

Programación de videojuegos

Breve descripción:

Durante este componente, el aprendiz tendrá la oportunidad de desarrollar un prototipo de videojuego. El proceso comienza con la comprensión de la metodología de trabajo en un entorno real, utilizando métodos ágiles como Scrum. Posteriormente, el aprendiz adquirirá una comprensión de los conceptos fundamentales de la programación, lo que le permitirá implementar el prototipo del videojuego utilizando Unity.

Diciembre 2023

Tabla de contenido

Introducción	4
1. Metodologías de desarrollo	6
1.1. Metodologías ágiles.....	7
1.2. Metodología SCRUM	10
1.3. Metodología SUM	14
2. Programación en Unity 3D	19
2.1. Conceptos básicos de algoritmo y programa.....	21
2.2. Diagramas de flujo.....	26
2.3. Conceptos de variables, operadores, funciones, métodos, estructuras de control	30
2.4. Funciones de eventos y control de acciones	40
2.5. Estructura básica de un script de Unity 3D.....	47
2.6. Control mediante interfaz de usuario (Inspector)	53
2.7. Arreglos	57
2.8. Listas.....	60
3. Programación Orientada a Objetos.....	63
3.1. Clases.....	63
3.2. Atributos y cualificadores	66
3.3. Métodos	66
4. Matemáticas para Videojuegos.....	70
4.1. Vectores.....	70
4.2. Matrices.....	72

4.3. Operaciones sobre matrices	73
5. Interfaz y flujo de integración de Unity 3D	77
5.1. Instalación de Unity 3D.....	77
5.2. Navegación en Unity 3D	77
5.3. Escenario y ventanas de Unity 3D	82
6. Unity 3D importar elementos 3D - assets	84
7. Componer los escenarios (personajes, props, fondos)	102
8. Iluminación	106
8.1. La iluminación en los videojuegos	106
8.2. Luces en Unity.....	110
9. Cámara	112
10. Efectos visuales	113
11. Interfaz	114
11.1. Interfaz del videojuego	114
11.2. Diseño de interfaz del videojuego	114
Síntesis.....	122
Material complementario.....	123
Glosario	124
Referencias bibliográficas	126
Créditos	127

Introducción

Estimado aprendiz, bienvenido al componente formativo “Programación de videojuegos”. Para comenzar, le invitamos a ingresar al siguiente video para obtener más información:

Video 1. Programación orientada a objetos



[Enlace de reproducción del video](#)

Síntesis del video: Programación orientada a objetos

En el mundo de videojuegos existen herramientas que facilitan el diseño de juegos con las mecánicas de uso deseadas. Pero hay momentos en que se necesita aplicar funciones que no están en el motor gráfico utilizado y que son necesarias para

completar la lógica del juego. Esto conlleva a que el creador del juego se enfrente a la escritura de códigos para completar las acciones que lo hacen atractivo para los jugadores. Se hace entonces indispensable aprender sobre la forma de programar en un videojuego, por lo que es necesario comprender la estructura básica de un programa y cómo se debe iniciar su escritura en lenguaje de programación que pueda ser procesado por el motor que se esté utilizando como herramienta de creación del videojuego.

En este componente formativo se podrán adquirir los fundamentos de desarrollo y la estructura básica de un programa en lenguaje de programación C#, utilizado en el motor gráfico y Unity, que se viene estudiando a lo largo del proceso para la creación de comportamientos específicos de los personajes, diseñados adicionar mecánicas inexistentes de manera automática, construcción de nuevos componentes, entre otras. No solo es importante adquirir el conocimiento técnico sobre la creación de nuevos elementos en los videojuegos, sino también el conocimiento de nuevas metodologías de trabajo que permitan desarrollos ágiles y el logro de resultados en el menor tiempo posible, cuando se trata de entregar productos a los usuarios de un sector productivo que los consume. ¡Manos a la obra!

1. Metodologías de desarrollo

Son un conjunto de métodos y técnicas con el objetivo de organizar los equipos de trabajo para diseñar soluciones y desarrollar las funciones de un programa de una manera organizada y eficiente.

- **Metodología rígida.** Anteriormente para el desarrollo de un proyecto de “software” se utilizaba una metodología muy rígida en la que se llevaban a cabo procesos de identificación de necesidades, análisis y diseño, planificación, codificación, prueba piloto e implementación; fases que todavía se aplican, pero que exigían grandes esfuerzos y se invertía muchísimo tiempo sobre todo en la etapa de planificación donde se debía documentar hasta el más mínimo detalle.
- **Reprocesos.** Toda esta rigurosidad se sustentaba en el argumento de que no se tuviera que hacer ningún reproceso y el producto liberado fuera de alta calidad. Entonces, con la cantidad de tiempo invertido se hacía casi imposible realizar modificaciones en los requisitos, pues tocaba empezar de cero con el levantamiento de la información, la planificación y la documentación para diseñar y desarrollar el producto nuevamente.
- **Metodologías ágiles.** Cuando se emprendían proyectos de desarrollo de “software” mucho más pequeños, que exigían tiempos de respuesta cortos para la obtención de resultados, esta metodología se tornó completamente ineficiente, pues se gastaba mucho tiempo y recursos en cada fase del proyecto, y no se podían realizar cambios a los requisitos para mejorar el producto. Por lo tanto, empezaron a surgir nuevos métodos más ágiles, con un enfoque iterativo, es decir, por bloques de

tareas, para llevar a cabo los requisitos de los usuarios, teniendo en cuenta que estos pueden cambiar durante todo el proceso de desarrollo.

1.1. Metodologías ágiles

Como se mencionó anteriormente, las metodologías ágiles surgieron de la necesidad de proporcionar respuestas rápidas a los requerimientos de los proyectos, manteniendo flexibilidad frente a los cambios que puedan surgir o se puedan generar durante el proceso de desarrollo.

Se utilizan para gestionar proyectos en los que el cliente no tiene claro todos los requisitos, por lo que no se puede definir el alcance desde el principio. Igualmente, cuando durante el desarrollo el cliente cambia de opinión o adiciona más requisitos.

Estas metodologías no solamente se utilizan en el sector de las TIC para el desarrollo de “software” y sistemas de información, también se pueden aplicar a cualquier sector productivo, proyecto que se quiera emprender o entorno donde se produzcan cambios de forma frecuente.

Existen aspectos muy marcados de las metodologías ágiles respecto a las metodologías tradicionales que dejan entrever sus características, presentados en la tabla 1:

Tabla 1. Metodologías ágiles vs Metodologías tradicionales

Metodologías Ágiles	Metodologías Tradicionales
Se enfoca en las personas	Se enfoca en los procesos
El cliente participa en todas las fases del proyecto	El cliente solo participa al inicio del proyecto

Metodologías Ágiles	Metodologías Tradicionales
El equipo trabaja colaborativamente	Cada miembro del equipo desarrolla una tarea
Trabaja con proyectos medianos y pequeños	Trabaja con cualquier proyecto independientemente del tamaño que sea
Lleva a cabo los cambios en los requisitos durante todo el proceso	No lleva a cabo ningún cambio en los requisitos durante el proceso
Planifica los requisitos del proyecto en una lista de iteraciones (tareas) para realizar en corto tiempo, luego el desarrollo es iterativo	Planifica los requisitos del proyecto en un desarrollo que se lleva a cabo de manera lineal durante todo el proceso
Hay retroalimentaciones por cada iteración para hacer mejoras al producto durante el proceso de desarrollo	No hay retroalimentaciones durante el proceso de desarrollo
Realiza entregas del proyecto al terminar cada iteración	Realiza solo una entrega cuando finaliza el proyecto

Agile Manifiesto

Debido a los diferentes procedimientos que se utilizaban para llevar a cabo los proyectos de “software” y con el afán de unificarlos en un conjunto de buenas prácticas que se pudieran aplicar en los procesos de desarrollo, en el 2001 se creó la organización Agile Alliance; cuyo principal objetivo fue definir un conjunto de principios bajo los cuales se pueda trabajar de manera ágil y con calidad en cualquier entorno.

Dichos principios fueron acuñados con el nombre de “Agile Manifiesto” y su fundamento se halla en valores que promueven una cultura basada en la aceptación y adaptación a los cambios que sean necesarios para el progreso de una organización. Los valores mencionados priorizan:

- a) Los individuos y las interacciones que se puedan realizar sobre los procesos y las herramientas.
- b) La funcionalidad del “software” que se desarrolle sobre la excesiva documentación que se haga sobre este.
- c) La colaboración con el cliente sobre los detalles de los aspectos contractuales.
- d) Las respuestas ante los cambios sobre seguir un plan preestablecido.

Los valores anteriores se concretaron en doce (12) principios que fueron consignados en un documento llamado Manifiesto Ágil con el objeto de que sirvan como marco de trabajo para cualquier equipo ágil:

Figura 1. Manifiesto Ágil



Para complementar el tema se recomienda realizar las lecturas Gestión Ágil de proyectos y Manifiesto por el Desarrollo Ágil de “software”, que se encuentran en el material complementario.

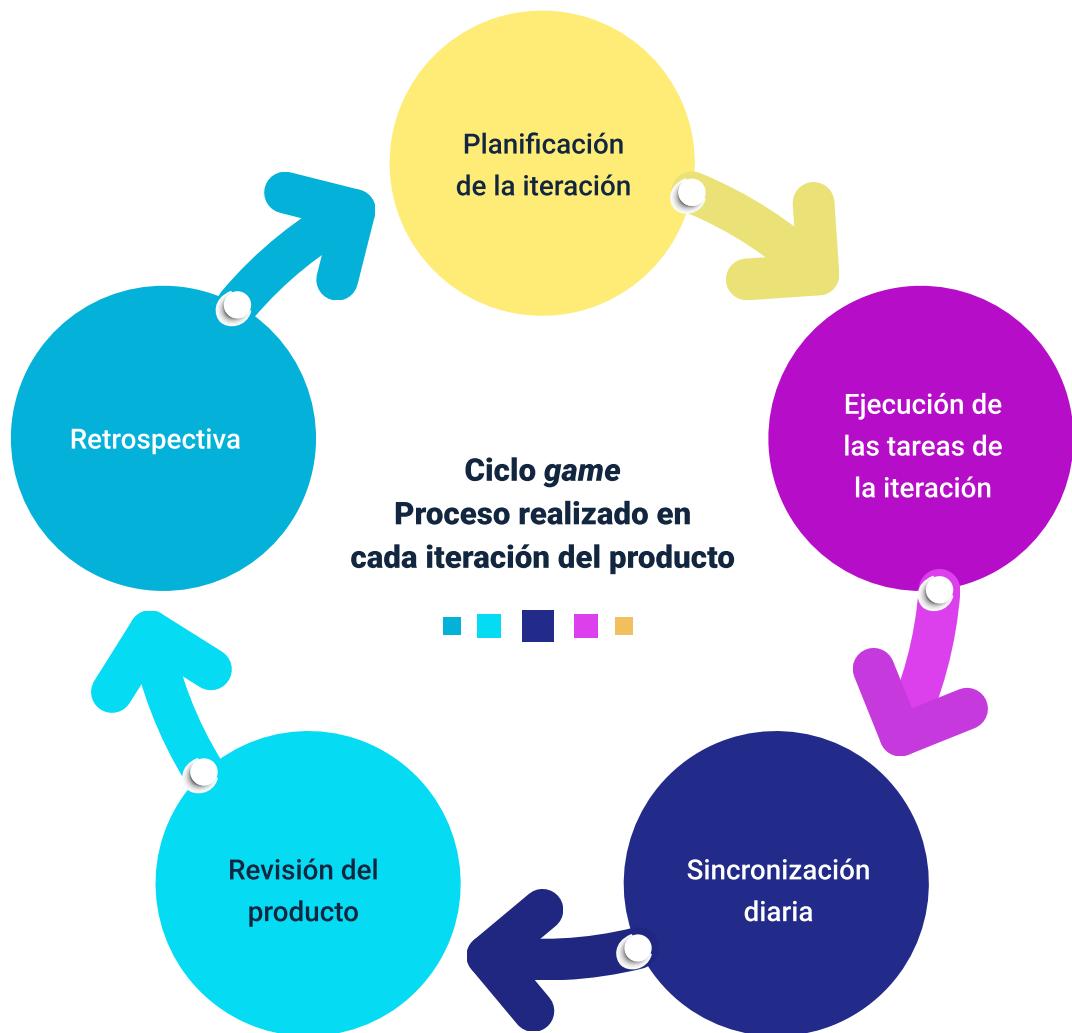
1.2. Metodología SCRUM

SCRUM es una de las metodologías ágiles que determinan un marco de trabajo mediante el cual se pueden enfocar problemáticas y adaptar soluciones para entregar productos de la máxima calidad y valor posibles. Se basa en el control de procesos en el desarrollo de productos y se puede llevar a cabo en tres ciclos; “pre game”, “game” y “pos game” basado en los conocimientos y la experiencia de las personas que participan en ellos.

- **“Pregame”.** Se realiza una lista ordenada de lo que es necesario para el desarrollo del producto y es la fuente de requisitos para la realización de los cambios, ya que se puede variar a medida que avanza el proceso y es ordenada porque se va desarrollando de acuerdo con las prioridades establecidas. Cada elemento de la lista contiene la visión del usuario sobre las funcionalidades que espera encontrar en el producto.
En este ciclo se seleccionan los “Sprint” (requisitos), se determinan los costos y los recursos para la construcción del producto, se hace un análisis general de lo que se debe entregar como producto final y se realiza un diseño del prototipo de este.
- **“Game”.** En este ciclo se desarrolla la lista de requisitos que se priorizan a través de una planificación; se realiza la lista de las tareas por llevar a cabo (iteraciones); se determina el tiempo de las iteraciones que es aproximadamente de una semana (1) por actividad hasta completar un

máximo de cuatro (4) por iteración, es decir, 30 días y el equipo se organizan en parejas o en las personas que sean necesarias para completarla. Los miembros del equipo se auto organizan y establecen sus propios objetivos de acuerdo con el producto que debe entregar al finalizar la iteración, revisar figura 2.

Figura 2. “Ciclo game” - Proceso realizado en cada iteración del producto



- Después de planificar la iteración que algunos autores como Sánchez (2018) llaman “Sprint”, se comienza a ejecutar cada tarea de la iteración:
- a)** Cada día se realiza una sincronización que consiste en una reunión del equipo con duración de 15 minutos frente a un tablero para autoevaluarse frente a los avances en la tarea en la que los miembros deben responder a las preguntas:
 - ¿Qué he hecho desde ayer para aportar al cumplimiento de la tarea?
 - ¿Qué voy a hacer a partir de este momento para ayudar al equipo a cumplir el objetivo de la tarea?
 - ¿Qué obstáculos me impiden colaborar con el trabajo en equipo para conseguir el objetivo?
 - b)** De acuerdo con las respuestas proporcionadas por los integrantes del equipo, el SCRUM “manager” (gestionador del proyecto) debe encargarse de hacer la inspección de los obstáculos y hacer las adaptaciones necesarias para eliminarlos, garantizando que el equipo pueda realizar las actividades para terminar la iteración.
 - c)** Después de cada superación de obstáculos y para terminar la iteración se hace una revisión del producto con lo cual se verifica el cumplimiento de los requisitos. Finalmente el día que se tiene previsto terminar la iteración, se hace nuevamente una reunión de 1 o 2 horas con el cliente para realizar una demostración del cumplimiento de los requisitos plasmados en el producto incremental, resultado de la iteración completada.
 - d)** De acuerdo con las observaciones recibidas por el cliente se inspeccionan de nuevo los obstáculos que impiden la optimización del producto

obtenido y se hacen las adaptaciones necesarias. El SCRUM “manager” junto con el equipo realiza una retrospectiva del trabajo para identificar en que se puede mejorar la metodología y las técnicas utilizadas para incrementar la productividad.

“Post-game”

Corresponde al cierre del proyecto dado que se han llevado a cabo todas las iteraciones que lo han perfeccionado para que el cliente quede a satisfacción con las funciones que esperaba. Entonces se prepara el producto para ser liberado haciendo la verificación de las versiones anteriores que se tengan de este, las cuales deben ser almacenadas en repositorios o carpetas para tener un control de versiones.

En esta metodología se pueden distinguir claramente tres (3) roles principales:

- **“Product Owner”.** Es el responsable de entregar un producto de calidad al cliente como resultado del trabajo del resto del equipo. Es el encargado de gestionar la lista de requisitos al seleccionarlos y priorizarlos, también se asegura de que todos los miembros tengan claridad sobre ellos.
- **“Scrum Master”.** Ayuda a entender a todos los miembros del equipo sobre las reglas y valores en la realización del trabajo. Se asegura que haya claridad sobre los objetivos y el alcance del producto a desarrollar e igualmente orienta sobre cómo aplicar creatividad, autoorganizarse como equipo y eliminar los impedimentos para maximizar la productividad.
- **“Scrum Team”.** Es el equipo encargado de ejecutar las tareas de cada iteración para entregar un incremento del producto de acuerdo con las especificaciones entregadas por el “Product Owner”. Organizan y

gestionan su propio trabajo y están conformados por el número de personas necesarias para lograr el cumplimiento de los objetivos trazados.

Para complementar el tema, se recomienda realizar la lectura Metodología, la cual se encuentra en el material complementario.

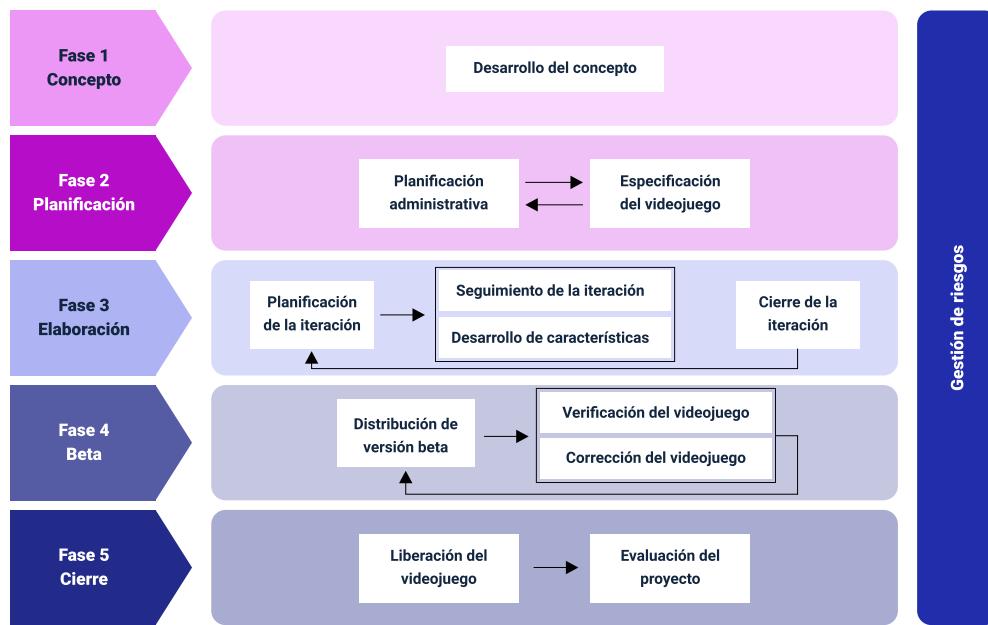
1.3. Metodología SUM

Es un método ágil que se basa en SCRUM para el desarrollo de proyectos, es fácilmente combinable con otras metodologías ágiles por lo que es adaptable fácilmente al desarrollo de videojuegos, ya que se pueden prever y administrar los riesgos, los recursos y así mismo los resultados obtenidos. Se destacan las siguientes características:

- a)** El cliente participa en todas las fases del proceso.
- b)** Los equipos de trabajo deben ser multidisciplinar y máximo de siete (7) integrantes.
- c)** Los proyectos por desarrollar deben ser pequeños.
- d)** El tiempo de ejecución de un proyecto debe ser menor a un año.

En la siguiente imagen se presenta la estructura de SUM según Acerenza (2009):

Figura 3. Estructura de SUM



- **Concepto.** En esta etapa el equipo de trabajo y el cliente proponen ideas que precisen el tipo de producto a desarrollar, se hacen bocetos para identificar las características que poseerá, se define el alcance y los objetivos, la forma de lograrlos y los retos que se deben afrontar. Esta fase termina con un boceto o prototipo que resume el concepto del producto, que en el caso del videojuego contiene el nombre, la historia, el entorno y los personajes que harán parte de este.
- **Planificación.** Tiene que ver con la preparación de las actividades que se deben llevar a cabo para la ejecución del proyecto. La planificación administrativa se encarga de organizar todos los aspectos para el cumplimiento de los objetivos del proyecto, como el equipo de desarrollo, el cronograma y el presupuesto.

Las especificaciones del proyecto, que este caso es el desarrollo de un videojuego, es la definición de todos los requisitos y funcionalidades que tendrá al igual que su priorización.

- **Elaboración.** En esta etapa se refinan los requisitos seleccionándolos y priorizándolos de nuevo para comenzar con el desarrollo de la iteración; se monitorea el cumplimiento de los objetivos de la iteración aplicando técnicas de evaluación definidas con anterioridad.
Para cerrar la iteración, se le muestra al cliente lo obtenido, se da por culminada la iteración y se planifica la siguiente.
- **Beta.** Antes de liberar la versión del producto, se verifica la totalidad de sus funcionalidades, se hace un reporte de los ajustes que se deben realizar para dejarlo a punto, se ejecutan los cambios y posteriormente se hace la publicación y distribución.
- **Cierre.** En esta fase se libera la versión del producto que ha sido ajustada de acuerdo con las verificaciones y se hace una evaluación de lo sucedido durante el desarrollo, teniendo en cuenta las dificultades presentadas y los logros alcanzados para documentar las lecciones aprendidas y mejorar la productividad del equipo.
- **Gestión de riesgos.** Se identifican y describen los riesgos a los que se ve expuesto el proyecto, determinando su probabilidad de ocurrencia y el impacto que tienen en el desarrollo del producto, determinando las estrategias que se pueden aplicar para mitigarlos y elaborando un plan de contingencia para saber cómo actuar en caso de que el riesgo ocurra.

Se pueden distinguir los siguientes roles:

- **Cliente.** Se encarga de validar el concepto definido para el producto, priorizando los requisitos que le dan mayor valor; evalúa el producto al final de cada iteración, haciendo la retroalimentación y proponiendo los cambios a realizar para que se vaya acercando a lo que se definió en el concepto. Valida las versiones que se van a liberar.
- **Productor interno.** Ayuda en la construcción de los objetivos del producto y es el responsable de su planificación y desarrollo, solventando todas las dificultades que se presenten durante la ejecución de las iteraciones. Establece las acciones que se deben implementar para la mejora continua y se comunica constantemente con el cliente para mantenerlo informado de los avances del proyecto en general.
- **Equipo de desarrollo.** Está conformado por diseñadores del concepto del producto, diseñadores gráficos y programadores. El equipo participa en la definición del concepto del producto, define y realiza todas las tareas a ejecutar en cada iteración del producto, estimando la duración de cada una. Participa en la evaluación de la iteración como tal, para ayudar a implementar acciones de mejora.
- **Verificador Beta.** Participa en la fase 4 Beta del proyecto y es el responsable de verificar todas las funciones del producto. El verificador no hace parte del equipo de desarrollo, ya que ejerce un rol imparcial en las pruebas que realiza. Generalmente, utiliza un “software” especializado para hacer el seguimiento y detectar los errores que se deben corregir por parte del equipo desarrollador.

Para complementar el aprendizaje de este tema, se recomienda realizar la lectura SUM para el desarrollo de videojuegos, la cual se encuentra en el material complementario.

2. Programación en Unity 3D

El programa Unity es un motor de tipo gráfico; esto quiere decir que contiene muchas funcionalidades que resuelven aspectos en la creación de un videojuego sin que se necesite programar; por ejemplo, poner color a un objeto seleccionado, iluminar un plano o cambiar el nivel de una imagen entre otros.

Para adicionar algún tipo de comportamiento, componente o atributo que no se pueda hacer desde el menú del programa, lo que se debe hacer es escribir líneas de código o “scripts” para que le proporcionen estas características. Esto es lo que se llama programación orientada a objetos.

La Programación Orientada a Objetos -POO- posee las siguientes características:

- **Abstracción.** Consiste en abstraer o aislar las características o atributos esenciales que definen un objeto y que lo distinguen de los demás con el propósito de que las pueda usar otro objeto. Por ejemplo, si el objeto es una puerta, las características esenciales son que tenga un marco, bisagras y cerradura sin importar el tamaño, el color y el tipo de material entre otras.
- **Encapsulamiento.** Es el conjunto de características esenciales que han sido separadas de las no esenciales. En el ejemplo de la puerta en el encapsulamiento se encuentra el marco, las bisagras y la cerradura, ya que son los atributos principales o generales que hacen que el objeto sea definido como puerta, los demás atributos como el color, el material y el tamaño, hacen parte lo que se consideran características, pues estos son atributos que pueden poseer innumerables tipos de objetos.

