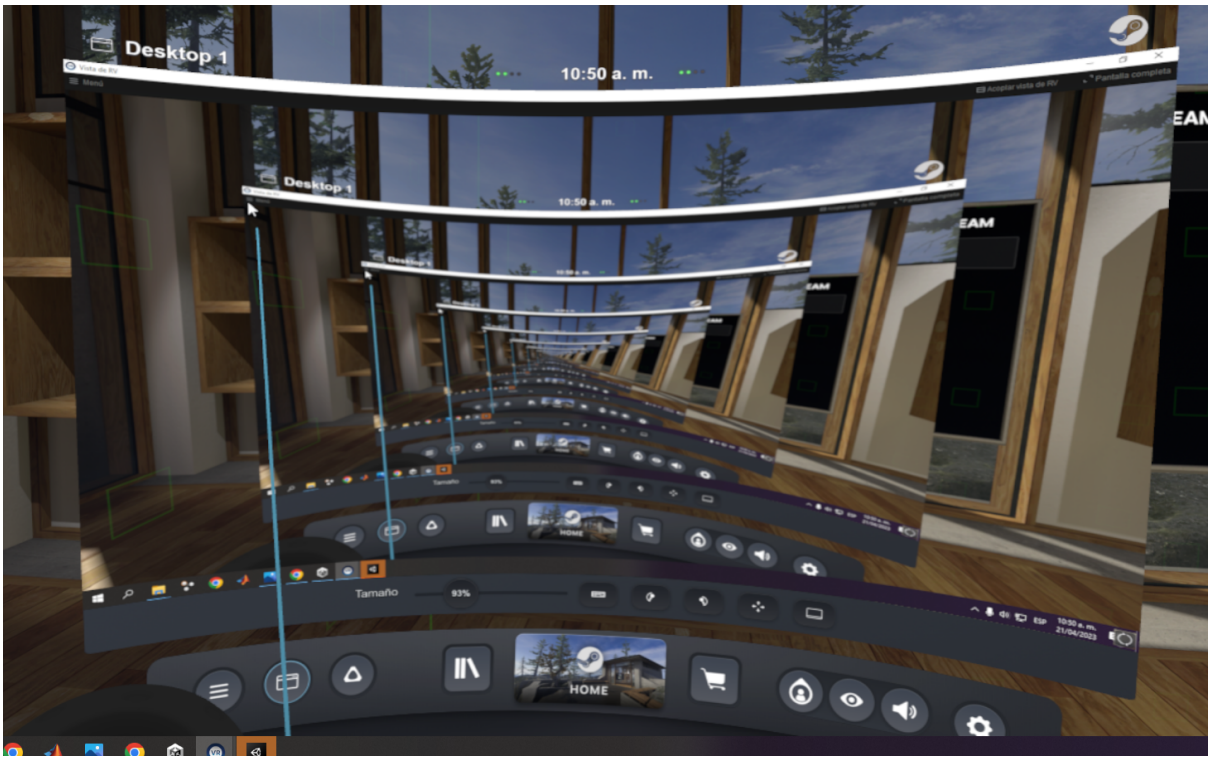


Figura 8.4.1.2. Vista del monitor desde el sistema de realidad virtual.

De similar manera, tener abierto el monitor dentro del sistema de RV nos abrirá una segunda barra de herramientas, la cual contiene opciones como tamaño del monitor virtual, teclado en pantalla, o poder anclar el monitor virtual a los controles, o dejarlo flotando en el ambiente virtual. Es recomendable experimentar con estas opciones para comprender de mejor manera su funcionamiento.

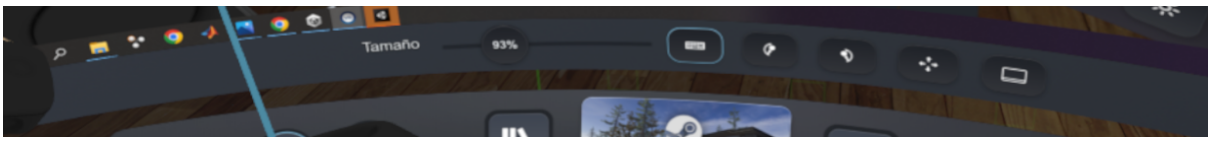


Figura 8.4.1.3. Opciones de monitor virtual.

8.4.2. Editar el juego desde la realidad virtual

Con el monitor virtual abierto, puede abrir Unity en el equipo para editarlo desde el sistema de realidad virtual. Aunque esto no ofrece mucha libertad o precisión, es especialmente útil para arreglar errores mínimos en la escena sin tener que remover el casco de realidad virtual. Con esta técnica también podrá iniciar, detener y reiniciar el videojuego en modo de pruebas sin tener que quitarse el casco.