- **Herencia.** Al encapsulamiento de las características de un objeto se le llama clase; que está conformada por un patrón de atributos que la hacen de orden superior o superclase porque tiene características generales, pero pueden surgir clases hijas o subclases que hereden estas características y se encuentren en orden inferior dentro de la jerarquía.

En el ejemplo de la puerta, las clases hijas que heredan las características son los tipos de puertas que pueden elaborarse como las correderas, plegables, pivotantes, de seguridad, entre otras.

- **Polimorfismo.** Consiste en definir un método con un conjunto de parámetros, el cual puede ser aplicado en varios tipos de objetos. El método puede aumentar o disminuir los parámetros dependiendo del objeto al que se aplique. Por ejemplo, si el método se llama Open y es utilizado para abrir diferentes objetos como: puerta, cuenta bancaria, evento, historia clínica, entre otros. Dependiendo de lo que se abra, el proceso como tal, tiene más o menos actividades por realizar.

El lenguaje de programación que se utiliza en Unity para generar las líneas de código o “scripts” es C#; el cual se compone de variables, funciones y clases que se explicarán más adelante. Para escribir el código, Unity tiene un editor de texto asociado llamado Visual Studio, aunque el programador puede utilizar simplemente el bloc de notas. El editor (Visual Studio) le ayuda a determinar los errores de sintaxis que se puedan cometer en el desarrollo, por esa razón es recomendable hacer uso de este.

Antes de comenzar a estructurar un programa en Unity, revisar la explicación de algunos conceptos básicos en el siguiente enlace:

Fundamentos de programación: algoritmos, estructura de datos y objetos.

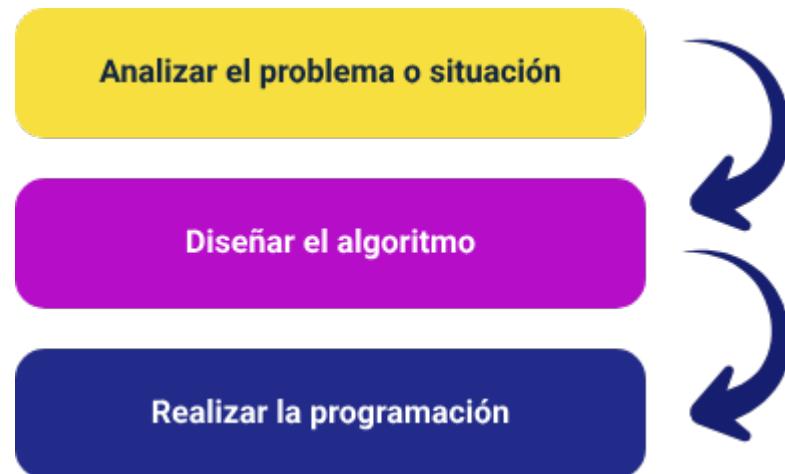
Consultar en la base de datos SENA, ingresando su usuario y contraseña.

<https://www-ebooks7-24-com.bdigital.sena.edu.co/?il=10409>

2.1. Conceptos básicos de algoritmo y programa

Para introducirse en el mundo de la programación, es necesario comprender los procedimientos y técnicas para desarrollar programas, teniendo claro de antemano que estos (programas) se conciben como la solución a un problema o situación determinada. A continuación, se exponen los pasos fundamentales que conducen al desarrollo de un “software”:

Figura 4. Pasos fundamentales que conducen al desarrollo de un “software”



- **Analizar el problema o situación.** Consiste en apoyar al desarrollador en la comprensión del contexto y las variables que determinan la situación o problema presentado. Es la identificación de los datos iniciales que se deben procesar para llegar a la solución.
- **Diseñar el algoritmo.** Consiste en definir una serie de acciones a ejecutar en el proceso y que deben culminar con la solución del problema.

- **Realizar la programación.** Es la codificación del algoritmo en un lenguaje de programación que sea interpretado y ejecutado por el ordenador para que se lleve a cabo la solución del problema.

Para lograr el objetivo de solucionar un problema o tarea, se debe analizar la situación para tener claridad de lo que se pretende, se diseña un algoritmo con los pasos a seguir y se realiza la programación con líneas de código escritas en un lenguaje para ser procesadas y ejecutadas por un ordenador del cual se obtienen los resultados esperados.

Algoritmo

Es el conjunto de pasos o instrucciones ordenadas lógicamente para resolver un problema o realizar una tarea específica. Es uno de los pasos previos antes de entrar a la programación, es decir, antes de escribir las líneas de código. Un algoritmo posee características muy definidas para que resulte efectivo:

- a) **Preciso:** tener un objetivo claramente definido.
- b) **Secuencial:** llevar a cabo una serie de pasos en secuencia y siguiendo un orden lógico.
- c) **Finito:** poseer un número delimitado de pasos.
- d) **Determinado:** obtener un mismo resultado no importa cuántas veces se ejecute.
- e) **Llegar a un resultado:** solucionar el problema detectado o resolver la tarea que se ha planteado.
- f) **Corto:** analizar los tiempos para optimizar los recursos de programación.

Los algoritmos pueden ser:

- **Cualitativos.** Cuando dentro de los pasos lógicos que lo conforman no se involucran operaciones matemáticas para obtener el resultado deseado.
- **Cuantitativos.** Cuando dentro de los pasos lógicos se involucra operaciones y cálculos matemáticos que son necesarios para obtener el resultado esperado.

Y un algoritmo se puede diseñar utilizando la técnica:

- **Pseudocódigo.** Las instrucciones que se deben seguir paso a paso de manera lógica y ordenada se representan en forma descriptiva, elaborando una serie de expresiones a manera de proposición o afirmación sobre un evento.
- **Diagrama de flujo.** Las instrucciones son representadas por medio de símbolos con formas específicas de acuerdo con la acción que se desea realizar. Este concepto se ampliará en el siguiente apartado.

Ejemplo

En la historia “La isla de los tesoros escondidos”, se debe diseñar un algoritmo de búsqueda para encontrar el primer tesoro de cinco (5), el jugador pasará al siguiente nivel solo si lo ha encontrado y podrá terminar la primera fase del juego:

- **Paso 1:** inicio.
- **Paso 2:** mirar en el mapa los datos de la ubicación actual y las rutas que puede seguir.
- **Paso 3:** seleccionar una ruta.

- **Paso 4:** emprender la búsqueda.
- **Paso 5:** ¿encontró un tesoro?
- **Paso 6:** si es verdad pasar al siguiente nivel y terminar fase, de lo contrario volver al paso 2 y repetir la secuencia hasta encontrar el tesoro.
- **Paso 7:** fin.

Más adelante se verá este mismo algoritmo expresado en forma de diagrama de flujo para comprender la diferencia.

Programa

Debido a que el diseño del algoritmo representado en forma de pseudocódigo o diagrama de flujo, no es comprensible por ningún ordenador que vaya a procesar las instrucciones, estas se deben traducir a un lenguaje entendible. Entonces:

Un programa es la codificación de las instrucciones de un algoritmo en un lenguaje de programación, con una sintaxis correcta para que pueda ser interpretado por el ordenador.

La sintaxis correcta se refiere a que cada lenguaje de programación tiene ciertas reglas semánticas que se deben respetarse para que en el momento de ser procesadas por la computadora no se generen errores que impidan lograr los resultados esperados. Al proceso de escribir las líneas de código del programa se le denomina programación y las personas que se dedican a esta labor se les denomina programadores.

- **Características de un lenguaje de programación.**
 - Poseen reglas de sintaxis y semántica que se debe seguir al pie de la letra para que pueda ser interpretado por la computadora.

- Son de alto nivel porque son comprendidos por cualquier persona cuando los escribe al seguir las reglas, por ejemplo Contador = 10; asigna el valor de 10 a la variable Contador.
 - Deben ser traducidos a un lenguaje que entienda el ordenador para que se conviertan en lenguajes ejecutables.
- **Tipos de programas.** Pueden ser de dos tipos:
 - Sistema: son un conjunto de instrucciones que sirven para gestionar el funcionamiento del ordenador como la Unidad Central de Procesos – CPU-, los recursos de memoria como el disco duro y la memoria RAM, periféricos como impresoras y dispositivos de comunicación entre otros.
 - Aplicación: conjunto de instrucciones escritas por programadores para que se ejecuten tareas específicas, por ejemplo, procesar textos (Word), realizar cálculos (Excel), hacer presentaciones (PowerPoint), desarrollar videojuegos (Unity); entre muchos otros.

El proceso que se lleva a cabo para que un programa sea ejecutado por el ordenador y realice la función deseada se presenta a continuación:

- **Diseño del algoritmo.** Es la escritura ordenada y lógica de las instrucciones que se deben llevar a cabo para resolver un problema o situación determinada, representado en forma de pseudocódigo o de diagrama de flujo.
- **Elaboración del programa fuente.** Codificación del algoritmo a través de un editor de código que sirve para escribir las instrucciones del algoritmo en un lenguaje de programación de alto nivel, llamado programa fuente.

- **Compilación del programa.** Al programa fuente se aplica la opción de compilar del editor de código utilizado, para que el ordenador a través de un lenguaje intérprete (interno en la máquina) lo convierta en un programa objeto que puede entender la máquina.
- **Entrega del programa objeto.** El programa que se obtiene producto del proceso de compilación que ha realizado la máquina. Si el programa fuente llega a tener algún error de sintaxis cometido por el programador, el compilador entrega los errores generados para que sean corregidos y se vuelva a realizar la compilación.

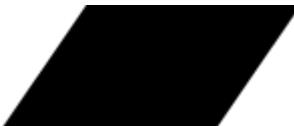
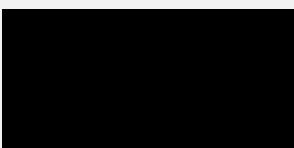
Nota. Proceso llevado a cabo para que un programa desarrollado sea ejecutado por el ordenador. Elaboración Propia (2021).
- **Enlazador**
- **Entrega del programa ejecutable.** Es el programa que se obtiene producto del proceso de compilación que ha realizado la máquina. Si el programa fuente llega a tener algún error de sintaxis cometido por el programador, el compilador entrega los errores generados para que sean corregidos y se vuelva a realizar la compilación.
- **Ejecución del programa.** El programa está listo para ser ejecutado por el ordenador y producir los resultados que se esperan de este.

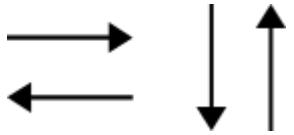
2.2. Diagramas de flujo

Es una herramienta de tipo gráfica que representa el diseño de un algoritmo y está compuesta por símbolos que indican las acciones a realizar para obtener un resultado que resuelve un problema, situación o tarea. Los símbolos están unidos por flechas, las cuales señalan la secuencia lógica de ejecución de dichas acciones. Todo

algoritmo tiene un comienzo y un final representados por sus símbolos respectivos. A continuación, se explica cada uno de ellos, ver tabla 2:

Tabla 2. Símbolos utilizados en los diagramas de flujo.

Símbolo	Descripción
	Indica el inicio y la terminación del algoritmo.
	Símbolo de entrada de datos, indica los valores iniciales que deberán ingresar para ser procesados.
	Realización de un proceso llevado a cabo con los datos introducidos, indica operaciones matemáticas de las cuales se genera un resultado; por ejemplo: $A \leftarrow B + C$ el resultado de sumarle B a C se almacena en la variable A.
	Símbolo que indica la toma de una decisión. Al interior de él se coloca una pregunta que puede ser cierta/falsa y de la cual salen dos flechas que indican la ruta a seguir de acuerdo con la respuesta.
	Símbolos que indican haber alcanzado la solución de la tarea, evento o problema. Al interior se coloca la respuesta obtenida.
	Conector que se utiliza para enlazar dos partes del diagrama de flujo que se encuentran en una misma página o interfaz (pantalla) de usuario.

Símbolo	Descripción
	Conector que enlaza dos partes de un diagrama de flujo donde sus partes se encuentran en páginas diferentes.
	Flechas que conectan los procesos e indican el flujo del diagrama de flujo.

Los diagramas de flujo tienen unas reglas de elaboración, las cuales son:

- a) Se deben elaborar en orden lógico que empieza arriba y termina abajo.
- b) Inician y finalizan con el mismo símbolo.
- c) Las flechas que se coloquen deben indicar el flujo o secuencia de las acciones a realizar.
- d) Los símbolos que indican decisión deben tener dos líneas de salida; una cuando la pregunta o afirmación que se coloca al interior, es verdadera debe tener encima la palabra SI y la otra en el caso contrario, se coloca encima la palabra NO.
- e) Verdadera la cual debe tener encima la palabra SI y la otra en el caso contrario y a la cual se le debe colocar encima la palabra NO.
- f) Lo que se escriba al interior de los símbolos debe tener el menor número de palabras posibles o contener operaciones claras: $Z \leftarrow A - B$.
- g) El diagrama debe probarse con datos reales para verificar su funcionamiento. A esta verificación se le llama “prueba de escritorio”.

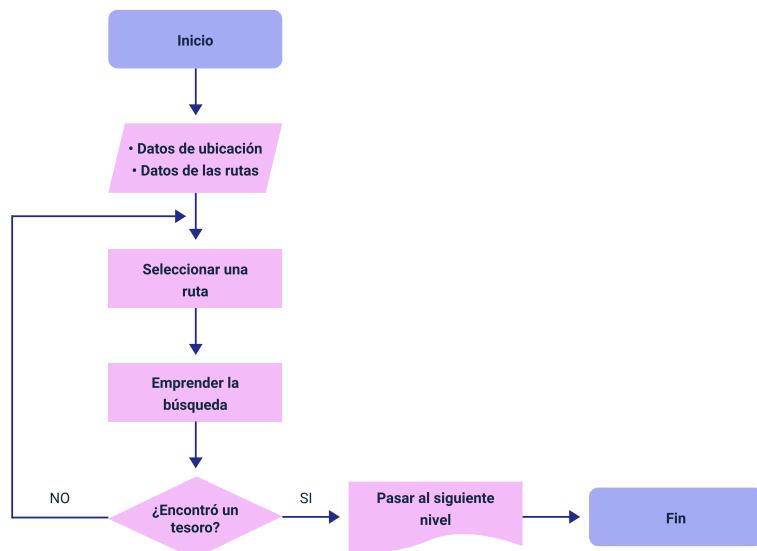
Las ventajas que tiene elaborar un algoritmo utilizando esta técnica, son las siguientes:

- a) Facilita la visualización general del problema, situación o tarea a resolver.
- b) Identifica con facilidad si los pasos que se han diseñado para la solución del problema tienen la secuencia lógica para lograr una solución adecuada.
- c) Permite verificar con datos reales si el diseño de las instrucciones logra la solución planteada.
- d) Comprueba si hay procedimientos que se repiten.

Ejemplo

En la siguiente figura se observa el diagrama de flujo del ejemplo de la historia “La isla de los tesoros escondidos”, en la cual se había elaborado un algoritmo con la técnica de Pseudocódigo. En este diagrama, En este diagrama (figura 5), se ilustra la actividad que se le ha propuesto llevar a cabo al jugador.

Figura 5. Diagrama de flujo de una tarea a realizar en el juego “La isla de los tesoros escondidos”



Este flujo de tareas a realizar es apenas una parte muy pequeña del juego y se puede continuar agregando tareas de acuerdo con las mecánicas definidas, para lo cual se sigue agrandando el diagrama que posteriormente se codificará.

2.3. Conceptos de variables, operadores, funciones, métodos, estructuras de control

En los lenguajes de programación dentro de la sintaxis de las líneas de código; se utilizan elementos o identificadores, los cuales son utilizados dentro de la estructura del programa para lograr la solución del problema, situación o tarea determinada. A continuación, se estudiará cada uno de ellos.

Variables.

Son espacios en la memoria del ordenador con un nombre que las identifica (identificador); los cuales son destinados para el almacenamiento de datos que pueden tomar diferentes tipos de valores durante la ejecución del programa.

Los tipos de datos que pueden almacenar las variables son:

- **Numéricos.** Representados por valores enteros o valores reales que incorporan el signo, por ejemplo, el valor -5 corresponde a un número entero con signo negativo y 4,5 corresponde a un valor real con un solo decimal y sin signo.
- **Lógicos.** Son tipos de datos que representa uno de dos valores “falso” o “verdadero”, utilizados en los algoritmos cuando se diseñan instrucciones en las cuales se debe cumplir una condición para la toma de una decisión; por ejemplo la condición es encontrar el tesoro; si la respuesta es

verdadera puede continuar al siguiente nivel y si es falsa debe seguir buscando.

- **Alfanumérico.** Son datos representados por caracteres diferentes a los números puros o a los lógicos; pues estos pueden contener letras con números y símbolos especiales como guion (-), asterisco (*), símbolo de número (#), entre otros.

Dentro de las líneas de código del programa se debe indicar el tipo de variable que se está creando; es decir qué tipo de valor almacenará; a esto se le llama declarar una variable. Por ejemplo, si la variable se declara tipo numérica, entonces no podrá contener caracteres.

En el lenguaje de programación de C# utilizado en el programa de Unity para el desarrollo de videojuegos, se deben tener en cuenta una serie de reglas para definir variables:

- El nombre de la variable no puede comenzar con un número, pero sí puede hacer parte del nombre.
- No se pueden colocar espacios dentro del nombre.
- No puede contener caracteres como +, - que los cuales son utilizados en operaciones matemáticas.
- No pueden existir dos variables que se llamen igual dentro de un mismo bloque de instrucciones.
- No se pueden utilizar nombres con palabras propias del lenguaje de programación; por ejemplo, no se puede llamar a una variable “int” pues

esta es una palabra que reserva el programa para definir que el tipo de datos es entero.

- El lenguaje hace diferencia entre letras mayúsculas y minúsculas; por ejemplo, las variables X y x son diferentes.
- Se recomienda darle un valor inicial a la variable.

Ejemplo

En este caso se han definido unas variables en C#

Las palabras marcadas en rojo son palabras que hacen parte de la sintaxis del lenguaje; “bool” indica que la variable A es de tipo lógico y que puede contener el valor de true o false y en este caso la variable se inicializó con “false”. La variable B es de tipo numérico que puede tomar valores enteros y se ha inicializado con el valor “80”.

```
bool A = false  
int B = 80
```

Operadores

Son elementos dentro de la programación, con los cuales se realizan operaciones básicas sobre los datos. Cuando el programa fuente es compilado, el ordenador realiza internamente los procedimientos matemáticos o lógicos para entregar el resultado.

Operador	Operandos
Es el elemento como tal, con el que se realiza la operación de adición, substracción, comparación: +, -, <,>	Son los elementos sobre los cuales se aplican las operaciones: variables, números

En el lenguaje de programación C# el cual se utiliza en Unity para el desarrollo de los videojuegos, se distinguen los siguientes tipos de operadores:

- Lo que se quiere es sumar las variables A, B y colocar el resultado en la variable Total
- Limpiar los valores

A = _____

B = _____

Total = _____

Limpiar (A y B): _____

- **Operadores de Asignación.** Se utilizan para asignar un nuevo valor a una variable que ha sido declarada dentro del programa. En la izquierda de la asignación se coloca el nombre de la variable y en la derecha el valor que se asigna. Ejemplo:

string logo; ⇒ se ha declarado la variable logo de tipo texto

int total; ⇒ se ha declarado la variable total de tipo numérico

logo = ETR; ⇒ se ha asignado a la variable logo (izquierda) el valor ETR (derecha)

total = 28; ⇒ se ha asignado a la variable total (izquierda) el valor 28 (derecha)

- **Operadores aritméticos.** Están conformados por los símbolos que permiten hacer operaciones matemáticas con los números como suma (+), resta (-), multiplicación (*), división (/); modular (%); este último devuelve como resultado el residuo de una división. Ejemplo:

Long A; ⇒ se ha declarado la variable A de tipo entero almacenando números muy grandes, diferente al tipo “int” que almacena números enteros muy pequeños.

Int B; ⇒ se ha declarado la variable B de tipo entero

Double C; ⇒ se ha declarado la variable C de tipo decimal albergando hasta quince números decimales después del punto.

A = 25 * 30;

B = 5 * 3;

C = 3 / 2;

- **Operadores Relacionales.** Estos operadores se utilizan para realizar la comparación entre dos valores donde el resultado es verdadero si la afirmación de la expresión comparada es correcta, de lo contrario el resultado es falso. Los símbolos para comparar las variables son:

Ejemplo:

bool total; ⇒ se ha declarado la variable total de tipo booleano es decir que solo puede tomar uno de dos valores; “verdadero” o “falso”

total = 8 > 9 ; ⇒ esto da como resultado falso (false).

Operador	Ejemplo	Descripción
<	a < b	a menor que b
>	a > b	a mayor que b
\leq	a \leq b	a menor o igual que b
\geq	a \geq b	a mayor o igual que b
==	a == b	a igual que b
!=	a != b	a distinto que b

- **Operadores Lógicos.**

- Estos operadores se utilizan con las tablas de verdad.
- El operador $\&\&$ (“AND”) es decir “y” ; solo devuelve un valor verdadero si las dos expresiones son verdad.
- El operador $\|$ (“OR”) es decir “o”; solo devuelve un valor verdadero si una de las dos expresiones es verdadera.
- El operador $!$ (“NOT”) es decir “no”; devuelve un valor verdadero si la expresión es falsa y devuelve un valor falso si la expresión es verdadera.

Ejemplo:

Int a = 10; \Rightarrow se ha declarado la variable a de tipo entero

Int b = 20; \Rightarrow se ha declarado la variable b de tipo entero

Int c = 8; \Rightarrow se ha declarado la variable c de tipo entero

bool total; \Rightarrow se ha declarado la variable total de tipo booleana

total = a < b && b > c; ⇒ el resultado es verdadero ya que a < b es verdad y b > c también es verdadero.

- **Operadores de Incremento.** Estos operadores se llevan a cabo incrementando o disminuyendo el valor de la variable en una unidad. El operador que indica incremento es “++” y el que indica disminución es “—”. Ejemplo:

Int a = 5; ⇒ se ha declarado la variable a de tipo entero

a ++; ⇒ se ha incrementado la variable a en 1 y ahora su valor es 6

a ++; ⇒ se ha incrementado la variable a en 1 y ahora su valor es 7

a ++; ⇒ se ha incrementado la variable a en 1 y ahora su valor es 8

a —; ⇒ se ha disminuido la variable a en 1 y ahora su valor es 7

Funciones y métodos

Son procedimientos dentro de la programación llamados bloques y en los cuales se escriben líneas de código o instrucciones para llevar a cabo una determinada acción dentro del programa. Estos elementos le proporcionan una estructura modular al programa, ya que líneas de código pueden ser invocados en otra parte del programa, sin necesidad de repetirlas o también se pueden invocar en otros programas para no escribirlas de nuevo. La diferencia entre estos dos procedimientos radica en la siguiente característica:

- **Función.** Es un bloque de instrucciones que cuando se ejecutan retornan siempre un valor de salida.

- **Método.** Es un bloque de instrucciones que cuando se ejecutan no necesariamente retornan un valor.

De acuerdo con lo anterior, todos los bloques de instrucciones que lleven a cabo alguna acción dentro del programa se llaman métodos.

Ejemplo

En el lenguaje de programación C# (que se utiliza en Unity):

```
private int local (int A, int B)
{
    Int total = A + B;
    return total;
}
```

La instrucción para realizar la primera parte corresponde a una función dado que devuelve un valor, entonces:

La primera instrucción indica que el procedimiento que se va a realizar se llama local el cual es de tipo privado y devuelve un valor entero, contiene dos parámetros: una variable de tipo entera llamada A y otra del mismo tipo llamada B.

Los procedimientos pueden ser de tipo “private” o “public”, es decir, si el bloque de instrucciones solo se puede utilizar en el programa actual es “private”, pero si se puede utilizar en otros es “public”. El procedimiento suma las variables A y B retornando el resultado en la variable total. Como hay un resultado que se genera el procedimiento es una Función.

```
private void limpiar ()  
{  
    txtA. clear ();  
    txtB. clear ();  
    txttotal. clear ();  
}
```

La primera instrucción indica que el procedimiento que se va a realizar se llama limpiar el cual es de tipo privado (“private”) y retorna un valor nulo (“void”), en las cajas de texto (txt) de las variables A,B y total. Obsérvese que como el procedimiento no devuelve un valor, entonces es un método.

Estructuras de control

Las estructuras de control permiten que se ejecuten los procedimientos o bloques de instrucciones de una manera secuencial, condicional o repetitiva.

- **Estructura de control secuencial.** Permite que las instrucciones de un procedimiento se ejecuten una tras otra, es decir en secuencia.

Ejemplo: en lenguaje C#; las siguientes instrucciones se deben llevar a cabo en secuencia:

- Ingresar el primer valor y en pantalla debe aparecer “Ingresar el primer número:”
- Ingresar el segundo valor y en pantalla debe aparecer “Ingresar el segundo número:”
- Sumar los dos valores ingresados y en pantalla debe aparecer “ la suma es:”

```

{
    Int Numero1, Numero2, Suma;                      ⇒ se declaran 3 variables de tipo entero
    string linea;                                     ⇒ a continuación los datos que se escribirán son de tipo texto
    Console.Write ("Ingrese primer valor: ");          ⇒ escribe en pantalla (consola) ese texto
    Linea = Console.ReadLine ();                      ⇒ capturamos en pantalla el número introducido
    Numero1 = int.Parse (linea);                      ⇒ convierte la línea ingresada en un número entero
    Console.Write ("Ingrese segundo valor:");          ⇒ escribe en pantalla (consola) ese texto
    Linea = Console.ReadLine ();                      ⇒ capturamos en pantalla el número introducido
    Numero2 = int.Parse (linea);                      ⇒ convierte la línea ingresada en un número entero
    Suma = Numero1 + Numero2;                         ⇒ realiza la suma y almacena el valor en la variable Suma
    Console.Write ("La suma es:");                     ⇒ escribe en pantalla (consola) ese texto
    Console.WriteLine (Suma);                          ⇒ escribe en pantalla (consola) el valor de Suma
}

```

- **Estructura de control condicional.** Permite que se ejecuten instrucciones siempre y cuando se cumpla una condición y el formato de la sintaxis se escribe con los códigos if y else.

Ejemplo: en lenguaje C#:

- La condición es que la venta sea mayor a 50 dólares
- Si es verdadero se aplica descuento
- Si es falso no se aplica el descuento

```

{
    Int venta = 100;                                ⇒ la variable
    de tipo entero tiene un valor de 100
    If (venta > 50) {                               ⇒ si (if) la
        variable venta es mayor que 50 ejecute lo que sigue
        Console.WriteLine ("Aplicar descuento");      ⇒ escribe en pantalla
        (consola) ese texto
    } else {                                         ⇒ si es falso
        (else)
            Console.WriteLine ("No aplicar descuento"); ⇒ convierte la línea ingresada
            en un número entero
    }
}

```

- **Estructura repetitiva.** Para complementar el aprendizaje de las temáticas se recomienda la lectura de los textos Documentación de C# y Fundamentos del lenguaje C#, los cuales se encuentran en el material complementario.

Permite que se ejecuten un bloque de instrucciones varias veces hasta que se cumpla una condición. La sintaxis es se escribe con el código while (mientras).

Ejemplo: en lenguaje C#:

- Imprimir los múltiplos de 2
- Hasta que llegue a 20

```
{  
    Int multiplo;                                ➔ se  
    declara la variable multiplo de tipo entera  
    Int multiplo = 2;                            ➔ la  
    variable multiplo tiene un valor inicial de 2  
    While (multiplo <= 20);                      ➔ mientras multiplo sea  
    menor o igual a 20 ejecute lo que sigue  
    {  
        Console.Write (multiplo);                  ➔ escribe en la pantalla  
        (consola) el valor  
        Console.Write (",");                     ➔ escribe en la  
        pantalla (consola) la coma (,) para separar los múltiplos  
        multiplo = multiplo + 2;                  ➔ calcula los múltiplos de 2 y el  
        ciclo se repite hasta que se cumpla el While  
    }  
}
```

Para complementar el aprendizaje de las temáticas se recomienda la lectura de los textos Documentación de C# y Fundamentos del lenguaje C#, los cuales se encuentran en el material complementario.

2.4. Funciones de eventos y control de acciones

En la programación orientada a objetos se mencionaron las características de abstracción, encapsulamiento, herencia y polimorfismo; aspectos que se aplican a los objetos de una situación o tarea determinada y el desarrollador construye líneas de código que son aplicadas a estos casos:

- a) Cuando se desarrollan programas que resuelven problemas o tareas se construyen bloques de instrucciones que realizan determinadas funciones

y actúan sobre los objetos para que estos se comporten de cierta manera y generen resultados específicos.

- b) Cuando se desarrollan programas en los que hay fuerte interacción del usuario con las funciones de éste, como es el caso de los videojuegos; ocurren lo que se llaman eventos.

Evento

Es entonces una acción que es ejecutada sobre un objeto del programa y que es provocada por el usuario al interactuar con la interfaz de este (programa), recuérdese que la interfaz es la pantalla inicial que el usuario visualiza y en la cual puede llevar a cabo acciones.

Cuando se estén escribiendo las líneas de código, se deben tener en cuenta los eventos que se pueden dar sobre los objetos creados dentro del programa; por ejemplo, cuando el usuario hace clic sobre un botón, ocurre algo al interior del sistema que hace que se active alguna función determinada. Los eventos incluyen:

- “Mouse”
- Teclado
- Interfaz de usuario
- Acciones que se activan cuando ocurren

Funciones y controlador de eventos

Cuando ocurre un evento sobre algún objeto del programa se activa el bloque de instrucciones que contienen la función que debe entrar a operar para controlar el evento. Esta función despliega una serie de acciones que lleva a cabo el sistema para

controlar el evento y responderle al usuario en la interfaz con la que él está interactuando.

Las ventajas de tener en cuenta los eventos en el desarrollo de programas radican en que:

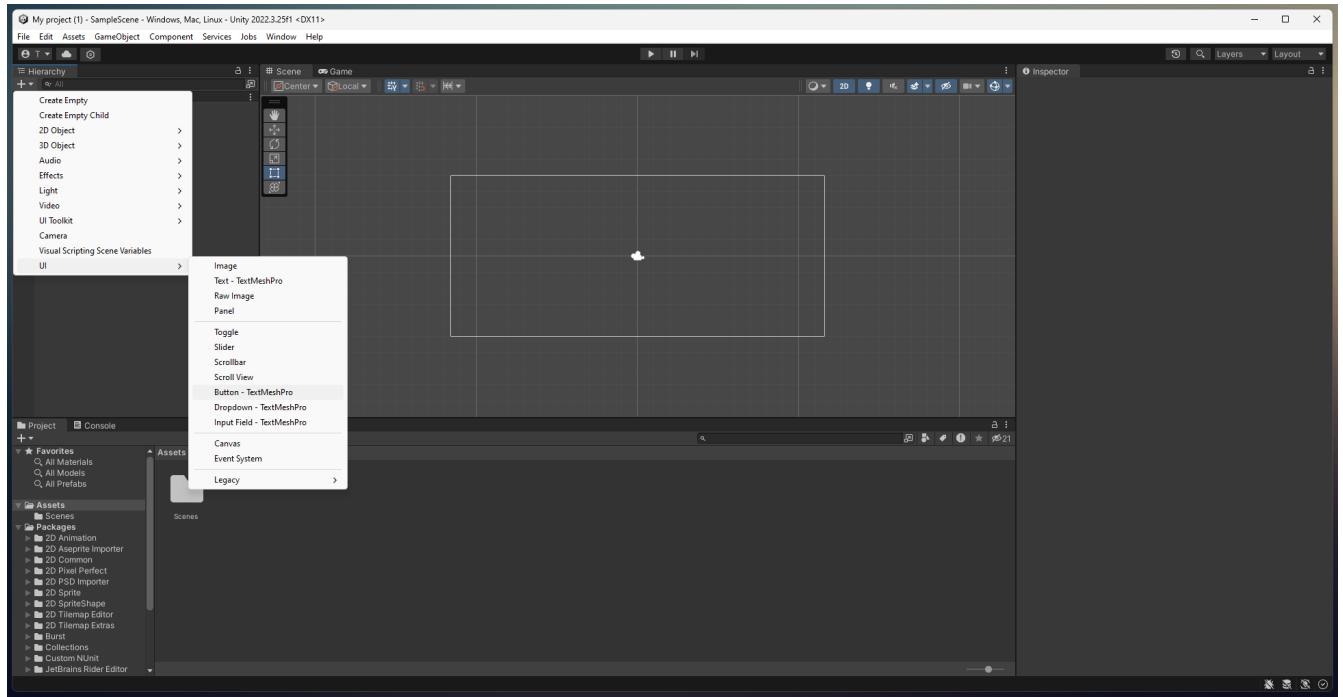
- a) El procesamiento de las funciones es mucho más rápido, ya que como se han tenido en cuenta las acciones que puede realizar el usuario al interactuar con el programa, se han desarrollado líneas de código que las activen cuando estos eventos ocurran.
- b) Se mejora la experiencia del usuario, ya que se tienen en cuenta todas las interacciones que este puede realizar con el programa y para cada una de ellas se planifican y ejecutan respuestas que lo ayuden a encontrar caminos o soluciones a lo que está buscando.
- c) Se minimizan las líneas de código, ya que es el usuario quien genera el evento y el sistema se encarga de responder.
- d) La interfaz del programa es más amigable, ya que ofrece opciones como botones, menús, ventanas, casillas de verificación, entre otras y los usuarios pueden escoger cuál de estos utilizar para navegar en él.

Ejemplo

En Unity se desea crear un evento en el cual el usuario presione el botón “Exit” y salga del juego. En el proyecto en el que se está trabajando realizar las siguientes acciones:

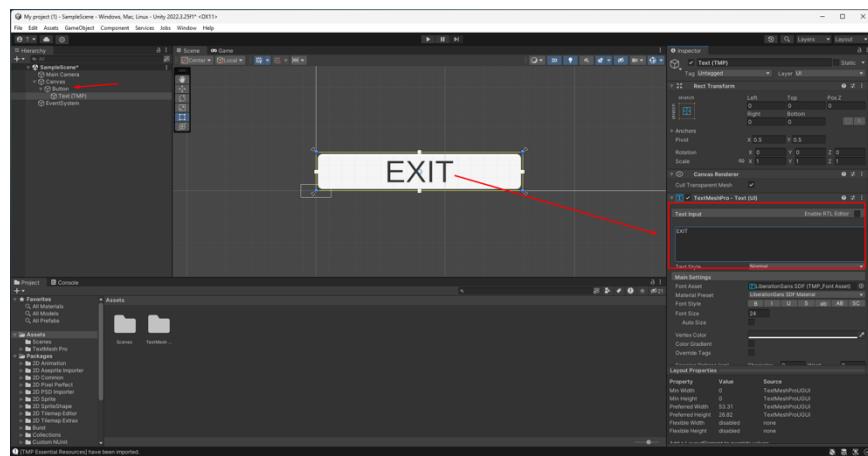
- **Evento.** En el programa Unity, dar clic en la ventana de jerarquía
 - Clic en la opción UI (Interfaz de Usuario)

- Clic en la opción “button”

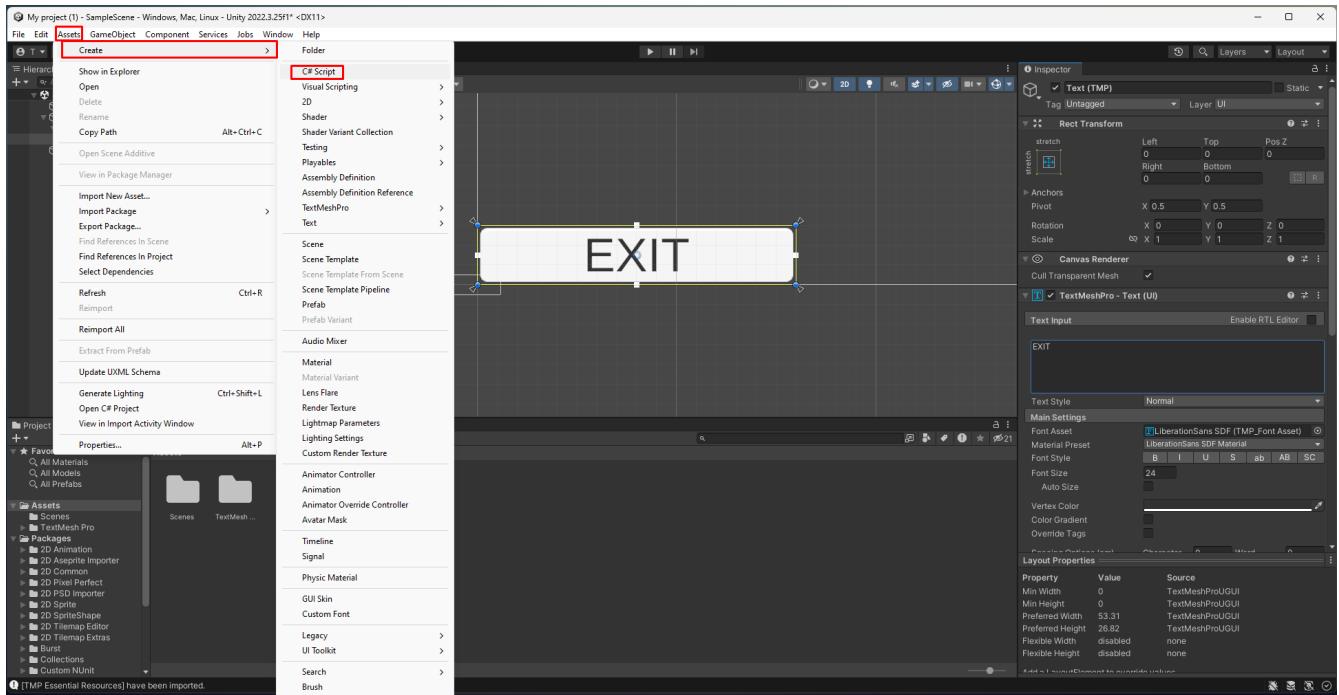


- **Botón.** Una vez creado el botón en la ventana de jerarquía se adiciona el objeto creado dentro de la plantilla “Canvas” y se pueden realizar las siguientes opciones:

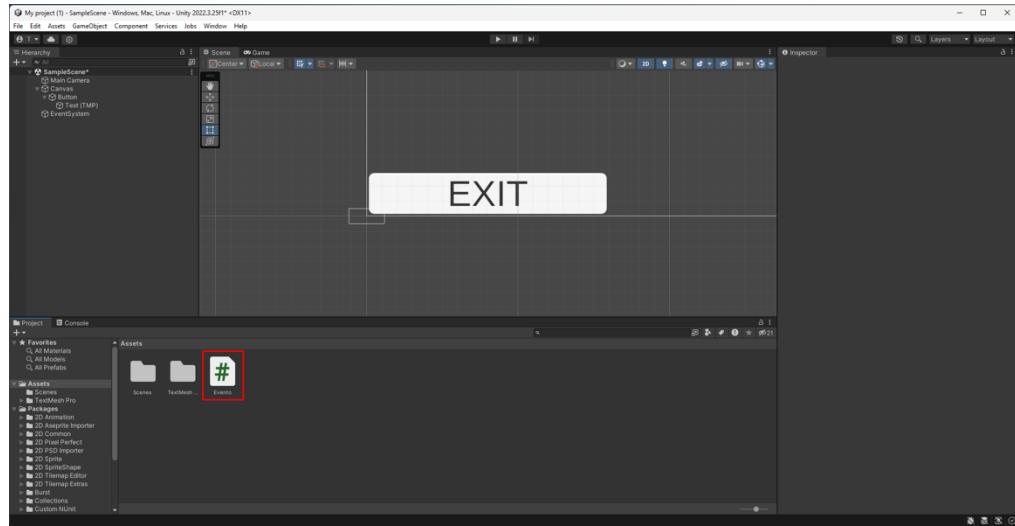
- Doble clic en la ventana de jerarquía en la opción “button”
 - Clic en el botón del menú para cambiar el tamaño arrastrando los puntos de los extremos
 - Doble clic en la ventana de jerarquía en la opción Text para cambiar el nombre del botón que este caso se va a llamar “Exit”.



- “**Script**”. Ya creado el botón debe asociarse la acción que va a realizar, que en este caso es salir y para ello se crea un “script” o líneas de código:
 - Clic en menú Assets
 - Clic en la opción Create
 - Clic en la opción C# Script



- **Archivo “script”.** En la parte inferior en la ventana de proyecto se crea el archivo donde vamos a escribir las líneas de código, en este caso se le colocó el nombre Salir y en la ventana del inspector se muestra unas instrucciones con la estructura básica del programa:



- **Editor.** Al hacer doble clic en el archivo C# denominado Salir se abre la ventana del editor Visual Studio en el cual se podrán escribir las líneas de código que ejecutarán la acción de salir.
En este caso se ha declarado una función de tipo pública (“public”), es decir, que se puede utilizar en cualquier y por otros programas y además tiene la característica “void”, que quiere decir, no devuelve ningún tipo de valor sino que ejecuta la acción. La sentencia “Application.Quit” ejecuta la acción de abandonar.

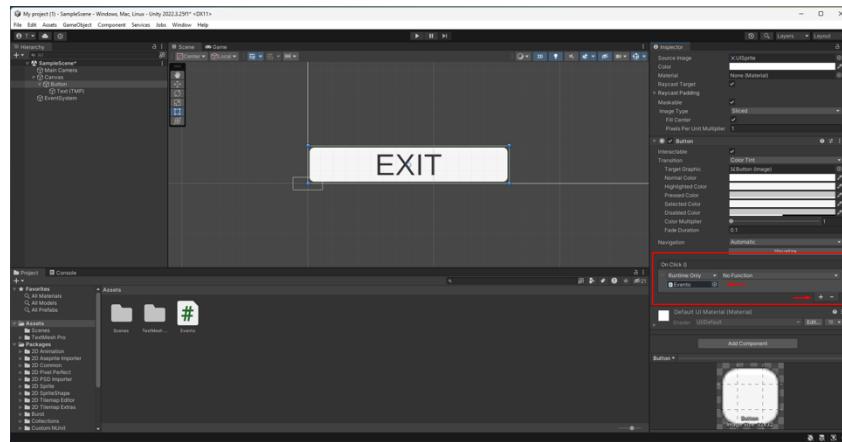
```

File Edit Selection Find View Goto Tools Project Preferences Help
Editor < Salir.cs >
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Salir : MonoBehaviour
6 {
7     public void SalirEvent()
8     {
9         Application.Quit();
10    }
11 }
12

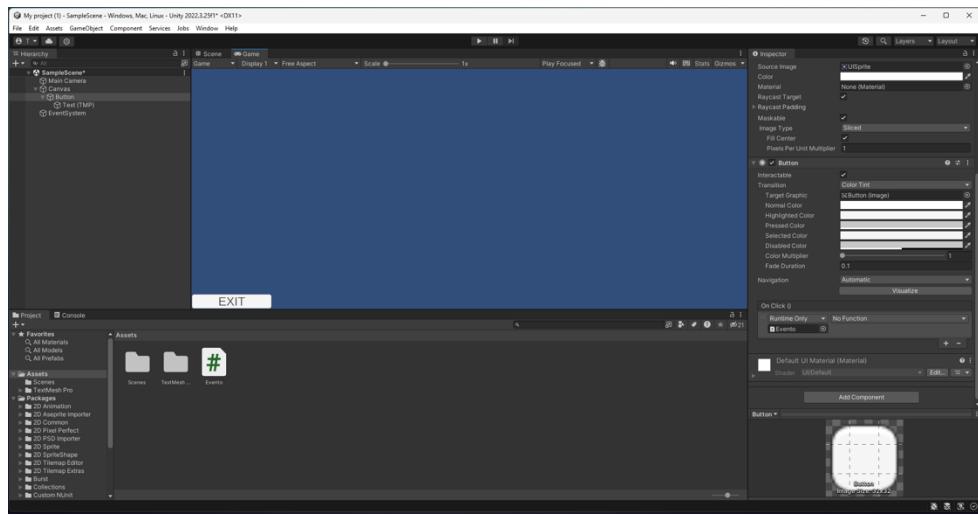
```

Line 5, Column 35 Spaces: 4 C#

- **Asociar.** Una vez construida la línea de código se debe anexar el evento en el inspector para que pueda ser ejecutado posteriormente.
 - En la barra 1 On Click () se debe asociar la función del evento
 - Clic en 2 el signo + para agregar la función salir
 - En el botón Runtime Only se ha agregado la función Salir (3)



- **Juego.** Cuando la función está lista para ser utilizada se observa en modo “game” y no en modo escena para visualizar cómo se verá el botón del evento en la interfaz del usuario para que pueda ser accionado por éste.



2.5. Estructura básica de un “script” de Unity 3D

El lenguaje de programación que se utiliza en Unity es el lenguaje C# y a las líneas de código que lo configuran se les llama “scripts”, las cuales consisten en una secuencia de instrucciones que se escriben en un editor de texto que es adjuntado en Unity al hacer su instalación en el PC; este editor de escritura es el Visual Studio.

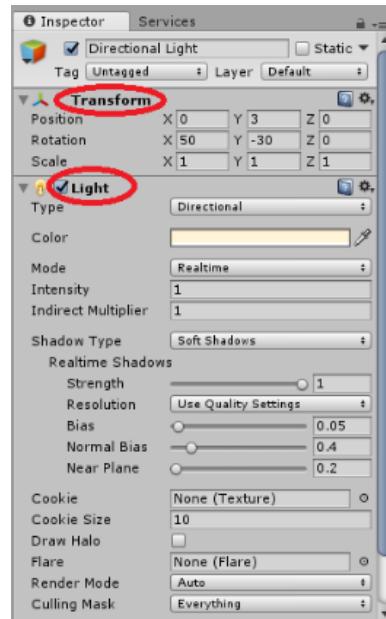
El objeto de escribir líneas de código en Unity es para asignarle comportamientos específicos a los componentes del juego (“GameObjects”), crear eventos y desarrollar funciones que controlen acciones determinadas.

El lenguaje C# se compone de:

- **Variables.** Como se mencionó antes son espacios reservados en la memoria para almacenar valores de diferentes tipos, ya se vio en los anteriores apartados su tipología (numéricos, lógicos, alfanuméricos), pero en Unity también hay tipos de datos de referencia que se aplican a los objetos como “Transform y Light” que se visualizan en la ventana del Inspector; donde la primera trata con aspectos de posición, rotación y tamaño del objeto y la segunda con efectos de iluminación.

Las variables deben ser declaradas al comienzo del bloque de las líneas de código, definiendo si van a ser públicas o privadas. Una variable pública es aquella que puede ser utilizada por otros “scripts” y cambiar su valor porque es accesible para ellos, ya que aparece en la ventana del Inspector y una variable privada es aquella que solo puede ser utilizada dentro de una misma clase (ya se verá este término a continuación) y dentro de un mismo método o función y no puede ser visualizada en el Inspector porque sus valores no son manipulables por otros “scripts”.

Cuando se codifica una variable se debe tener en cuenta aspectos de nomenclatura como que el nombre no debe empezar por un número y no debe tener espacios.

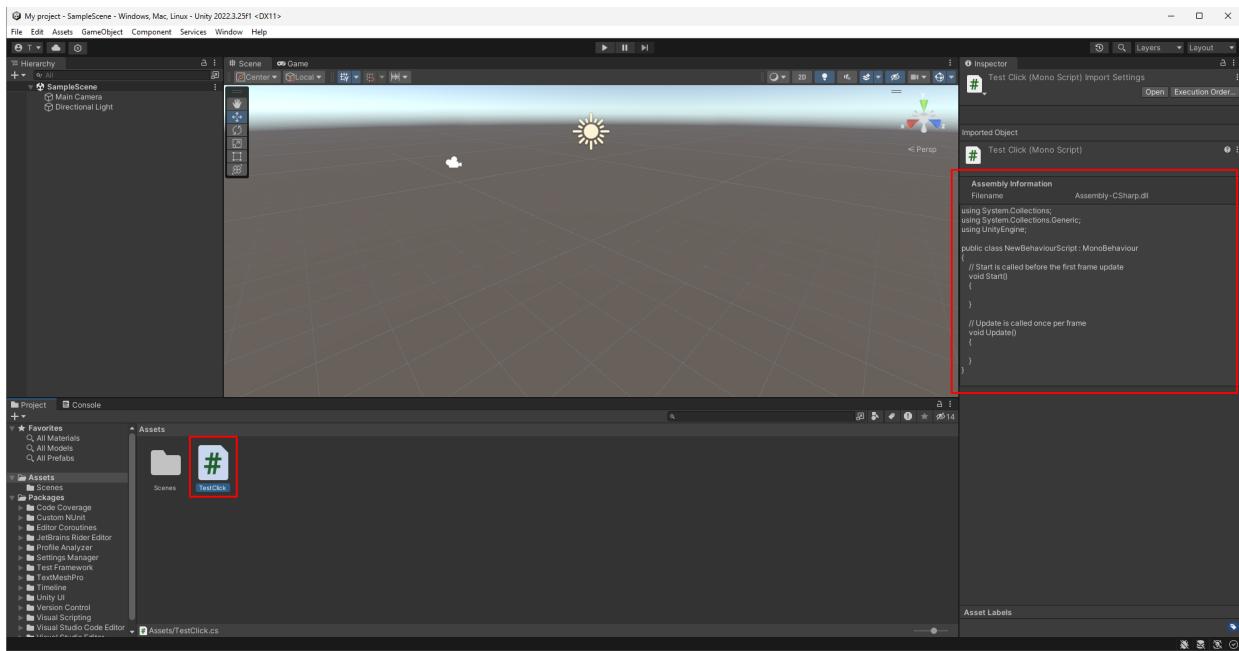


- **Funciones.** Una función en Unity es un procedimiento que realiza acciones sobre un objeto, a la función también se le llama método y consiste en un bloque de líneas de código que hacen un llamado a la acción sobre los componentes en los que se aplica. Cuando se va a construir un “script” en

Unity se activa la clase “Mono Behaviour” que es la clase base la cual llama funciones automáticas que se deben activar para iniciar el “script” como son:

- **“Awake”**: función que se encarga de llamar a todas las variables declaradas en el “script” y que necesitan ser inicializadas con un valor.
- **“Start”**: función que es llamada para inicializar el procedimiento.
- **“Update”**: función que se utiliza para el movimiento de los objetos sin utilizar el método física de Unity y se repite cada vez que se necesite para actualizar el estado de este.
- **“FixedUpdate”**: función que se utiliza para controlar objetos cuando dependen de la física.