Figura 8.4.2.1. Editor de Unity dentro del ambiente de realidad virtual.

8.4.3. Tamaño del jugador respecto al ambiente virtual

Finalmente, uno de los errores más comunes consiste en el del jugador viéndose muy grande o muy pequeño en respecto a los objetos que le rodean. La solución es simple, pero involucra estar cambiando valores poco a poco hasta que nos encontremos con un punto que sintamos cómodo. Lo primero es recordar siempre que la posición en Y del jugador debe permanecer en 0, y su escala en todos los ejes debe ser 1. Esto es para lograr la homogeneidad entre los diferentes minijuegos, y para no generar disparidad en el sistema de realidad virtual.

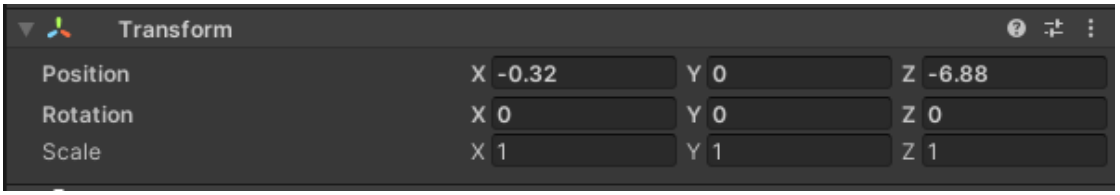


Figura 8.4.3.1. Componente de transformación del jugador.

Para arreglar el tamaño del jugador, será necesario modificar el resto de la escena. Para ello, podemos seleccionar todos los elementos en la escena salvo el jugador. Haremos esto presionando la tecla *Ctrl* mientras seleccionamos los elementos. Esto nos permitirá seleccionar múltiples elementos simultáneamente.

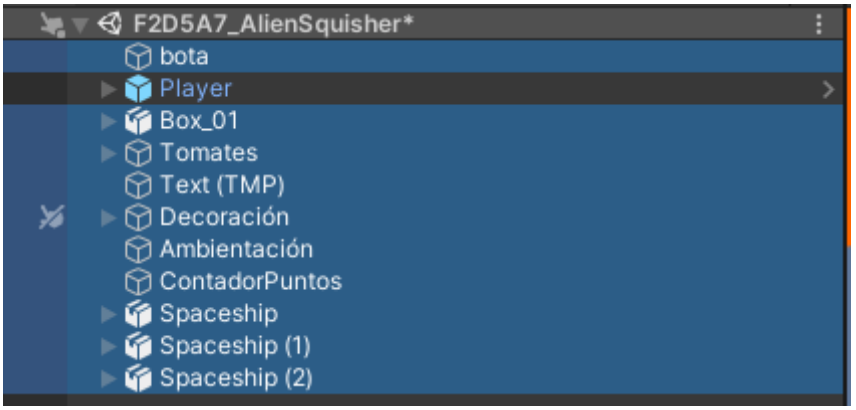


Figura 8.4.3.2. Selección de todos los elementos de la escena salvo el jugador.

Con estos elementos seleccionados, podemos acceder a la barra de herramientas y cambiar la manera en la que los modificaremos. Cambiaremos sólo la escala de los elementos, y cambiaremos el modo de transformación a “Centro”. De esta manera, los elementos cambiarán de tamaño respecto a la escena, en lugar de independientemente.

Figura 8.4.3.3. Configuración de barra de herramientas para cambiar todos los objetos simultáneamente.

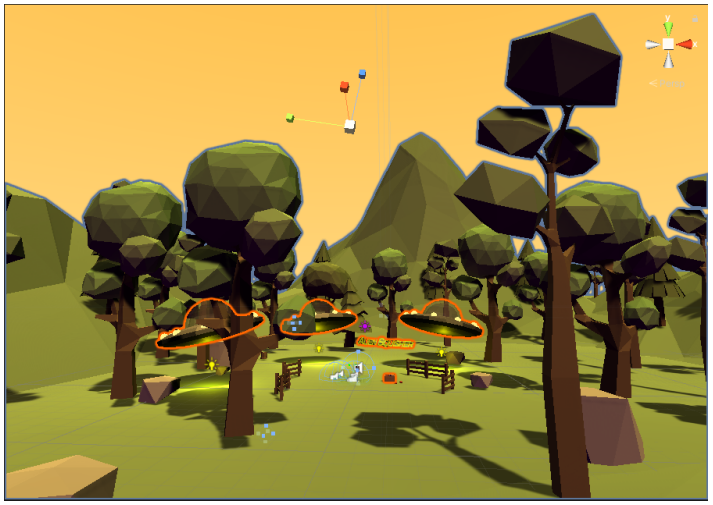


Figura 8.4.3.4. Eje para modificar todos los elementos en la escena.