Ejemplo: en la siguiente escena en Unity se van a adicionar unas líneas de código para crear un evento sobre el plano y el cubo para lo cual se creó un archivo “script” “TestClick” que al ser llamado desplegó en la ventana del Inspector el llamado a la clase base “Mono Behaviour” que a su vez activo las funciones “Start” y “Update”.



Para escribir líneas de código que se refieran a funciones (métodos) se deben tener en cuenta los siguientes aspectos:

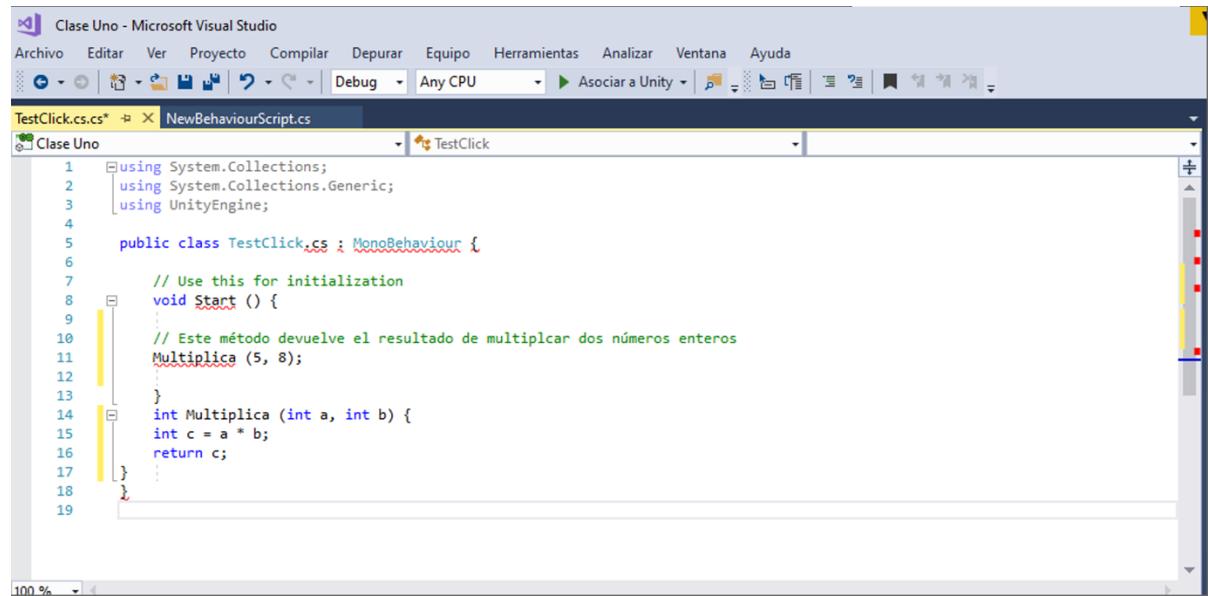
- Puede ser pública o privada
- Debe tener un identificador (nombre)
- Debe Incluir parámetros
- Debe retornar un tipo de dato así sea nulo

Ejemplo: en el editor de texto Visual Studio se ha creado un método con identificador **Multiplica**, el cual contiene dos parámetros de tipo entero (a y b) cuya funcionalidad es multiplicar dos números. El resultado que debe retornar el método (c) es de tipo entero.

El método **Multiplica** es de tipo público es decir puede ser utilizado en otros “scripts” y el nombre del archivo es **TestClick.cs** y el programa ha

llamado la clase MonoBehaviour que contiene las funciones de inicialización por defecto:

```
public class TestClick.cs : MonoBehaviour {
```



```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class TestClick : MonoBehaviour {
6
7      // Use this for initialization
8      void Start () {
9
10         // Este método devuelve el resultado de multiplicar dos números enteros
11         Multiplica (5, 8);
12     }
13
14     int Multiplica (int a, int b) {
15         int c = a * b;
16         return c;
17     }
18 }
```

- **Clases.** Son las que agrupan los atributos y características que van a contener los objetos que se crean en Unity; dentro de ellas se encuentran las variables y los métodos (funciones) que se aplicarán para que se comporte como se ha planeado.

El objeto se comporta de acuerdo con los atributos contenidos en la clase, lo que significa que esta (clase) actúa como un molde para los objetos que se construyan dentro de ella.

Las clases cumplen con las características de abstracción, encapsulamiento, heredad y polimorfismo propias de la Programación Orientada a Objetos – POO-. Algunas de las clases más importantes que se encuentran en el motor de Unity son:

Cuando se están escribiendo líneas de código y se crean varias clases para los objetos las cuales van a agrupar diferentes atributos, entonces se utiliza lo que se llama “Espacios de nombres”; para organizar clases dentro de un mismo entorno, agrupándolas para darle una mejor estructura al “script” y evitar confusiones cuando el motor de Unity haga la compilación de las instrucciones para ser ejecutadas.

Ejemplo: Crear un “script” e identificar los espacios de nombres.

Para crear el “script” hay dos formas:

Assets > Create > C# Script

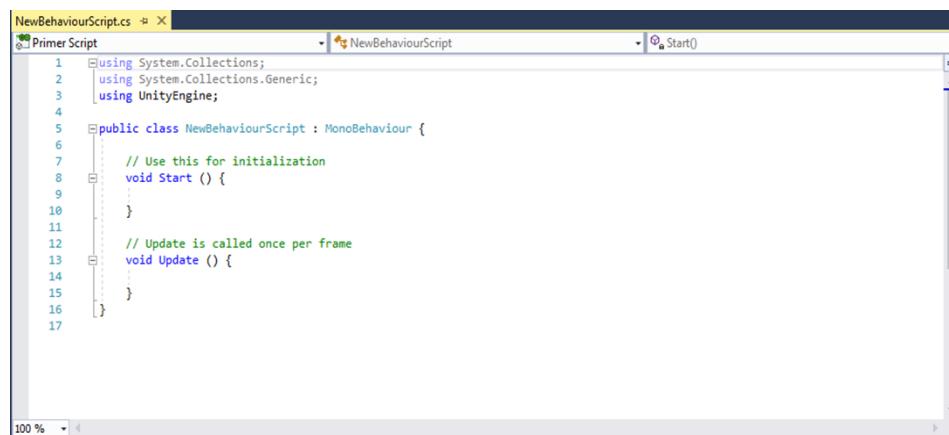
Project > Create > C# Script

Tabla 3. Clases Unity

Clase de Unity	Detalle de la clase
Transform	Proporciona información del objeto en cuanto a su posición, rotación y tamaño.
MonoBehaviour	Es una clase base que contiene todas las funciones que se pueden aplicar a los objetos o eventos que se construyan en Unity.
Rigidbody / Rigidbody2D	Es el motor de física de Unity que se utiliza como herramienta para mover los objetos y aplicar leyes como gravedad, fuerza, masa, aceleración y fricción.

Se ha creado entonces un archivo llamado “NewBehaviourScript.cs” que es el nombre por defecto, el cual puede ser cambiado por el que el usuario necesite para identificar las acciones que realiza.

- **“using System.Collections”**: esta línea de “script” significa que se declara una clase llamada Collections la cual agrupará diferentes objetos dependiendo de sus atributos y funciones.
- **“using System.Collections.Generic”**: dentro de la clase Collections hay elementos que son de un solo tipo. Esto quiere decir que cuando se recupera un elemento de esta clase no hay que definirle el tipo.
- **“using UnityEngine”**: permite que el “script” herede de la clase MonoBehaviour todas las funciones como “awake”, “Start”, “Update”, “Transform”, “Light”, entre otras.



The screenshot shows the Unity Editor's code editor window. The title bar says "NewBehaviourScript.cs". The code itself is as follows:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class NewBehaviourScript : MonoBehaviour {
6
7      // Use this for initialization
8      void Start () {
9          ...
10     }
11
12     // Update is called once per frame
13     void Update () {
14         ...
15     }
16 }
17
```

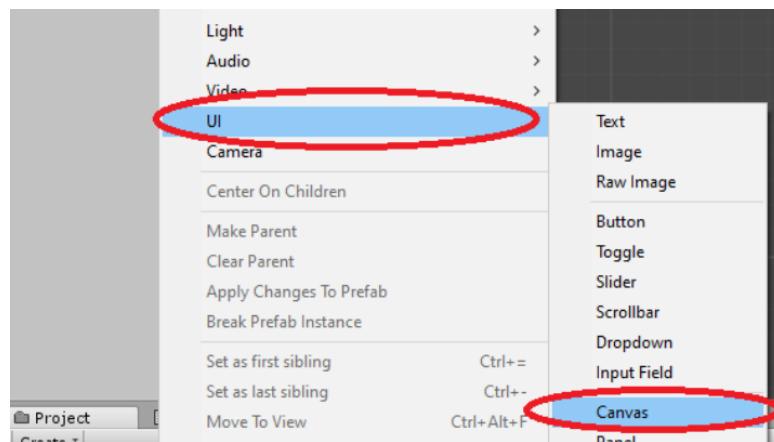
2.6. Control mediante interfaz de usuario (Inspector)

La interfaz de usuario es (UI) es la manera en que este interacciona con el programa; en este caso, la forma en que un jugador se comunica con el videojuego interactuando con él. El motor de Unity posee un componente llamado “Canvas”, que actúa como un contenedor o plantilla donde se pueden colocar todos los elementos u

objetos que van a hacer parte de la interfaz del usuario. “Canvas” quiere decir lienzo y es el espacio donde se van a colocar imágenes, botones, textos, personajes, formas, entre otros que pueden ser controlados a través de la ventana del inspector.

Para utilizar “Canvas” entonces en la interfaz inicial de Unity se realizan las siguientes acciones:

- Paso 1
 - Clic en la opción de menú GameObject
 - Clic en la opción UI
 - Clic en la opción “Canvas”



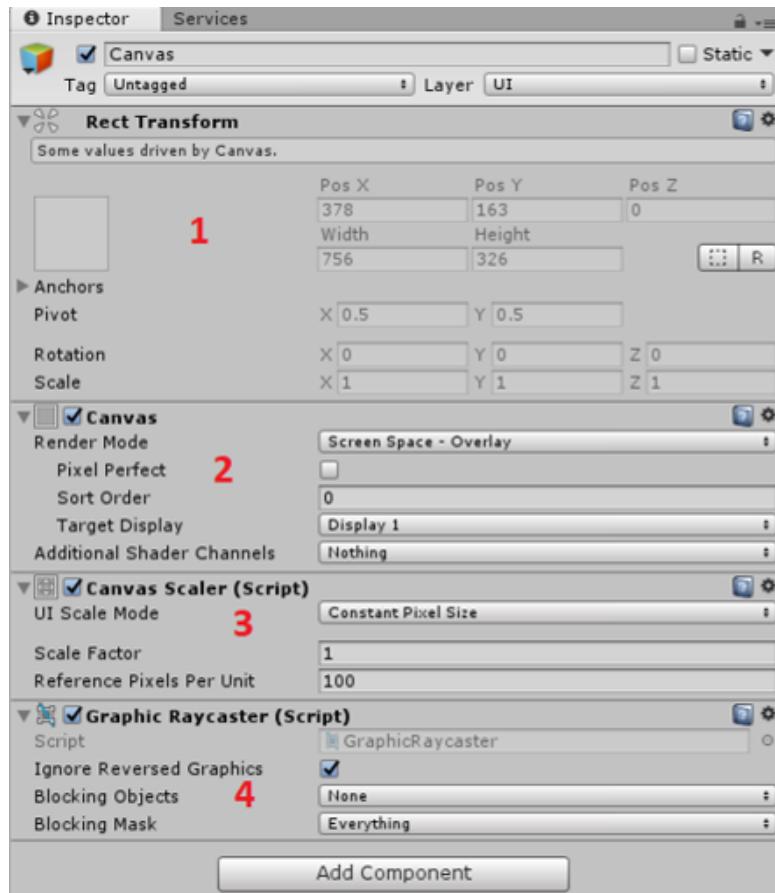
- **Paso 2.** Cuando se ha añadido “Canvas” a la escena en la ventana del Inspector se observa cada uno de sus componentes:

- 1) **“Rect Transform”**: es un componente en el que se establece la posición y tamaño de un elemento en la interfaz del usuario, el ancho y el alto.
- 2) **“Canvas”**: es el área bajo la cual se van a ubicar los elementos que va a contener la interfaz:

- “**Opción Screen Space**” – Overlay indica que la plantilla de “Canvas” se ajustara al tamaño de la pantalla.
- “**Opción Screen Space**” – Camera indica que la plantilla de “Canvas” se ajustara para encajar en el plano de la cámara.
- “**Opción World Space**” –indica que la plantilla de “Canvas” tratará la imagen como un objeto plano en la escena.

3) “Canvas Scaler” (“script”): tiene que ver con la densidad del pixel de los elementos que se coloquen dentro de la plantilla, esto hace que se vea con mayor o menor resolución.

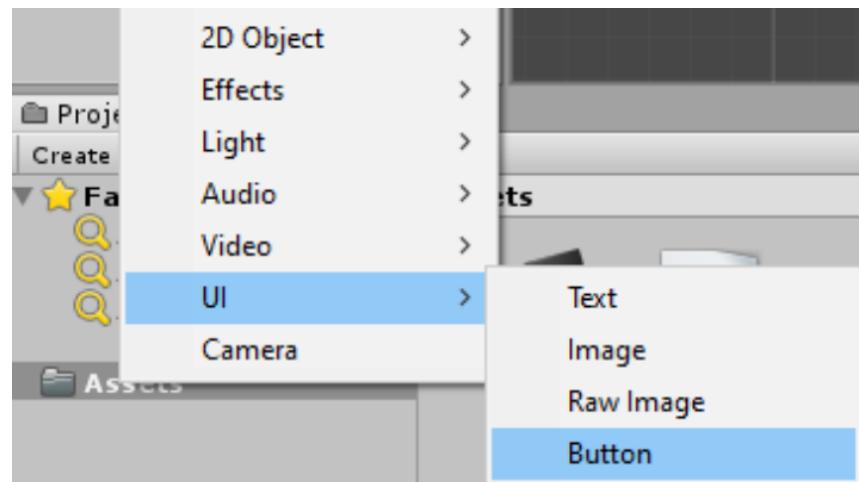
4) “Graphic Raycaster” (“script”): controla los elementos gráficos teniendo en cuenta los que se colocan en el fondo o delante.



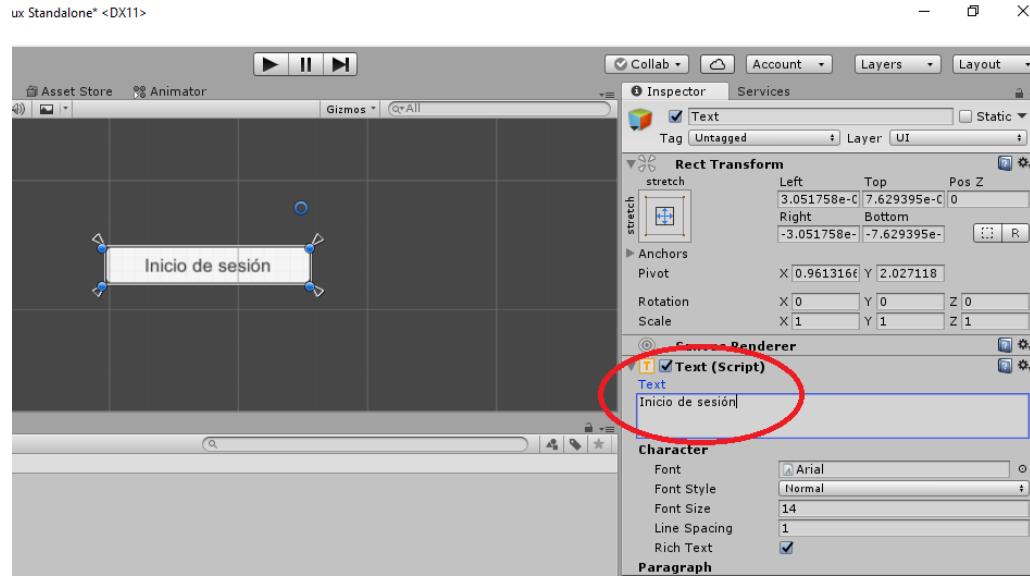
Ejemplo

Construir un botón de inicio de sesión utilizando interfaz de usuario.

- **Paso 3.** Después que se ha llamado la plantilla “Canvas” se realiza:
 - Clic derecho del “mouse” en el componente “Canvas” de la ventana de jerarquía
 - Clic en UI
 - Clic en “button”



- **Paso 4.** Una vez añadido el botón al área de la escena y dentro de “Canvas”, realizar:
 - Doble clic sobre “button” de la ventana de jerarquía para activar texto.
 - Escribir la palabra Inicio de sesión dentro del cajón de texto del Inspector.



Para profundizar en la ventana del Inspector del “Canvas” y en las temáticas de programación en Unity 3D se recomienda consultar los siguientes sitios web:

- ✓ Enlace web. Joyanes Aguilar, L. (2020). Fundamentos de programación: algoritmos, estructura de datos y objetos. McGraw-Hill. <https://www-ebooks7-24-com.bdigital.sena.edu.co/?il=10409>
- ✓ Enlace web. Unity. (2018). Documentation. <https://docs.unity.com/>

2.7. Arreglos

Un arreglo en C# (también conocido como matriz) es una estructura de datos que permite almacenar una colección de elementos del mismo tipo. Los arreglos son útiles cuando se necesita trabajar con muchos valores de datos que son del mismo tipo. En C#, los arreglos pueden ser de una sola dimensión (como una lista lineal de elementos), multidimensional (como una tabla con filas y columnas), o incluso arreglos dentados (donde cada fila puede tener una longitud diferente).

Arreglos y matrices en c#

En C#, que es el lenguaje base de Unity, una matriz es una estructura que representa una colección ordenada de valores u objetos del mismo tipo, de longitud fija.

Una matriz en C# se declara de manera similar a una variable, con la adición de corchetes ([]) después del especificador de tipo para indicar que es una matriz. También se puede inicializar una matriz con valores específicos utilizando llaves ({}). En este caso, no es necesario especificar la longitud de la matriz, ya que se infiere por la cantidad de elementos entre las llaves.

Ejemplos

```
string[] arreglo = new string[8]; // Se declara un arreglo de string de 8 posiciones.
```

```
int[] numeros = { 2, 14, 17 }; // Se inicializa una matriz de enteros con 3 elementos.
```

```
string[] personajes = new string[] { "Paco", "Juan", "Mario", "Sabino" }; // Se inicializa una matriz de cadenas con 4 elementos.
```

En C#, los elementos de una matriz se etiquetan de forma incremental, comenzando en 0 para el primer elemento. Por ejemplo, el tercer elemento de una matriz estaría indexado en 2 y el sexto elemento de una matriz estaría indexado en 5.

Por ejemplo

```
int [ ] números = { 2, 14, 17, 18, 15, 6 } ;
```

2	14	17	18	15	6
0	1	2	3	4	5

En la posición 0 tenemos almacenado el valor de 2, mientras en la posición 5 tenemos el número 6, siendo la posición 0 el primer elemento del arreglo y la posición 5 el sexto elemento de este.

Se puede acceder a un elemento específico utilizando el operador de corchetes, rodeando el índice entre corchetes. Una vez accedido, el elemento puede usarse en una expresión o modificarse como una variable normal.

Para el mismo ejemplo anterior:

números [4] = 20; // El número que estaba en la posición 4 (quinto elemento) fue reemplazado por el valor de 20.

Tamaño de un arreglo

Para determinar el tamaño de un arreglo en C# usamos la propiedad Length. Para el ejemplo anterior tendríamos:

```
int x = numeros.Length
```

En la variable x quedaría almacenado el número 6, ya que el tamaño del arreglo es 6.

Arreglos e iteraciones

Para iterar y recorrer un arreglo, podemos hacerlo mediante el uso de ciclos como “for”, “foreach” o “while”:

- **Ciclo “For”**

```
for (int i = 1; i <= 10; i++)
```

```
{  
    Debug.Log(números[ i ]);  
}
```

- **Ciclo “Foreach”**

```
string[ ] ciudades= { "Pereira", "Armenia", "Bogotá", "Medellín", "Cali", "Barranquilla" };  
  
foreach (string ciudad in ciudades ) {  
  
    Debug.Log(ciudad);
```

- **Ciclo “While”**

```
string[ ] ciudades= { "Pereira", "Armenia", "Bogotá", "Medellín", "Cali", "Barranquilla" };  
  
int i=0;  
  
while (i < ciudades.Length) {  
  
    Debug.Log(ciudades[i]);  
  
    i++;
```

2.8. Listas

Las listas son una estructura de datos esencial que permite almacenar y gestionar colecciones de elementos. Son especialmente útiles para trabajar con múltiples objetos del mismo tipo y ejecutar operaciones sobre ellos.

En Unity, puedes utilizar las listas para guardar distintos tipos de datos, incluidos objetos de juego, números enteros, cadenas o clases personalizadas. Las listas ofrecen una amplia gama de métodos y propiedades que facilitan la manipulación y el acceso a sus elementos.

Crear listas en Unity

Para utilizar listas en Unity, es necesario importar la biblioteca System.Collections.Generic. Para crear una nueva lista, debe usarse la palabra List seguida de corchetes angulares <>, dentro de los cuales se indica el tipo de datos de los elementos de la lista. Luego, se necesita crear una instancia empleando la palabra clave new seguida del constructor de la clase.

```
List<int> ejemplo = new List<int>();
```

Las listas son muy flexibles para asignar nuevos valores. Se pueden declarar e inicializar al mismo tiempo, o bien, se pueden agregar elementos más adelante según sea necesario. Dado que el tamaño de una lista no es fijo, permite agregar tantos elementos como se requieran.

Por ejemplo:

```
List<int> valores=new List<int>{4,1,3,7};
```

Igualmente se pueden agregar nuevos elementos a través del método Add
valores.Add(20) //agrega un nuevo elemento a la lista

Las listas igualmente se pueden adicionar una a otra a través del método AddRange

```
List<int> valores=new List<int>{4,1,3,7};
```

```
List<int> otrosvalores=new List<int>{5,9,2,8};
```

```
void Start()
```

```
{valores.AddRange(otrosvalores); //añade todos los elementos de la lista de otrosvalores a  
valores}
```

Otras operaciones que se pueden realizar sobre las listas son las siguientes:

- **Encontrar el tamaño de una lista.**

```
int tamaniolista= valores.Count;
```

- **Eliminar elementos de una lista.**

```
valores.Remove(10);
```

- **Ordenar la lista.**

```
valores.Sort()
```

- **Borrar la lista.**

```
valores.Clear();
```

- **Encontrar elementos de una lista.**

```
valores.Find(5)
```

- **Copiar una lista.**

```
List<int> copia = new List<int>(valores);
```

- **Invertir la lista.**

```
valores.Reverse();
```

- **Crear una lista a partir de otra lista.**

```
List<int> sublistas= valores.GetRange(1, 3);
```

- **Encontrar el índice de un elemento.**

```
Debug.Log(valores.IndexOf(4)); //busca el número 4 en la lista
```

- **Recorrer la lista.**

```
foreach (int item in valores)
```

```
{
```

```
    Debug.Log(item);
```

```
}
```

3. Programación Orientada a Objetos

La Programación Orientada a Objetos es un paradigma de programación que utiliza objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Es especialmente útil en el desarrollo de videojuegos, donde diferentes elementos del juego, como personajes, enemigos y “power-ups”, pueden ser modelados como objetos con sus propios atributos y comportamientos.

3.1. Clases

En la Programación Orientada a Objetos, cada objeto es una instancia de una clase. Una clase define las propiedades y comportamientos (atributos y métodos) que tendrán sus objetos. Por ejemplo, un videojuego, podría tener una clase Personaje que tiene atributos como vida, velocidad, y fuerza, y métodos como mover o atacar.

Clases en Unity C#

En la programación orientada a objetos, una clase es una agrupación de elementos que comparten las mismas características; está compuesta por métodos y atributos. Un objeto es una instancia específica de una clase; contiene valores concretos en lugar de variables.

La clase es un elemento fundamental en la programación orientada a objetos y se caracteriza por lo siguiente:

- Una clase puede tener subclases que heredan todas o algunas de las características de la clase.
- En relación con cada subclase, la clase original se considera una superclase.

- Las subclases pueden definir sus propios métodos y variables que no son parte de su superclase.
- La estructura de una clase y sus subclases se denomina jerarquía de clases.

¿De qué está compuesta una clase?

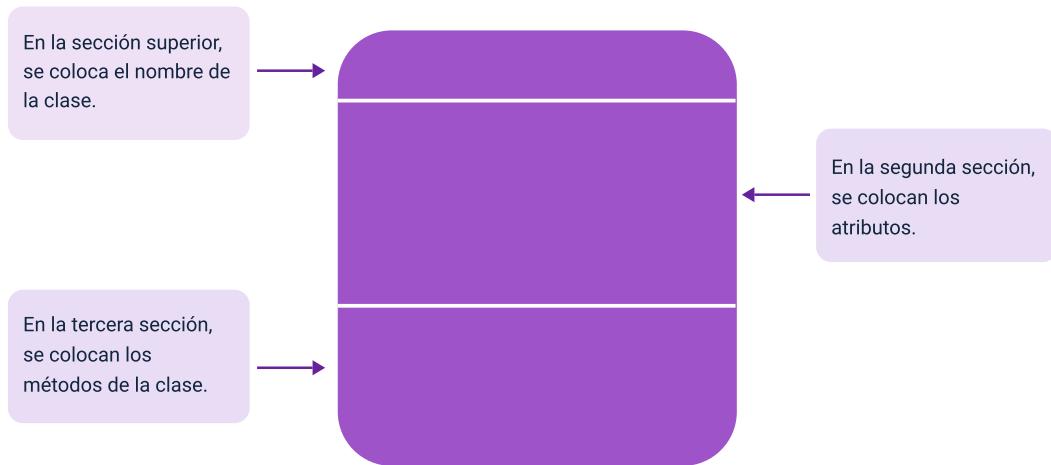
Las clases están compuestas por atributos y métodos:

- **Atributo:** es una característica que define a una clase. En programación, los atributos están representados por variables o incluso otras clases, y nos permiten describir las propiedades de los objetos.
- **Método:** son las acciones que puede realizar una clase. En programación, los métodos están asociados con funciones que definen el comportamiento de los objetos.

¿Cómo se representa una clase?

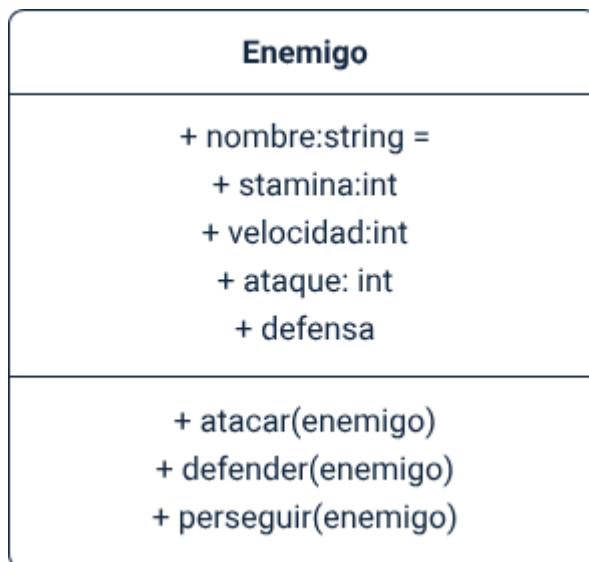
Las clases se representan comúnmente con un rectángulo con las esquinas redondeadas. Este rectángulo está dividido en tres secciones:

Figura 6. Representación de una clase



A continuación, se presenta un ejemplo:

"Enemigo" sería el nombre de la clase, con sus respectivos atributos (nombre, stamina, velocidad, ataque y defensa). Esta clase cuenta con métodos que identifican las acciones que el enemigo puede realizar, como son atacar, defender y perseguir.



Una forma sencilla de identificar qué es una clase, qué es un atributo y qué es un método es a través de su representación lingüística: generalmente, una clase se asocia con un sustantivo, los atributos con cualidades y los métodos con verbos.

3.2. Atributos y cualificadores

En los atributos existen 4 tipos de calificadores que corresponden a los atributos, estos son:

- **Público.** Son aquellos atributos que pueden ser vistos por las demás clases y pueden acceder a sus valores, igualmente pueden modificarlos en tiempo de ejecución.
- **Privado.** Son aquellos atributos que solo pueden ser accedidos por la propia clase. Ninguna clase diferente o heredada podrá acceder directamente a ellos. Si el programador desea que en algún momento sean accedidos o consultados, deberá especificarlo a través de los métodos.
- **Protegido.** Son aquellos atributos que solo pueden ser accedidos por la propia clase y aquellas que heredan de la misma.
- **Estático.** Son aquellos atributos que no pertenecen al objeto, sino a la clase y pueden ser llamados o accedidos sin necesidad de instanciar un objeto. Son conocidos como atributos de clase.

Estos cualificadores también aplican a las clases y a los métodos en la programación orientada a objetos.

3.3. Métodos

Un método es similar a una función en la programación orientada a objetos (POO). Así como un sustantivo suele ser el indicativo de una clase, y una variable es el

equivalente a un atributo dentro de esa clase, los métodos representan las acciones o comportamientos que la clase puede realizar. Un método puede aceptar parámetros como argumentos, procesarlos y, posteriormente, retornar un resultado cuando es invocado en una instancia de la clase. Aunque son similares a las funciones, los métodos se distinguen porque están asociados a una clase y frecuentemente operan sobre los atributos de dicha clase, modificando o utilizando su estado para realizar tareas.

Métodos de interfaz.

Los métodos de interfaz son aquellos diseñados con el propósito de proveer una interfaz a través de la cual un objeto puede interactuar con el entorno externo. Esto incluye, por ejemplo, la comunicación con métodos de otros objetos, la recepción de datos de entrada del usuario, el intercambio de datos con otros objetos, o cualquier interacción que trascienda los límites del objeto en cuestión.

Uno de los principios fundamentales del diseño orientado a objetos es la encapsulación, que implica construir objetos como cápsulas que contienen sus propios datos y métodos. Sin embargo, un objeto completamente aislado no tendría utilidad práctica; para que sea efectivo, debe integrarse dentro de un sistema más amplio. Los métodos de interfaz facilitan esta integración al ofrecer la interfaz mínima necesaria para que el objeto reciba datos externos y ofrezca salidas, permitiéndole así funcionar como parte de un sistema más grande y, a su vez, actuar como un subsistema autónomo.

```

C#
interface ISampleInterface
{
    void SampleMethod();
}
class ImplementationClass : ISampleInterface
{
    // Explicit interface member implementation:
    void ISampleInterface.SampleMethod()
    {
        // Method implementation.
    }
}
static void Main()
{
    // Declare an interface instance.
    ISampleInterface obj = new ImplementationClass();
    // Call the member.
    obj.SampleMethod();
}

```

Método constructor. Un método constructor se utiliza para inicializar una nueva instancia de una clase, lo que efectivamente resulta en la creación de un objeto. Por convención, los constructores suelen llevar el mismo nombre que la clase a la que pertenecen o utilizar una palabra reservada específica, como constructor en algunos lenguajes de programación. Los parámetros de un constructor, si los hay, generalmente representan los valores iniciales que se asignarán a los atributos del nuevo objeto durante su creación.

```

C#
public class Person
{
    private string last;
    private string first;

    public Person(string lastName, string firstName)
    {
        last = lastName;
        first = firstName;
    }
    // Remaining implementation of Person class.
}

```

Método de implementación. Un método de implementación es aquel que se asemeja a una función procedural estándar en programación. Como su nombre lo indica, este tipo de método se encarga de ejecutar la funcionalidad específica de un objeto. Son los métodos que un programador define para realizar las operaciones principales y manipular los estados internos del objeto. Estos métodos suelen modificar los atributos del objeto o realizar operaciones que producen un resultado. En algunos casos, pueden cambiar el estado interno del objeto de una manera que no produce un resultado inmediato visible externamente.

```
class TestMotorcycle : Motorcycle
{
    public override double GetTopSpeed()
    {
        return 108.4;
    }
    static void Main()
    {
        TestMotorcycle moto = new TestMotorcycle();