Ahora podemos agarrar el eje central de los elementos y modificarlos para hacer la escena más grande o más pequeña.

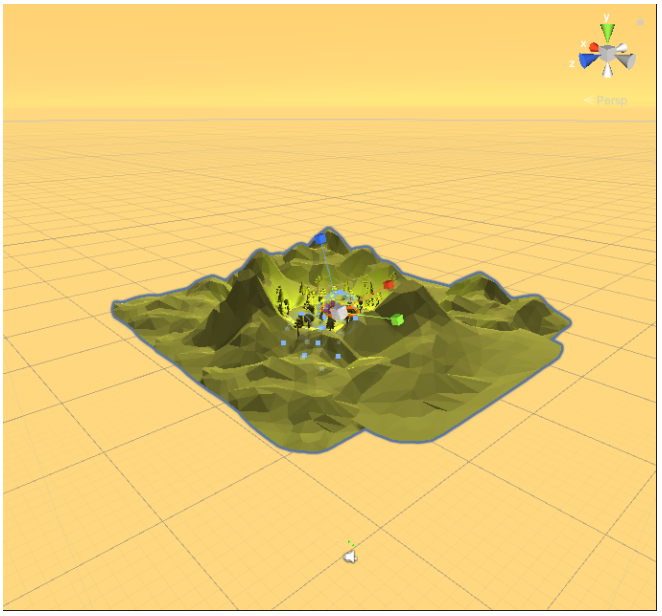


Figura 8.4.3.5. Escena hecha más pequeña

Figura 8.4.3.6. Escena hecha más grande.

Ahora puede hacer que la escena se sienta cómoda con el sistema de realidad virtual, haciendo que los elementos se vean a escala a los del mundo real. Si ahora los elementos con movimiento se mueven con mayor o menor velocidad, puede cambiar el valor de velocidad en el script de cada uno de los objetos para que se ajusten al nuevo tamaño de la escena.

9. Recursos

A continuación podrá encontrar una lista de páginas web en donde podrá encontrar recursos para la creación de videojuegos.

9.1. Documentación de Unity y educación

9.1.1. Documentación

- https://docs.unity3d.com/Manual/index.html
- https://docs.unity.com/
- https://docs.unity3d.com/ScriptReference/index.html

9.1.2. Guías y recursos de educación

- https://unity.com/learn/get-started
- https://www.tutorialspoint.com/unity/unity_tutorial.pdf
- https://www.makeuseof.com/tag/programming-game-unity-beginners-guide/
- https://www.juegostudio.com/blog/what-is-unity-3d-a-comprehensive-guide-to-unitys-features-and-uses

9.2. Texturas y recursos 2D

- https://polyhaven.com/
- https://itch.io/game-assets/free
- https://opengameart.org/latest
- https://kenney.nl/assets
- https://ambientcg.com/
- https://www.textures.com/
- https://nasa3d.arc.nasa.gov/images

9.3. Modelos y otros recursos 3D

- https://polyhaven.com/
- https://poly.pizza/
- https://sketchfab.com/feed
- https://www.turbosquid.com/
- https://www.cgtrader.com/
- https://itch.io/game-assets/free/tag-3d
- https://nasa3d.arc.nasa.gov/models
- https://www.poliigon.com/

- https://opengameart.org/latest
- https://kenney.nl/assets
- https://free3d.com/

9.4. Música y efectos de sonido

- https://freesound.org/
- http://openmusicarchive.org/
- https://freemusicarchive.org/
- https://incompetech.com/
- https://www.indiegamemusic.com/searchtracks.php
- https://kenney.nl/assets?q=audio

9.5. Guías para instalación de sistemas de realidad virtual

- Guía para montar el sistema HTC VIVE: https://www.vive.com/au/support/vive/category_howto/setting-up-for-the-first-time.html
- Guía para montar el sistema Oculus Quest: https://www.meta.com/quest/setup/
- Guía para montar el sistema Valve Index: https://help.steampowered.com/en/faqs/view/7F7D-77FB-8CAA-4329
- Guía para montar el sistema HP Reverb: https://h20195.www2.hp.com/v2/GetDocument.aspx?docname=4AA7-5420ENW

9.6. Demostración del videojuego

Aquí podrá encontrar el repositorio con el desarrollo de Alien Squisher. El propósito de este repositorio es el de ayudarle a encontrar posibles errores o a probar mecánicas de Unity que no queden claras con el seguimiento de esta guía.