        moto.StartEngine();
        moto.AddGas(15);
        moto.Drive(5, 20);
        double speed = moto.GetTopSpeed();
        Console.WriteLine("My top speed is {0}", speed);
    }
}
```

4. Matemáticas para Videojuegos

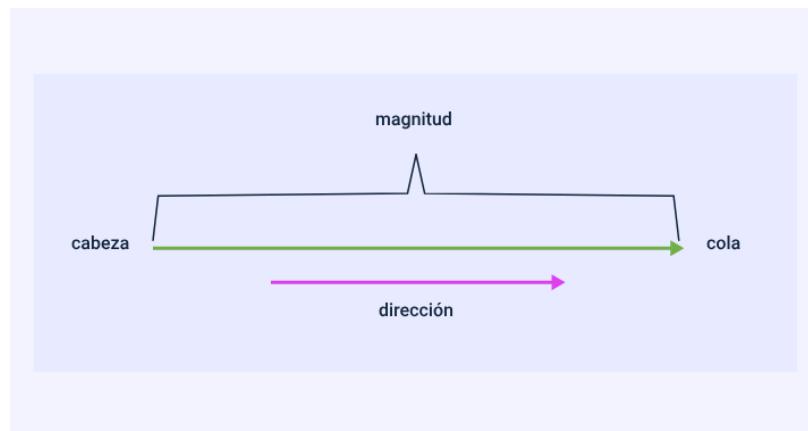
En el mundo de los videojuegos, las matemáticas son fundamentales para el diseño y la programación de cualquier juego. Se utilizan para crear las físicas que rigen los movimientos y las interacciones dentro del juego, calcular trayectorias, gestionar la inteligencia artificial de los personajes no jugables, y mucho más.

Por ejemplo, en los videojuegos 3D, el álgebra lineal es esencial para manejar las transformaciones geométricas y la proyección de escenas en 3D a la pantalla 2D. Se utilizan matrices y vectores para rotar, escalar y mover objetos en el espacio del juego.

4.1. Vectores

Un vector es un objeto matemático que posee magnitud y dirección. Geométricamente, podemos representar un vector como un segmento de línea con dirección, cuya longitud corresponde a la magnitud del vector, y está marcado con una flecha en un extremo que indica su dirección. La dirección del vector se define desde su punto de origen, o cola, hasta su punto final, o cabeza; tal como se ilustra en la figura a continuación:

Figura 7. Vector



Dos vectores son iguales si poseen la misma magnitud y dirección. Esto implica que, si trasladamos un vector a una posición diferente sin cambiar su dirección (es decir, sin rotarlo ni modificarlo de ninguna otra manera), el vector trasladado es considerado idéntico al vector original.

En física, los ejemplos habituales de vectores son aquellos que representan fuerzas y velocidades. La fuerza y la velocidad tienen direcciones definidas: la magnitud del vector fuerza señala la intensidad de dicha fuerza, mientras que la magnitud del vector velocidad indica la rapidez con la que un objeto se desplaza y en qué dirección lo hace.

Los vectores, representados gráficamente, poseen las siguientes características:

- **Dirección.** Definida como la recta sobre la cual se traza el vector, continuada infinitamente en el espacio.
- **Módulo o amplitud.** La longitud gráfica que equivale, dentro de un plano, a la magnitud del vector expresada numéricamente.
- **Sentido.** Representado por la punta de la flecha que gráficamente representa al vector, indica el lugar geométrico hacia el cual se dirige el vector.
- **Punto de aplicación.** Correspondiente al lugar o punto geométrico en donde inicia el vector gráficamente.
- **Nombre.** Representado mediante una letra que acompaña al vector gráficamente representado, y que coincide con la magnitud que expresa o con la suma de los puntos de inicio y fin de su valor.

4.2. Matrices

Una matriz es una disposición rectangular compuesta por números, variables, símbolos o expresiones, organizados en filas y columnas. Esta estructura permite efectuar diversas operaciones matemáticas como la suma, resta, multiplicación y transposición. Cada unidad de la matriz se conoce como “elemento”. Las secuencias de elementos en horizontal constituyen las filas y en vertical, las columnas. El tamaño u orden de una matriz se especifica por el número de filas y columnas que posee, denotándose como ' $m \times n$ ', donde ' m ' representa el número de filas y ' n ' el de columnas. Por ejemplo, una matriz B de 3 filas y 4 columnas se escribe como $B_{3 \times 4}$ y se denomina matriz de tres por cuatro.

Una forma sencilla de identificar qué es una clase, qué es un atributo y qué es un método es a través de su representación lingüística: generalmente, una clase se asocia con un sustantivo, los atributos con cualidades y los métodos con verbos.

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Notación de matrices

Si una matriz tiene m filas y n columnas, entonces contiene $m \times n$ elementos. Las matrices suelen representarse con letras mayúsculas, como ' A ', y sus elementos con la misma letra en minúscula, seguida de dos subíndices que indican su posición específica en términos de fila y columna. Por ejemplo, a_{ij} representa el elemento de la matriz A que se encuentra en la i -ésima fila y la j -ésima columna, como se presenta a continuación:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1j} \\ a_{21} & a_{22} & \cdots & a_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} \end{bmatrix}$$

4.3. Operaciones sobre matrices

Las operaciones sobre matrices son un pilar fundamental en el álgebra lineal y tienen numerosas aplicaciones en diferentes campos como la física, la ingeniería, las ciencias de la computación, entre otros. A continuación, se describirá algunas de las operaciones básicas que se pueden realizar con matrices:

- **Adición de matrices.** La suma de matrices sólo es posible si el número de filas y columnas de ambas matrices es el mismo. Se suma cada elemento con su correspondiente en la otra matriz.

Figura 8. Ejemplo suma de matrices

$$\begin{aligned}
 & \begin{bmatrix} 2 & -1 \\ 0 & 5 \end{bmatrix} + \begin{bmatrix} 0 & 2 \\ 1 & -2 \end{bmatrix} \\
 = & \begin{bmatrix} 2 + 0 & -1 + 2 \\ 0 + 1 & 5 + (-2) \end{bmatrix} \\
 = & \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}
 \end{aligned}$$

- **Resta de matrices.** La resta de matrices también es posible, solo si el número de filas y columnas de ambas matrices es el mismo. Mientras restamos 2 matrices, restamos los elementos correspondientes.

Figura 9. Ejemplo resta de matrices

$$\begin{aligned}
 & \begin{bmatrix} 2 & -1 \\ 0 & 5 \end{bmatrix} - \begin{bmatrix} 0 & 2 \\ 1 & -2 \end{bmatrix} \\
 = & \begin{bmatrix} 2 - 0 & -1 - 2 \\ 0 - 1 & 5 - (-2) \end{bmatrix} \\
 = & \begin{bmatrix} 2 & -3 \\ -1 & 7 \end{bmatrix}
 \end{aligned}$$

- **Multiplicación escalar.** El producto de una matriz A con cualquier número 'c' se obtiene multiplicando cada entrada de la matriz A por c, se llama multiplicación escalar, es decir, $(cA)_{ij}=c(A_{ij})$.
- **Propiedades de la multiplicación escalar en matrices.** Las diferentes propiedades de las matrices para la multiplicación escalar de cualquier escalar K y l, con las matrices A y B se dan como:

$$K(A + B) = KA + KB$$

$$(K + I)A = KA + IA$$

$$(KI)A = K(IA) = I(KA)$$

$$(-K)A = -(KA) = K(-A)$$

$$1 \cdot A = A$$

$$(-1)A = -A$$

- **Multiplicación de matrices.** La multiplicación de matrices se define solo si el número de columnas de la primera matriz y de filas de la segunda matriz son iguales. Para entender cómo se multiplican las matrices, consideremos primero un vector fila $R = [r_1 \ r_2 \ \dots \ r_n]$ y un vector de columna:

$$C = \begin{bmatrix} | & | & | \\ C_1 \\ C_2 \\ \vdots \\ C_n \\ | & | & | \end{bmatrix}$$

Entonces el producto de R y C se puede definir como:

$$\begin{aligned} RC &= [r_1 \ r_2 \ \dots \ r_n] \begin{bmatrix} | & | & | & | \\ C_1 & C_2 & \vdots & C_n \end{bmatrix} | & | \\ &= [r_1 \ C_1 + r_2 C_2 + \dots + r_n C_n] \end{aligned}$$

Por ejemplo,

$$[1 \ 2 \ 3] \begin{bmatrix} 2 \\ -1 \\ 4 \end{bmatrix} = [7]$$

5. Interfaz y flujo de integración de Unity 3D

A continuación, se describen los pasos requeridos para la instalación de Unity 3D e información general de la forma de operar y/o utilizar algunas de sus funcionalidades básicas.

5.1. Instalación de Unity 3D

En este caso se utilizará la versión gratuita que puede descargar desde de <https://unity.com/es>

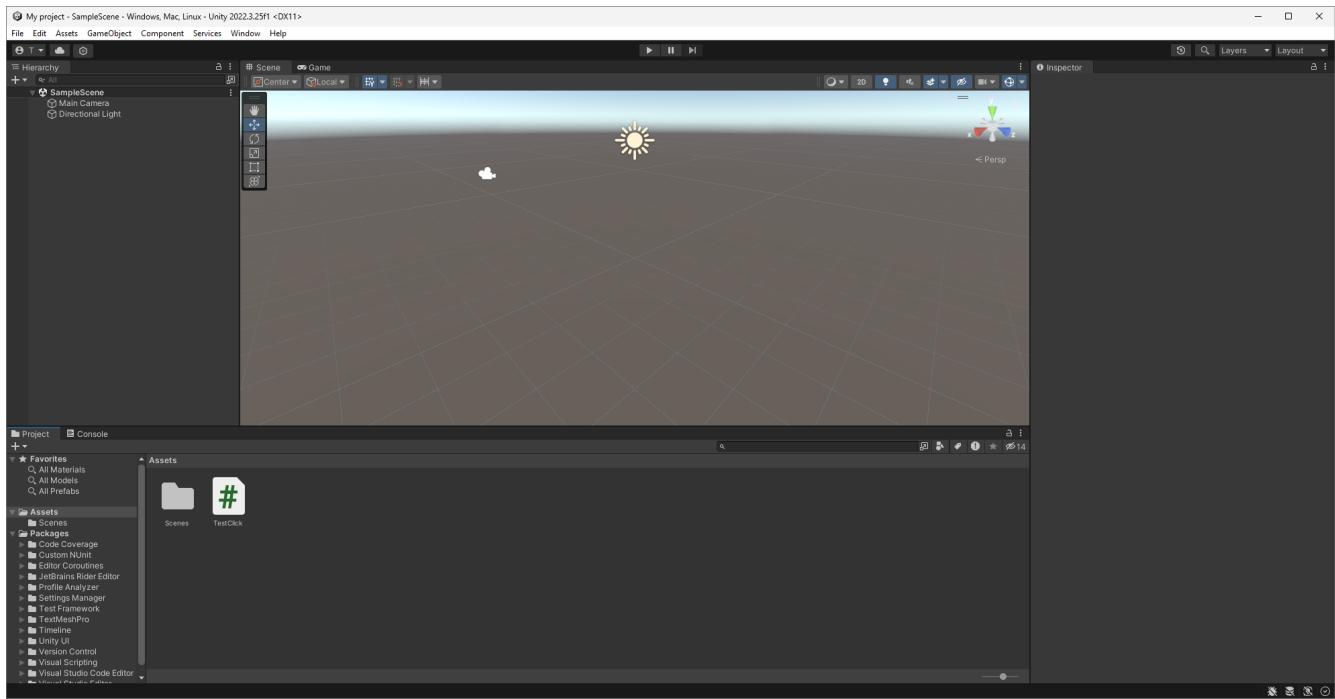
Unity instalación. En la carpeta Anexos encontrará un documento denominado “**Unity instalación**” en él puede consultar el detalle con los pasos de instalación, creación de Unity ID y el tipo de microjuego inicial.

5.2. Navegación en Unity 3D

Después de finalizada la creación del proyecto, se muestra la interfaz de Unity como se ve en la siguiente imagen.

En dicha pantalla se puede ver el entorno 3D y las pestañas, desde allí se puede configurar la interfaz según la necesidad requerida.

Figura 10. Interfaz Unity



Nota. Pantalla principal proyecto Unity.

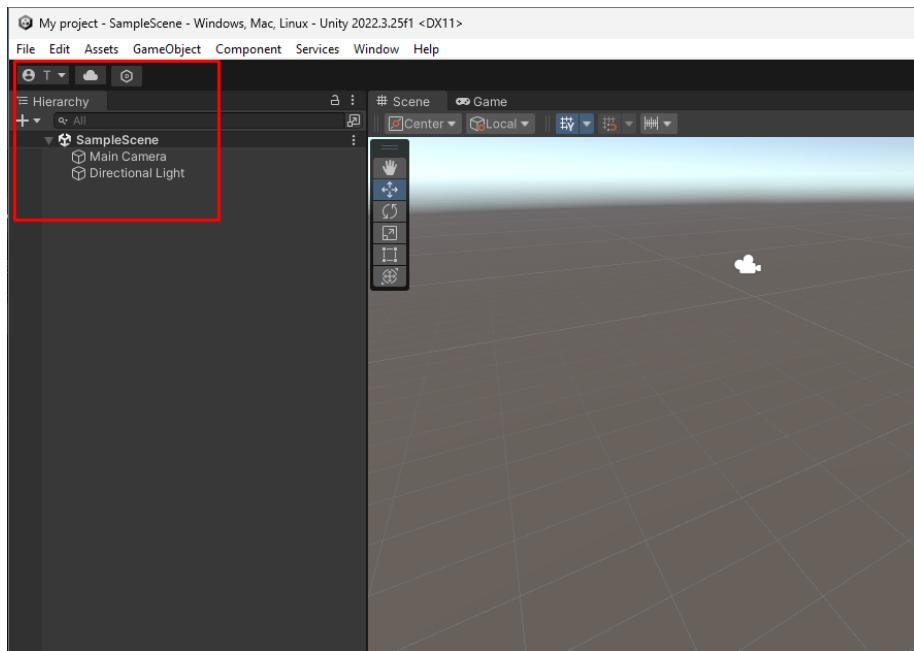
Además, puede configurar la interfaz de Unity 3D según la necesidad requerida a través de las siguientes opciones:

- “**Game**”. En esta pestaña se ve el juego funcionando, para probarlo y verlo en ejecución, se debe dar clic en el botón “play”. Cuando este botón está resaltado en azul, indica que el juego está activo, tal como se presenta en la siguiente figura.



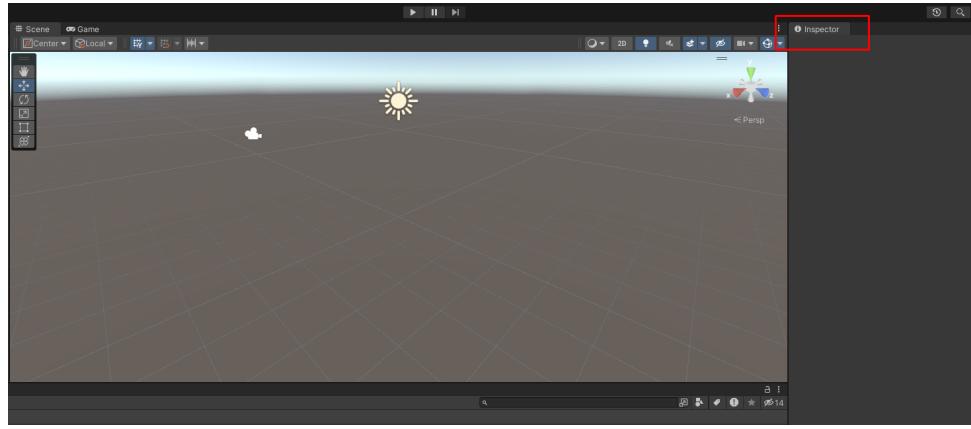
Nota. Ejecución juego.

- “**Scene**”. Pestaña que permite configurar el escenario del videojuego, donde van a ir ubicados los elementos del mismo. Dichos elementos estarán listados en la pestaña de la izquierda llamada “hierarchy”. Unity por defecto ubica dos elementos, cámara (“Main Camera”) e iluminación (“Directional Light”) como se muestra en la siguiente figura.

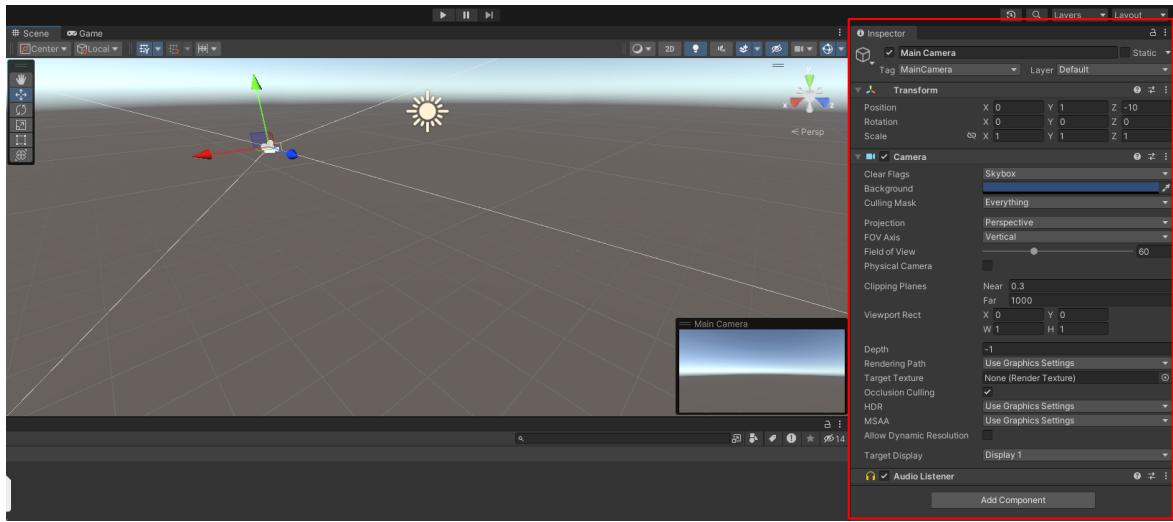


Nota. Opciones pestaña Scene.

- **Inspector**. Al lado derecho se muestra la pestaña Inspector, en la cual se pueden modificar los atributos y propiedades de cada elemento, como se presenta en las siguientes figuras:

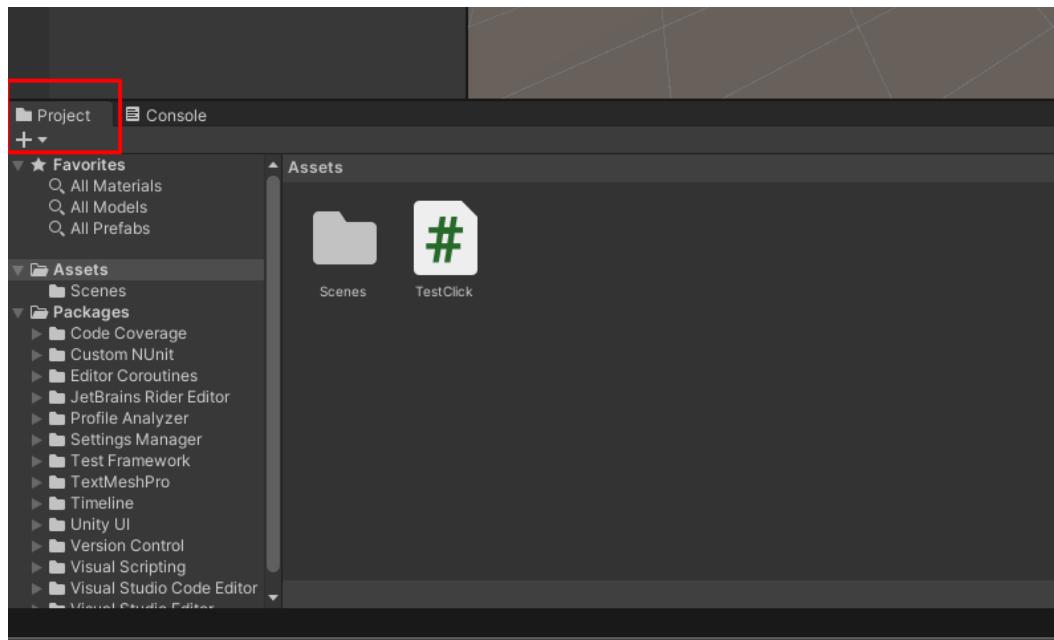


Nota. Pestaña Inspector.



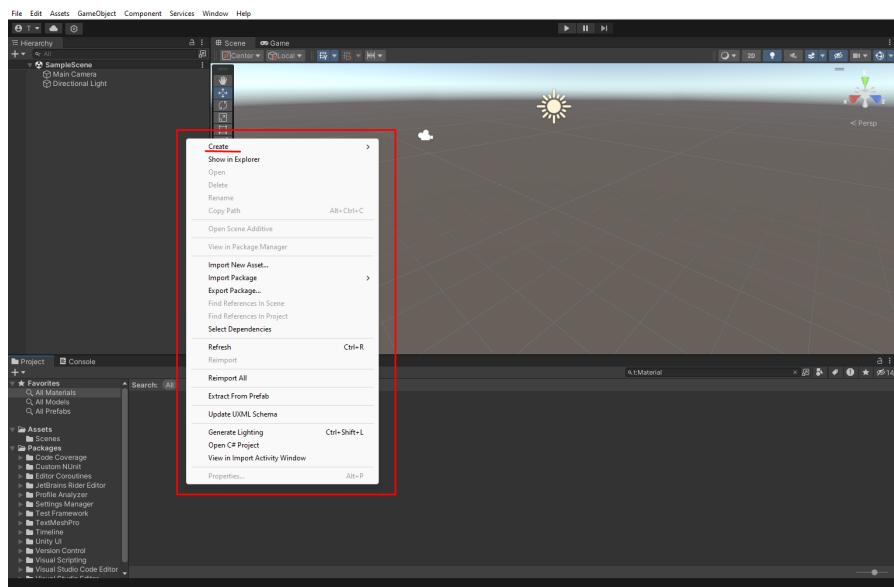
Nota. Opciones Inspector.

- “**Project**”. En la parte inferior izquierda se encuentra la pestaña “**Project**”, que es de suma importancia, pues allí se agregarán archivos externos a Unity como sonidos, texturas, imágenes, entre otros; los cuales pueden anexarse simplemente arrastrándolos, dando clic en la pestaña “**assets**” - crear como se presenta en la siguiente figura.



Nota. Opciones pestaña Project.

O directamente en la ventana “Project”, dando clic derecho en “Create” como se presenta en la siguiente figura.



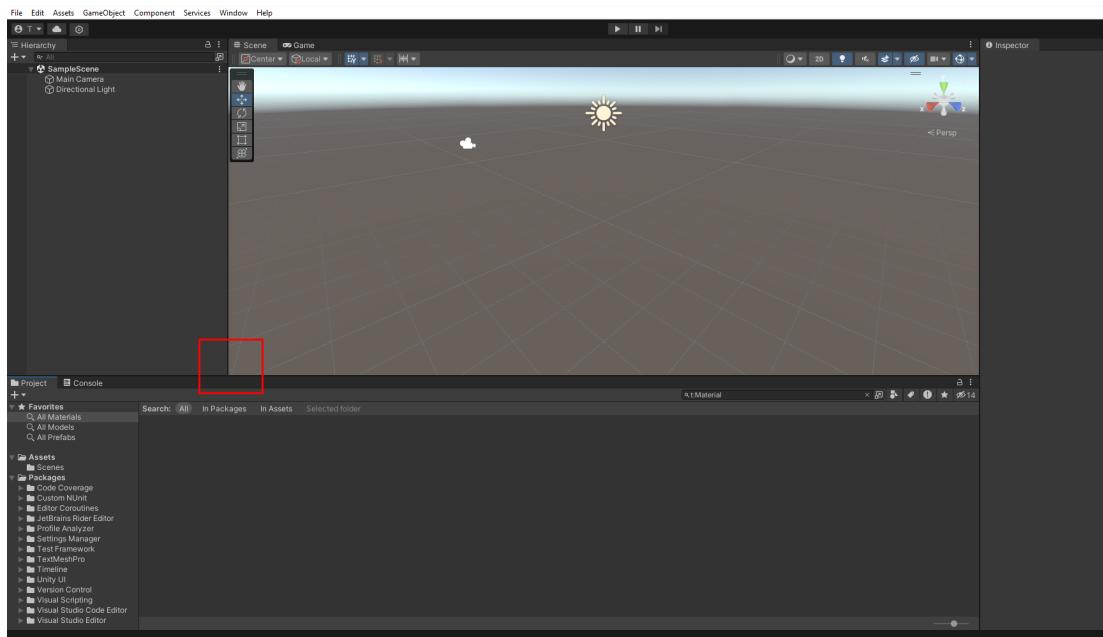
5.3. Escenario y ventanas de Unity 3D

Se puede modificar la vista y tamaño de la escena por medio de las siguientes alternativas:

- **Tamaño escena.** La ventana y vista de la escena se puede ampliar arrastrándola a partir de sus bordes, como se identifica en la siguiente figura.

En el escenario hay 3 ejes, alto, ancho y profundidad, que se indican en la parte superior derecha con las letras X, Y, Z.

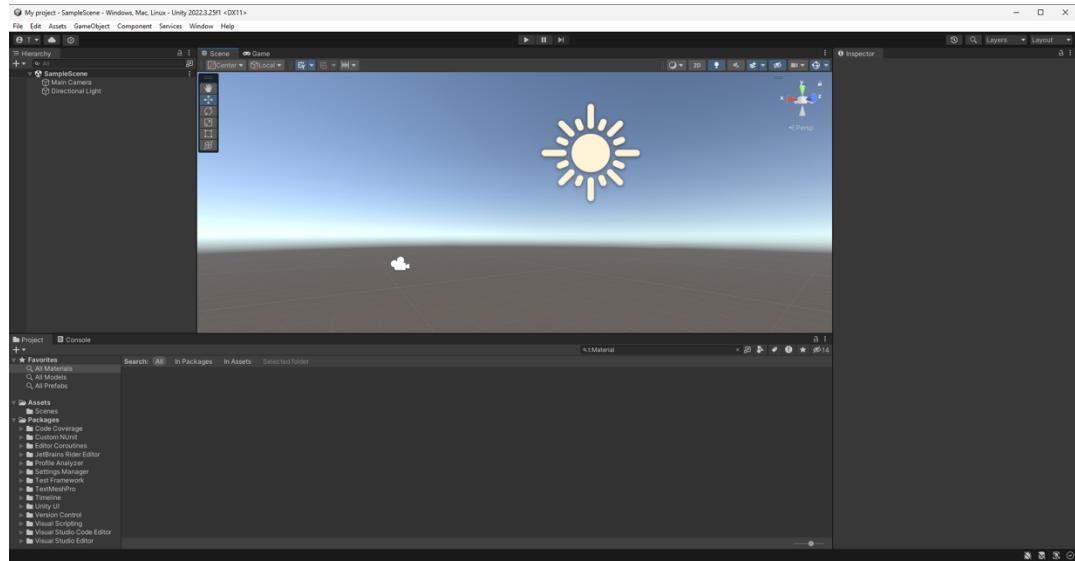
Al dar clic derecho con el “mouse” se puede rotar la vista, además de que si se mantiene oprimido y se presionan las teclas WASD se podrá mover por el escenario como si se tratase de un videojuego en primera persona.



Nota. Identificación bordes cambio de tamaño escena.

- **“Zoom”.** Al girar la rueda del “mouse” se puede hacer “zoom” o acercamiento, al presionarla y mantenerla oprimida, el cursor toma forma

de mano, lo que permite mover la vista del escenario, como se presenta en la siguiente figura.



Nota. Movimiento escenario - “zoom”

- **Vista.** Si se mantiene la tecla alt + clic izquierdo se puede girar la vista en torno al punto central de la misma, como se presenta en la siguiente figura.



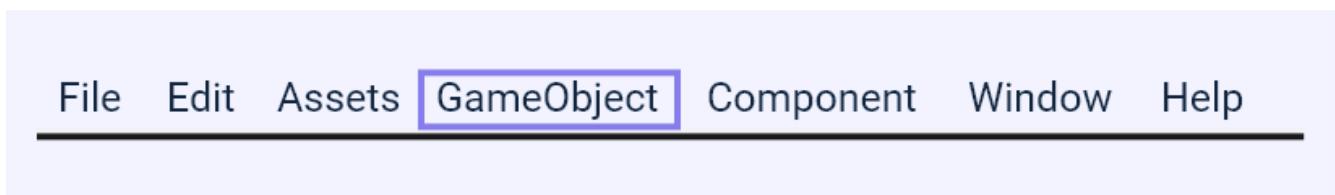
Nota. Movimiento escenario - vista.

6. Unity 3D importar elementos 3D - “assets”

A continuación, se describe el funcionamiento de algunas de las opciones de Unity 3D a través de ejemplos con su correspondiente explicación.

Para este ejemplo, se creará un cubo y se posicionará en el escenario. La ruta es la siguiente, en el menú principal situado en la parte superior se selecciona la opción “GameObject”, como se presenta en la siguiente figura.

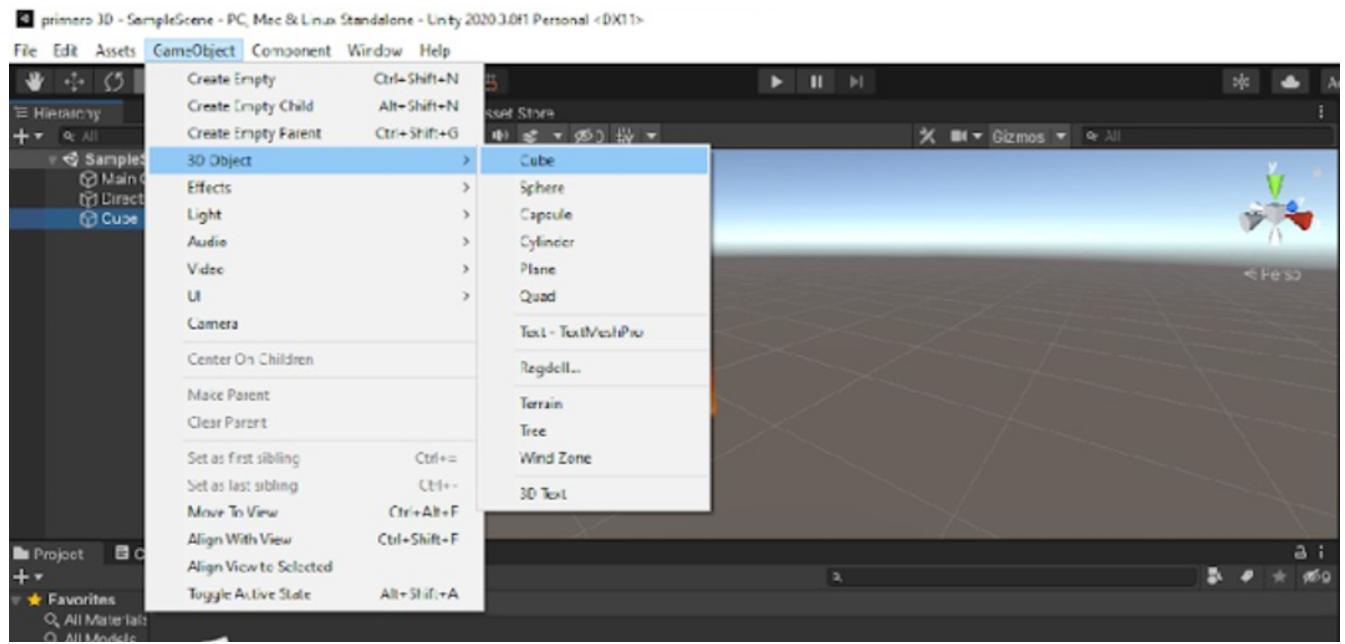
Figura 11. Interfaz Unity



Nota. Opción menú “GameObject”

Después se selecciona la opción Cube del menú 3D “Object”, como se presenta en la siguiente figura.

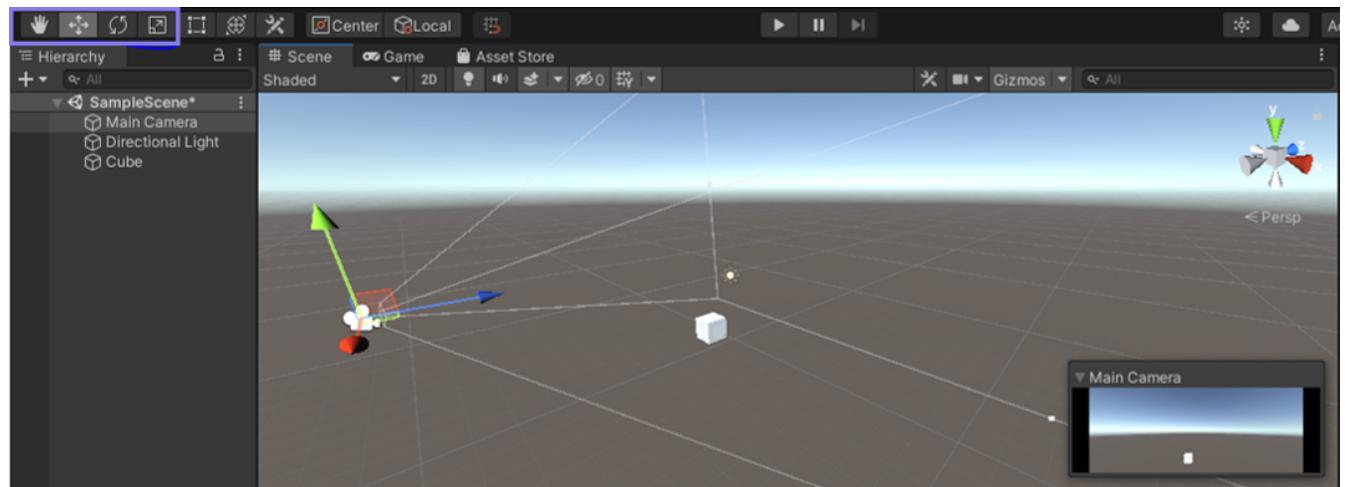
Figura 12. Crear cubo



Nota. Opción menú crear cubo.

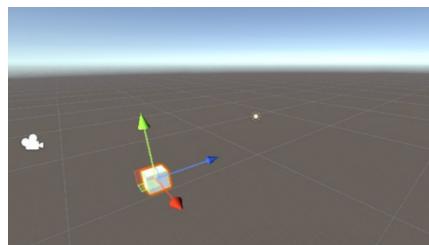
Una vez hecho esto, en el escenario se incluirá un cubo. Para mover los objetos en el escenario, en la parte superior izquierda se encuentran 3 iconos: mover, rotar y escalar, como se resalta en la siguiente figura.

Figura 13. Mover, rotar, escalar



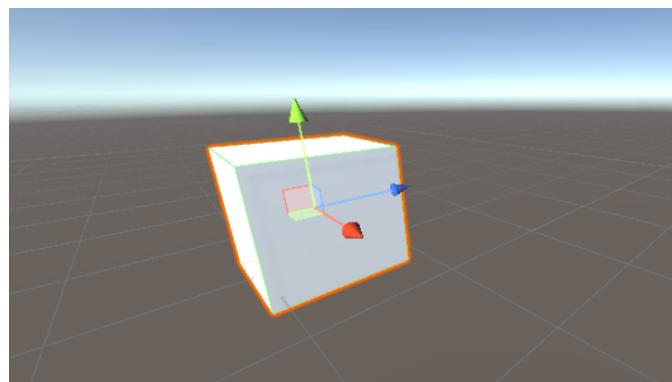
Nota. Opciones iconos mover, rotar, escalar.

- **Botón mover.** Cuando se selecciona un objeto en el escenario, habiendo pulsado el botón mover, en el objeto se mostrarán 3 flechas que representan los 3 ejes mencionados anteriormente. Si se da clic sostenido en dichas flechas y se mueve el “mouse” se traslada en línea recta el objeto en el escenario en el eje de coordenadas seleccionado, como se representa en la siguiente figura.



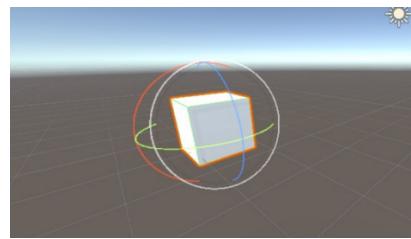
Nota. Botón mover.

- **Botón mover - dos ejes.** Si se quiere mover el objeto en dos ejes al mismo tiempo se da clic en los recuadros internos del objeto, el cuadro selecciona esos dos ejes que componen el cuadro pequeño interno, como se presenta en la siguiente figura.



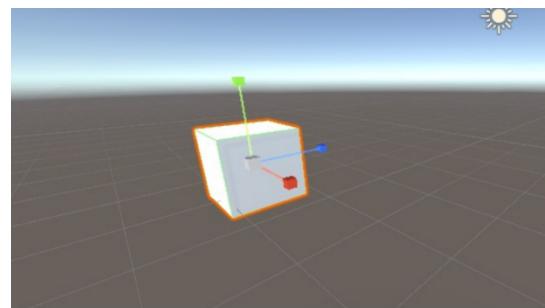
Nota. Botón mover - dos ejes.

- **Botón rotar o girar.** Cuando se activa el botón rotar y se selecciona un objeto, este se ve envuelto por líneas circundantes en sus respectivos ejes y una circundante global, como puede evidenciarse en la siguiente figura. Al dar clic en alguna de las líneas circundantes y mover, se puede ver que el objeto hace rotación según el eje que se haya elegido. Para rotar sin depender de ningún eje, es decir, libremente, se da clic dentro de las esferas circundantes, pero sin seleccionar ninguna. Si se selecciona la línea global (en este caso la blanca) se rotaría dependiendo de la cámara o vista.



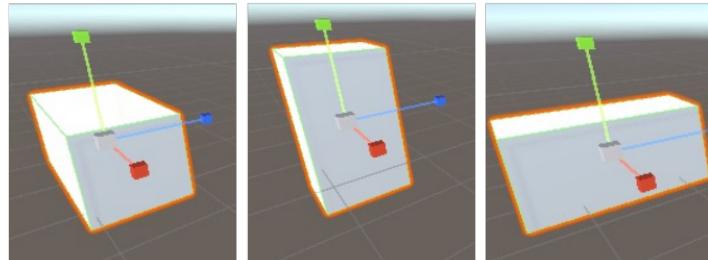
Nota. Botón rotar.

- **Botón escala.** Se usa para hacer más grande o pequeño el objeto. Al dar clic sobre él y seleccionar un objeto, se muestran nuevamente 3 líneas que representan los ejes, esta vez terminadas en cubo como se visualiza, a continuación:



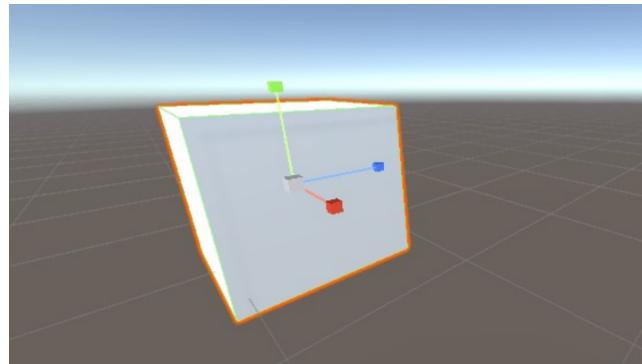
Nota. Botón escala.

- **Ejemplos botón escala.** Dependiendo del eje seleccionado, y al estirarse al dar clic, el objeto escalará siguiendo esas coordenadas. A continuación, se presentar algunos ejemplos:



Nota. Ejemplos botón escalar.

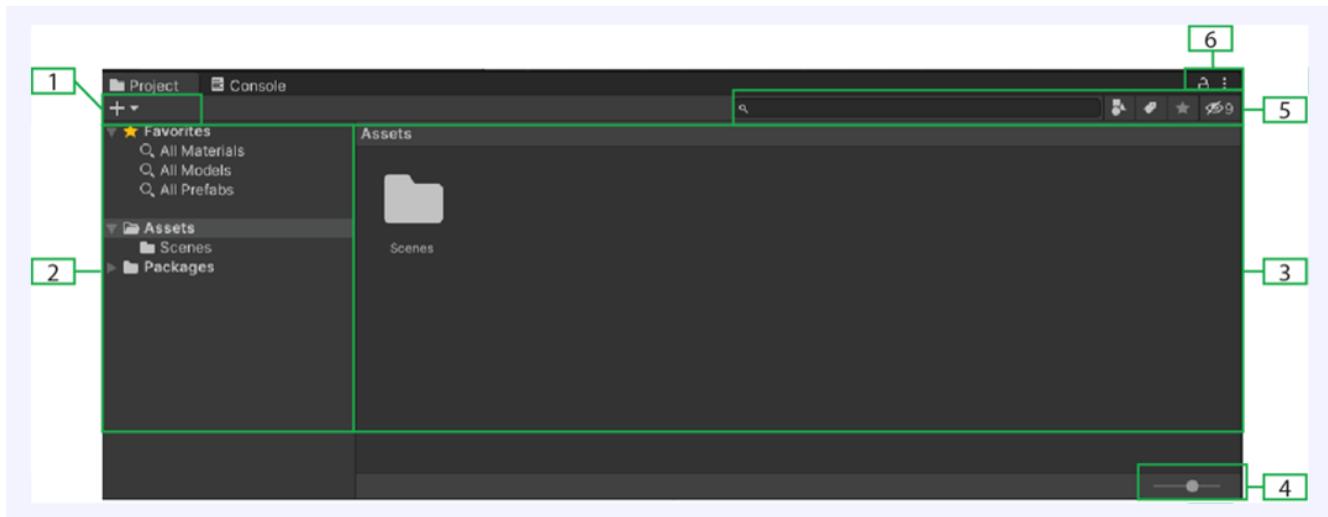
- **Botón escala.** Por otra parte, si se quiere escalar uniformemente, se selecciona el cubo interior que aparece en el objeto, como se presenta a continuación:



Nota. Ejemplos botón escalar.

- **Ventana “Project”.** Esta ventana es la que contendrá los elementos del proyecto, su carpeta estará asociada a una carpeta en el computador. A continuación, se encuentra una figura en la que se enumeran algunos de sus componentes:

Figura 14. Ventana “Project”



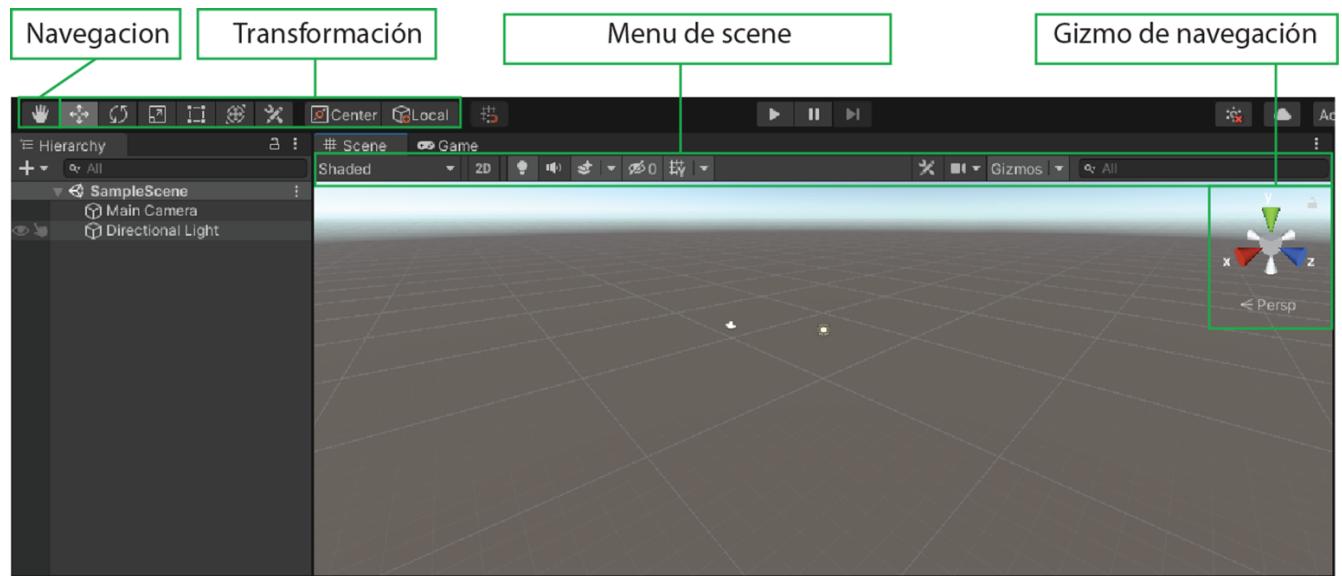
Elementos ventana “Project”

1. Permite crear una variedad de elementos para el proyecto.
2. En la “assets” se pueden agregar elementos y organizarlos por carpetas. También se puede usar la sección de favoritos en la cual estarán los elementos más utilizados.
3. Se pueden ver los elementos disponibles, las carpetas, sonidos, modelos, entre otros.
4. Este “Slider” permite aumentar o reducir el tamaño de los “assets” que se tienen en la sección del ítem 3.
5. Permite buscar elementos por su nombre.
6. El candado bloquea la ventana, así no será posible su modificación. Con el símbolo de los 3 puntos se pueden agregar columnas u otro tipo de ventanas.

“Ventana Scene View”

En esta ventana se crean los escenarios y se posicionan los elementos que se requieren para su construcción, tales como cámaras, luces, entre otros. A continuación, se enumeran algunos de sus componentes:

Figura 15. “Ventana Scene View”

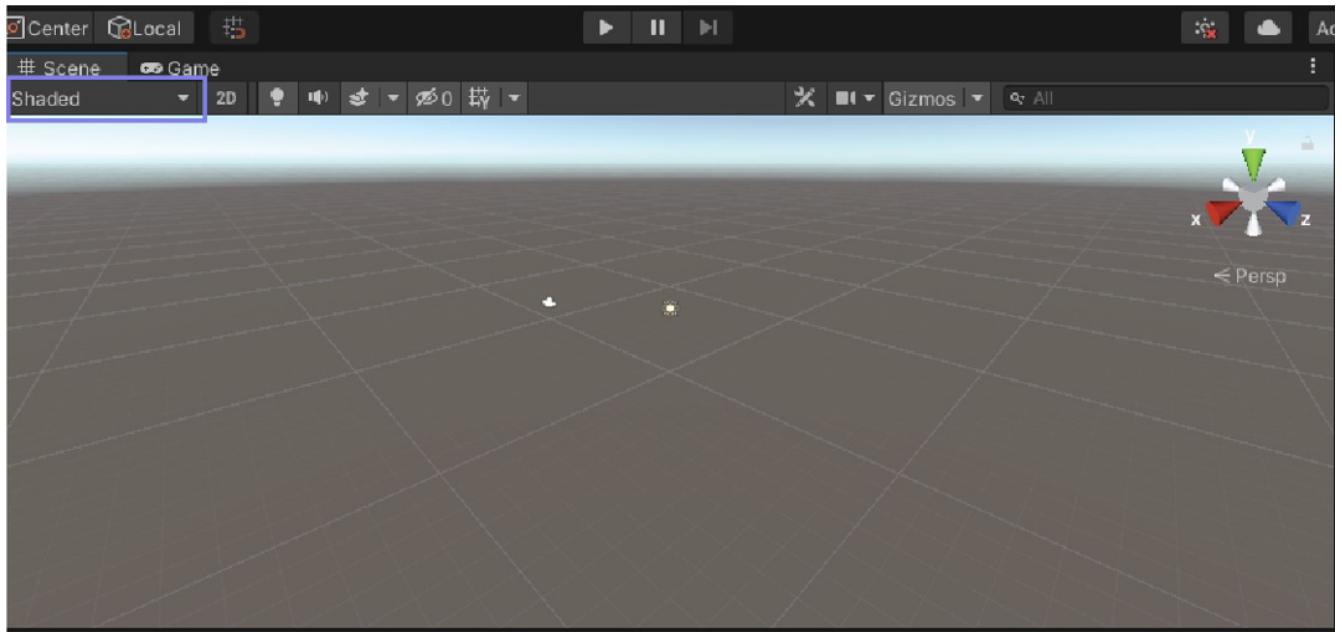


Nota. Elemento “Ventana Scene View”

Los iconos de la sección navegación permiten desplazarse por la escena, y los de la sección transformación permiten rotar, mover y escalar los objetos en la escena.

La pestaña “Shaded” permite modificar la manera en que se visualiza el escenario, se puede cambiar a una vista alambrada para observar los polígonos que componen los elementos, como se presenta a continuación:

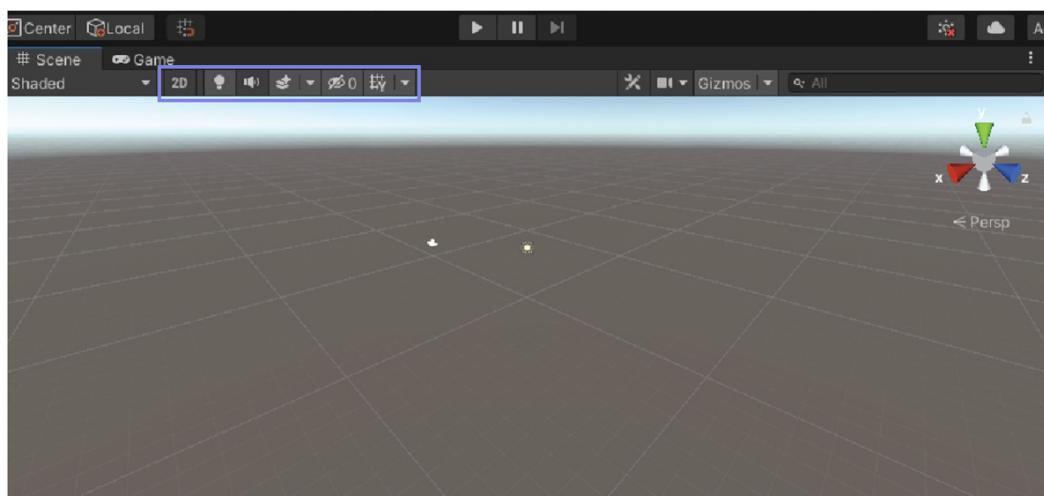
Figura 16. “Shaded”



Nota. Elementos ventana Shaded.

Con los botones señalados en la siguiente figura, se pueden activar o desactivar de la escena las luces, los sonidos y la ambientación.

Figura 17. Botones de activación

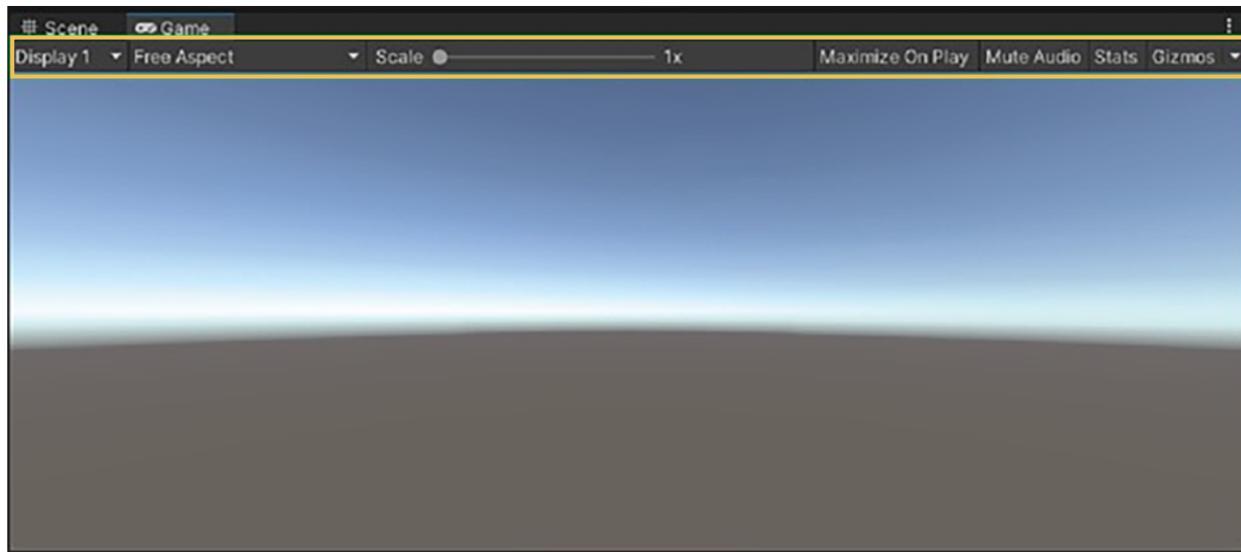


Nota. Elementos ventana Shaded – botones de activación.

“Ventana Game View”

Aquí se puede visualizar la compilación de los objetos y funciones del videojuego a través de la cámara que los esté observando.

Figura 18. “Ventana Game View”



Nota. Elementos ventana “Game View”

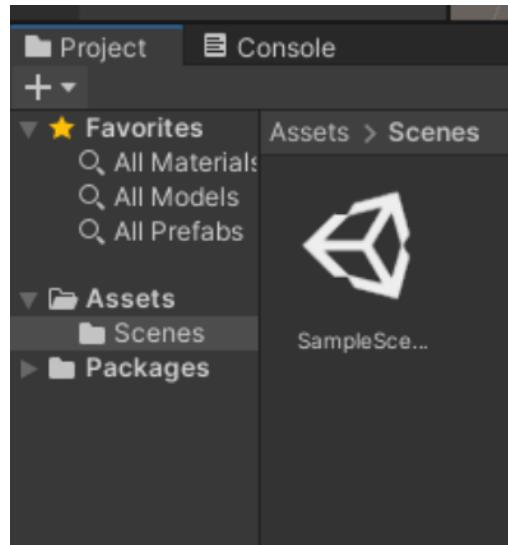
- **“Display”**: opciones de menú para escoger la vista de la cámara de la escena, por defecto está configurado “Display 1”.
- **“Free Aspect”**: valores predefinidos para probar cómo se vería el videojuego en distintos formatos.
- **“Scale”**: permite hacer “zoom” en la ventana de juego para visualizar con mayor detalle el juego o para ver de manera global cómo se ve desde lejos.
- **“Maximize on Play”**: si esta activa, al ejecutar el juego en modo “play” la ventana “game” se maximiza para jugar en pantalla completa.

- “**Mute Audio**”: al activarla se silencia el audio del juego cuando está en modo “play”.
- “**Status**”: activa/inactiva la ventana de estadísticas, la cual muestra información del renderizado gráfico y audio al estar en modo “play”.
- “**Gizmos**”: activa/inactiva la visualización de cierto tipo de “gizmos” o iconos que se muestran en la venta escena.

Tipos de archivos

A través de la ventana “Project” se crean las carpetas para organizar el proyecto como se presenta a continuación:

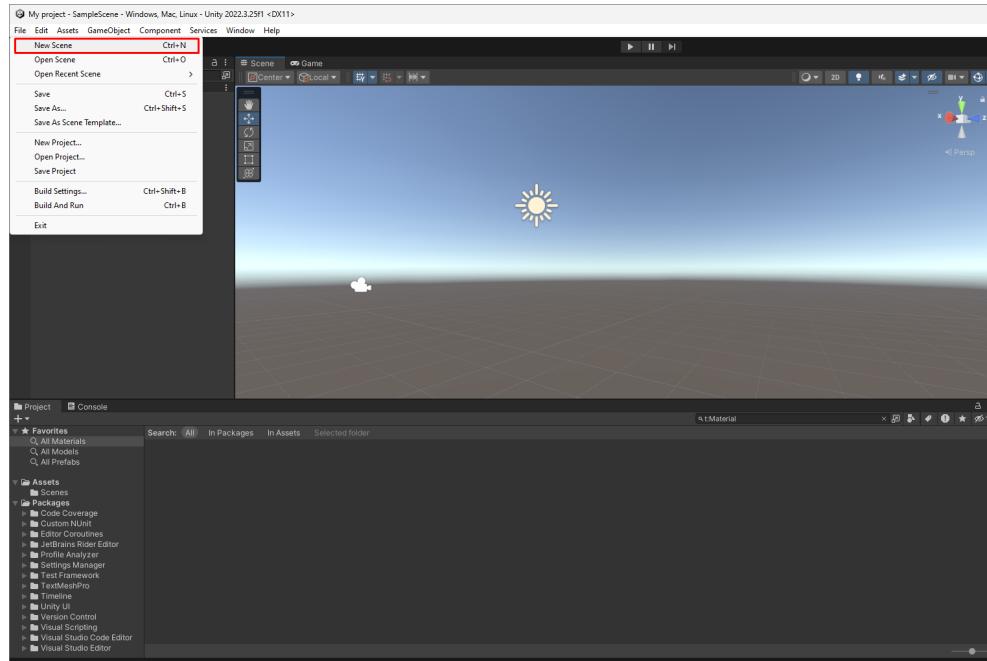
Figura 19. Tipos de archivos



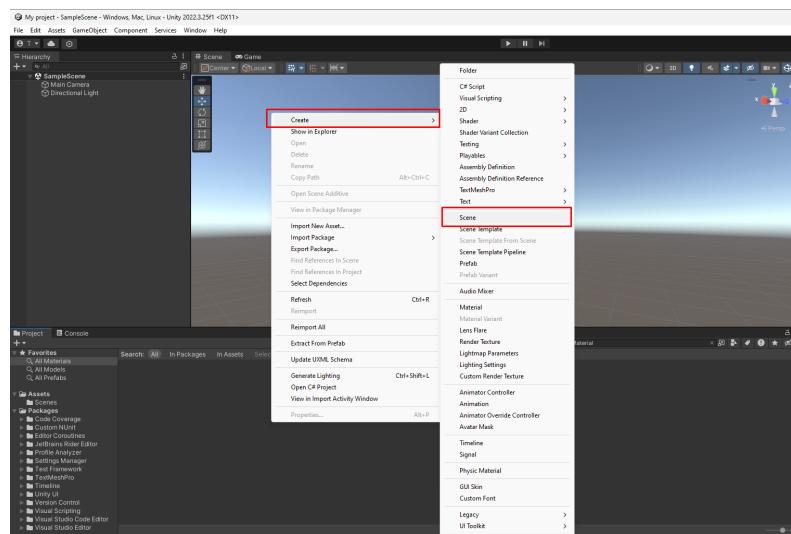
Nota. Elementos ventana tipos de archivos.

- **Archivo tipo escena.** Se crea por defecto al abrir un proyecto nuevo y se ubica en la carpeta “scenes”. Una escena es similar a un nivel del videojuego, esta contendrá elementos tales como cámaras, luces, personajes, acciones y

comportamientos específicos en ella. Por ello, si se necesitan más escenas para crear por ejemplo nuevos niveles, la forma de hacerlo sería en el menú principal, “file” - “new scene” o utilizando el comando control + N. Se presenta en las siguientes figuras:



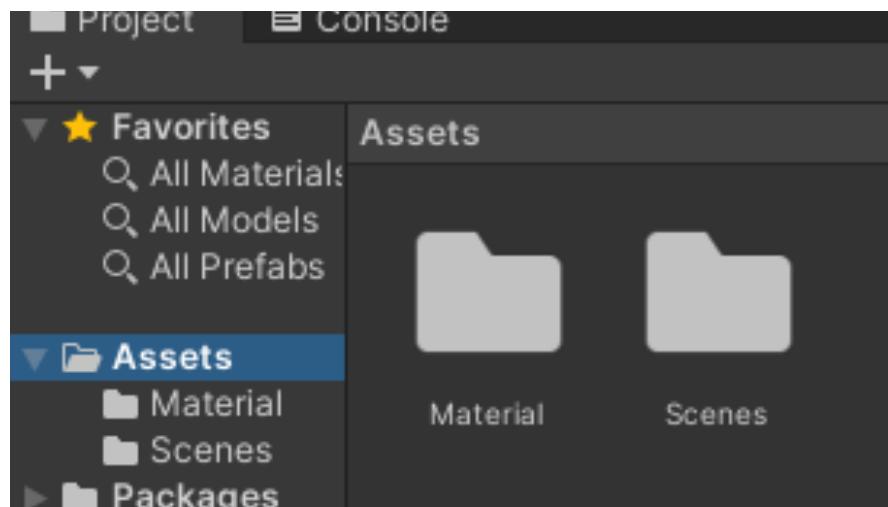
Nota. Elementos ventana “new scene”.



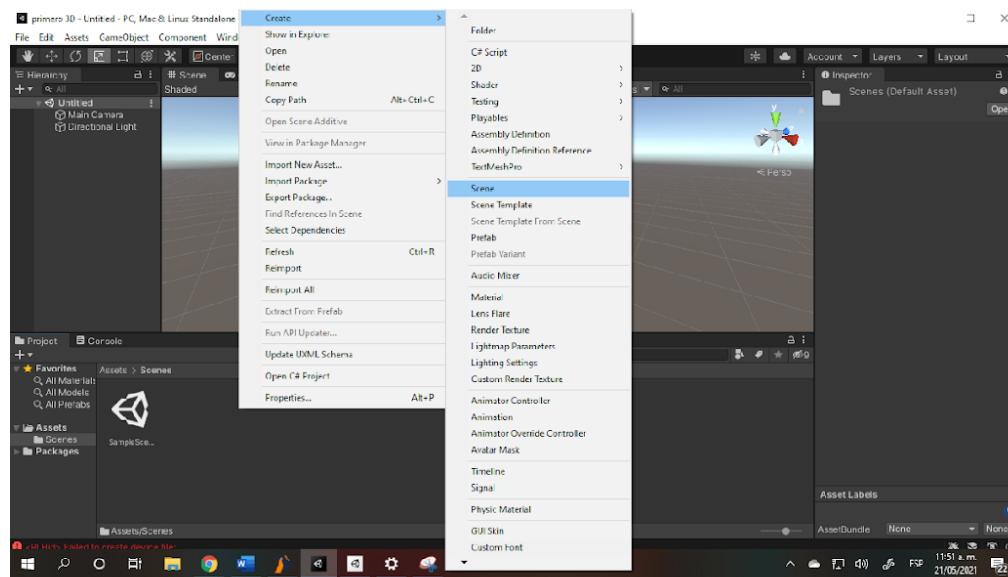
Nota. Elementos ventana “new scene” - clic derecho ventana inferior, “Create” – scene.

Archivo tipo material.

Se crea por defecto al abrir un proyecto nuevo y se ubica en la carpeta scenes. Una escena es similar a un nivel del videojuego, esta contendrá elementos tales como cámaras, luces, personajes, acciones y comportamientos específicos en ella. Por ello, si se necesitan más escenas para crear por ejemplo nuevos niveles, la forma de hacerlo sería en el menú principal, “file” - “new scene” o utilizando el comando control + N. Se presenta en las siguientes figuras:

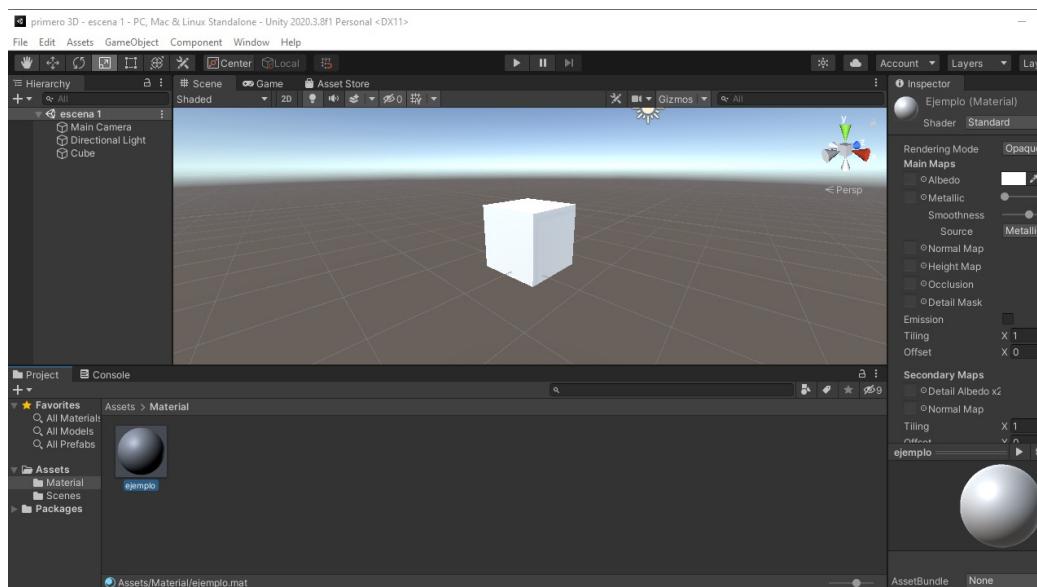


Nota. Elementos ventana “new scene”.



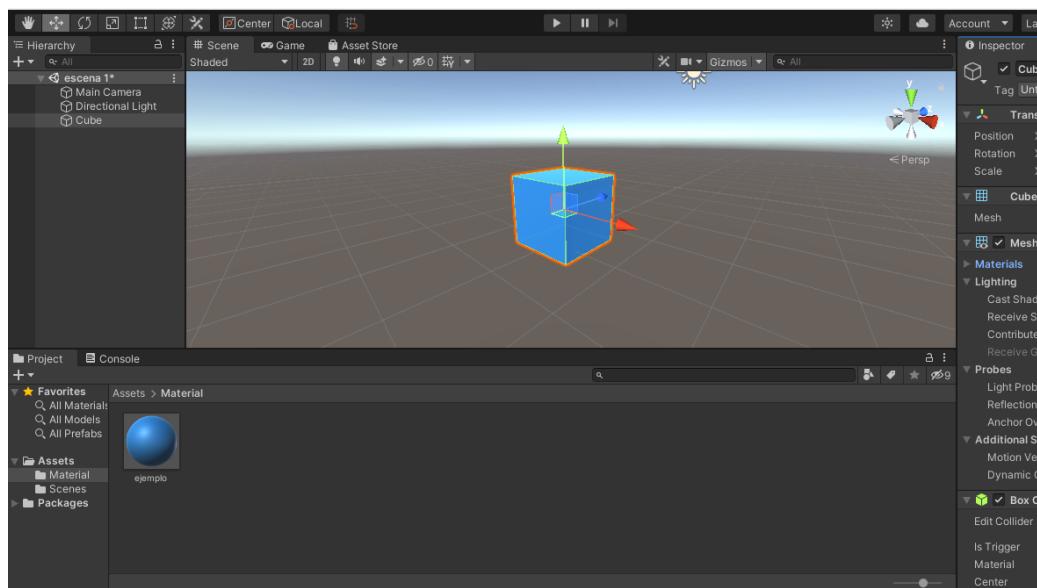
Nota. Elementos ventana “new scene” - clic derecho ventana inferior, “Create” – *scene*.

Al ejecutar el paso descrito previamente aparecerá el material, y a la derecha en la pestaña inspector mostrará sus atributos o especificaciones, como se presenta en la siguiente figura:



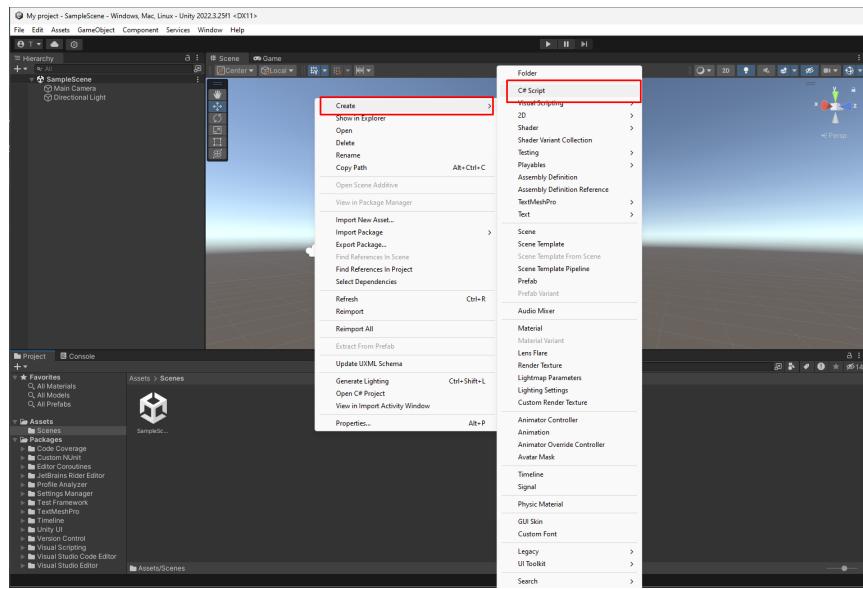
Nota. Elementos ventana *inspector material*.

En este caso se selecciona *Shader – Standard* en la pestaña *Inspector*, se pueden cambiar características como por ejemplo el color una vez teniendo el material, en este caso un color simple sin más se procede a aplicarlo al objeto manteniendo presionado clic en el material y arrastrándolo encima del objeto, al hacerlo, este cambia tal como se presenta a continuación:



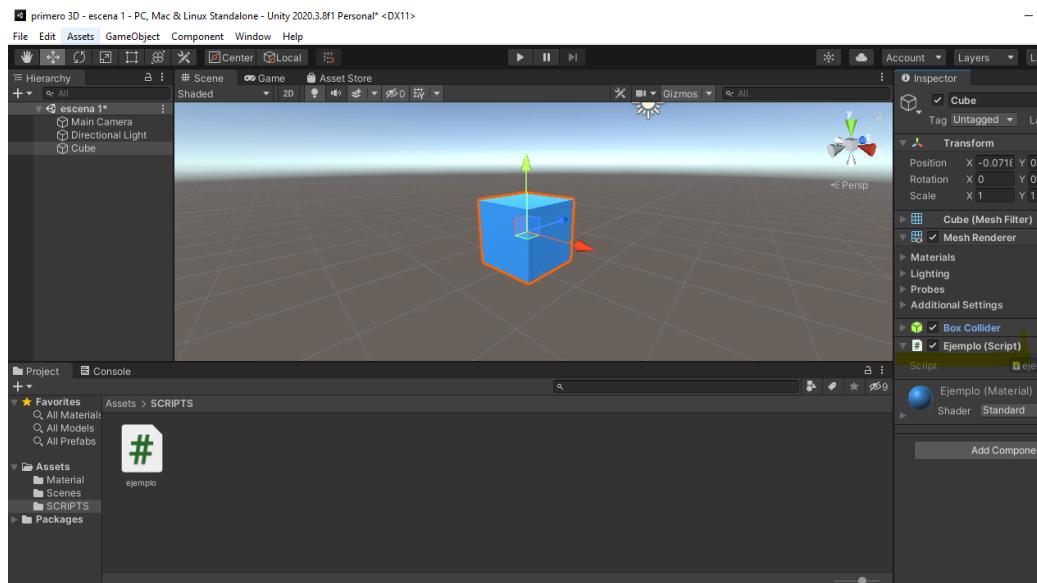
Nota. Ejemplo aplicación material a objeto.

- **Archivo tipo “script”.** Son instrucciones de programación para que el objeto a quien se le asigna realice dichos comportamientos y funciones. La ruta de creación es en la pestaña proyecto – “Create” – C# “script”, como se presenta en la siguiente figura.



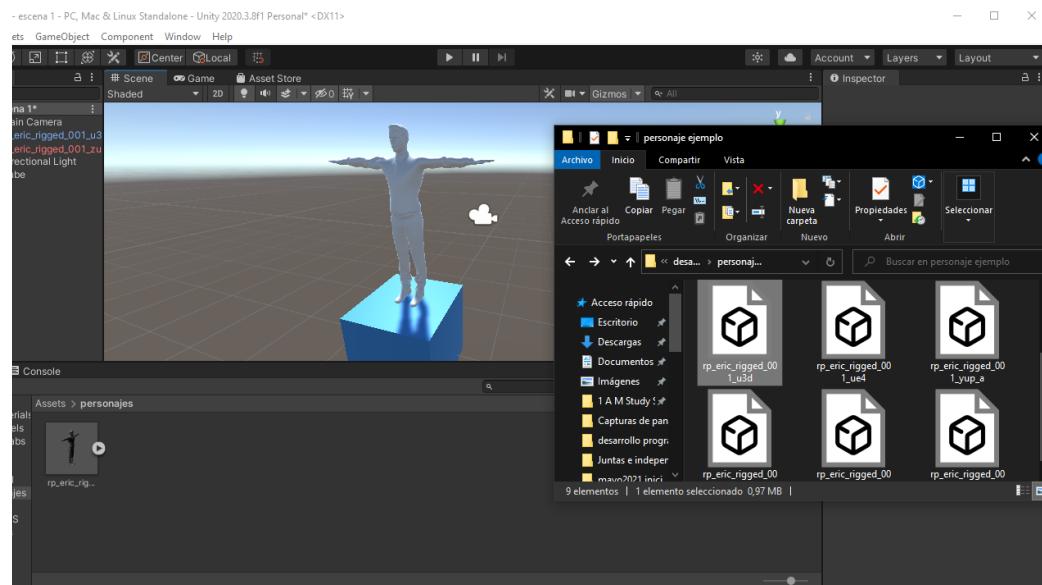
Nota. Crear C# “script”.

La forma de aplicarlo al objeto es similar al material, es decir, se arrastra al objeto para verificar su aplicación y se puede hacer a través de la pestaña *Inspector*, seleccionando previamente el objeto al que se aplicó el “script”, tal como se presenta en la siguiente figura.



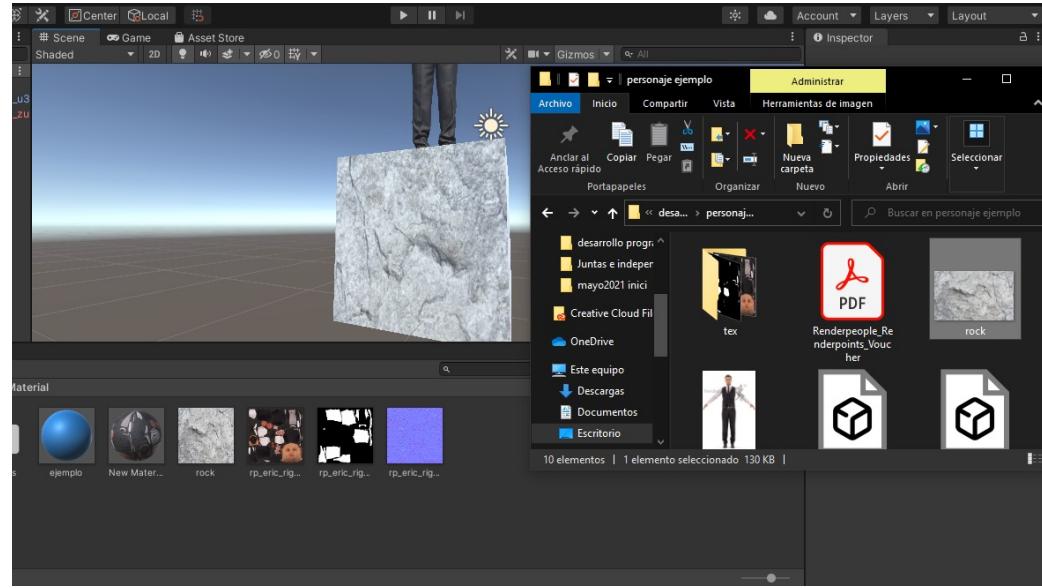
Nota. Verificar aplicación C# “script”.

- **Tipos de archivos externos.** Unity tiene elementos 3D básicos, si se quiere utilizar elementos más desarrollados se deben crear en programas externos, y luego importarlos, en este caso se pueden importar los archivos. fbx simplemente arrastrándolos desde la carpeta del computador hacia la ventana proyecto, en una nueva carpeta creada para los personajes o modelados, como se presenta en la siguiente figura.



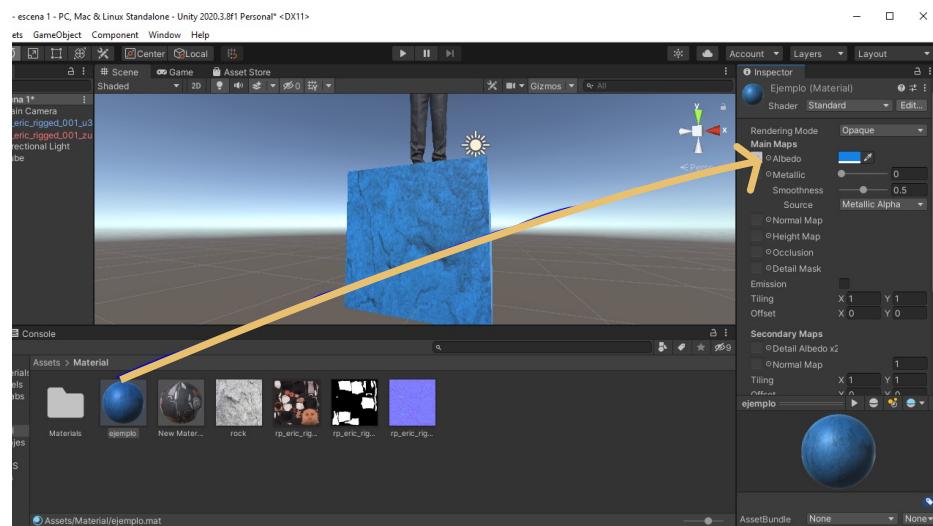
Nota. Visualización elementos externos agregados.

También se puede importar archivos de texturas de la misma manera, es decir, arrastrándolos desde la carpeta del computador al proyecto, con esto se creará un nuevo material. Se presenta en la siguiente figura:



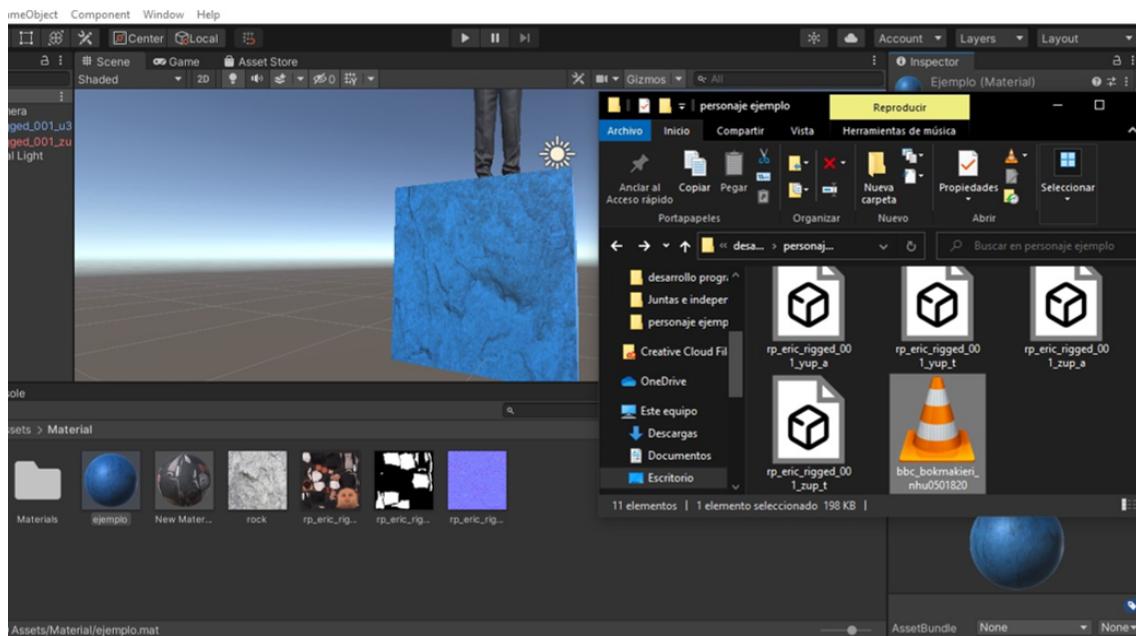
Nota. Visualización texturas agregadas.

En este caso se usa la textura tipo roca en formato de archivo .jpg, la cual se aplica al cubo. Si se quiere conservar el material azul, pero con textura rocosa como la imagen de muestra, la forma de aplicarlo es la siguiente: el material se arrastra de la pestaña proyecto hacia la casilla cuadrada al lado de la opción Albedo en la pestaña Inspector que está a la derecha, se presenta en la siguiente figura:



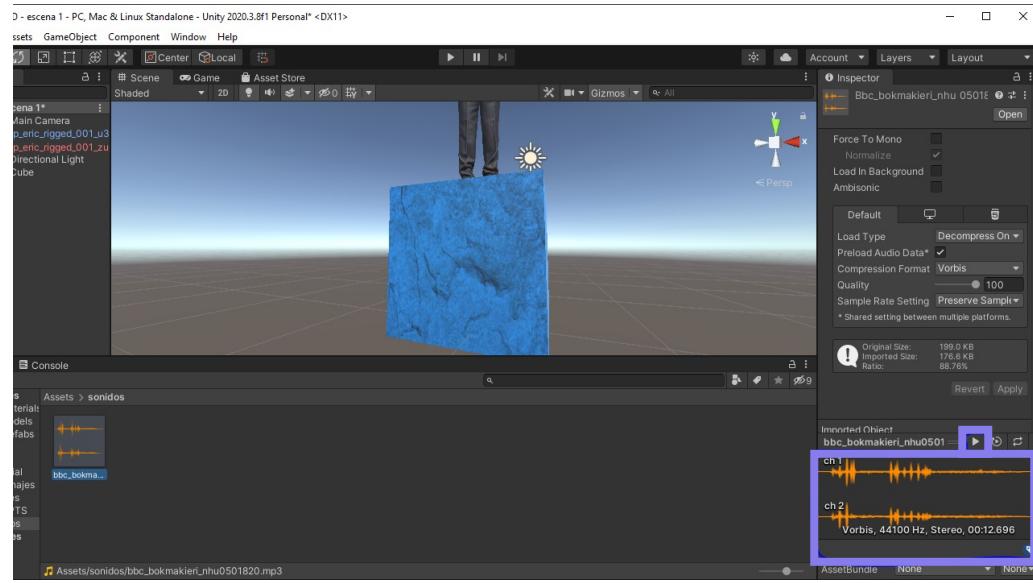
Nota. Ejemplo aplicación textura.

De la misma manera se agregan archivos en formato .mp3. wav. ogg. Para el caso del ejemplo se usará .mp3, la manera de integrarlo es idéntica, arrastrándolos desde la carpeta del computador al proyecto, como se presenta a continuación:



Nota. Ejemplo agregar archivos de audio.

Una vez importado el archivo de audio se puede escuchar en el panel Inspector, en la parte inferior, dando clic en el botón “play”, como se presenta en la siguiente figura.

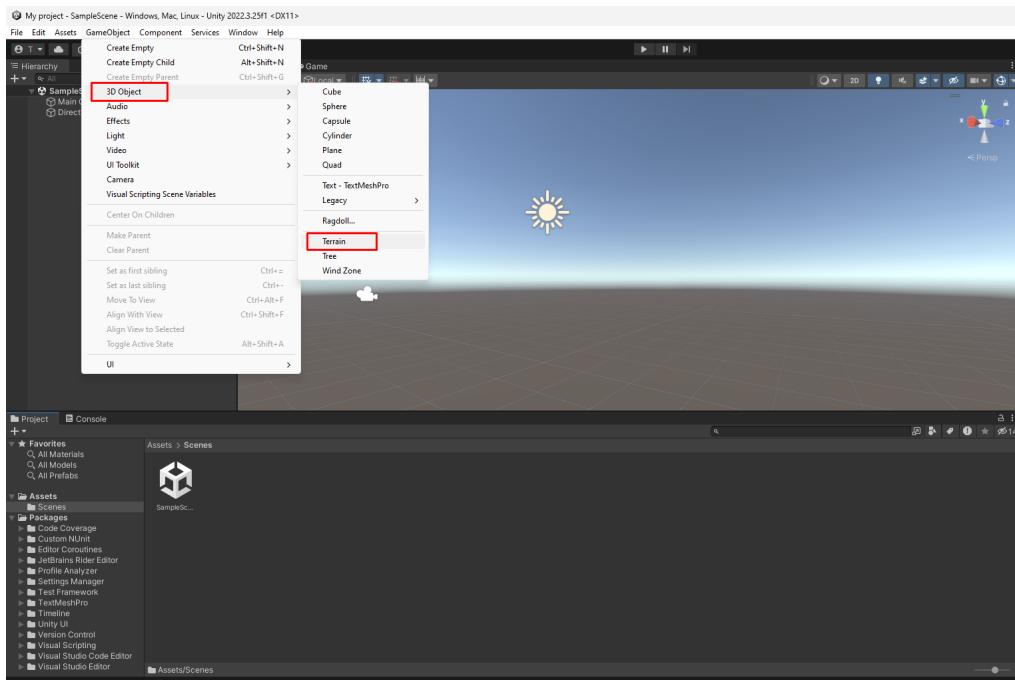


Nota. “Play” archivo de audio

7. Componer los escenarios (personajes, “props”, fondos)

Para la composición de escenarios Unity cuenta con herramientas que facilitan su desarrollo, como es el caso de la herramienta editor de terrenos, en la cual se pueden asignar texturas y color a los mismos. Para usar la herramienta crear terrenos se sigue la ruta de GameObject – 3D Object – Terrain, como se presenta a continuación:

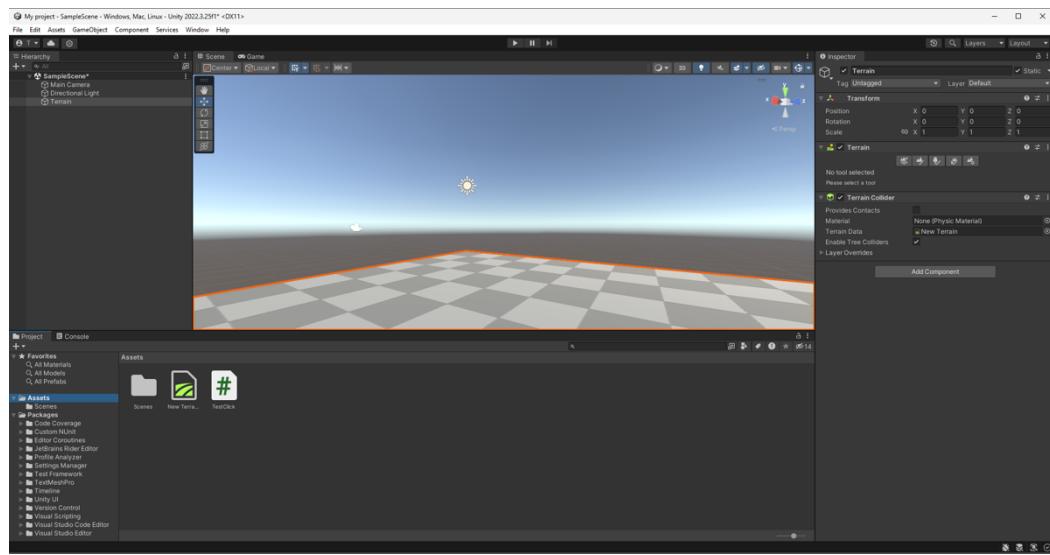
Figura 20. Terrain



Nota. Opción crear Terrain.

Al hacerlo se forma un plano de recuadros en el escenario, ahora en el panel de Inspector se puede configurar el terreno, como se presenta en la siguiente figura.

Figura 21. Configurar terreno



Nota. Opciones de configuración terreno.

Unity Terrenos

Ingresé a la carpeta anexos y descargue el “**Unity Terrenos**” puede consultar información complementaria relacionada con las propiedades de terrenos.

A continuación, puede visualizar la integración de “assets” en Unity para la creación de escenarios.

Video 2. Integración de “assets” en Unity para creación de escenarios



[Enlace de reproducción del video](#)

Síntesis del video: Integración de “assets” en Unity para creación de escenarios

El video Integración de “assets” en Unity para creación de escenarios enseña cómo crear un escenario 3D en Unity. Se inicia con la preparación de un nuevo archivo 3D y la configuración del terreno utilizando diversas herramientas y pinceles. Se explican técnicas para navegar en el escenario y ajustar parámetros como resolución de malla, texturas, y luces. El tutorial abarca la importación y aplicación de texturas, la adición de elementos naturales como árboles y vegetación, y la personalización de efectos como agua y viento, para crear un escenario realista y dinámico.

8. Iluminación

Es parte importante del diseño y desarrollo de los videojuegos, ya que cumple funciones específicas dentro del mismo. La primera es la de mejorar la estética de la escena en pantalla, pues a partir de una técnica de iluminación bien aplicada se puede crear efectos más realistas que logren transmitir diferentes sensaciones y emociones a los jugadores.

La segunda función es hacer más fácil la exploración del entorno, guiando al jugador por un camino mejor, iluminado o atrayendo su atención hacia una fuente de luz, además, de servir como un elemento interactivo (antorchas, linterna, fósforo, etc.) que le ayude a ubicarse y a percibir lo que ocurre a su alrededor, permitiéndole encontrar objetos que le pueden ser útiles más adelante. También, se usa para llamar la atención hacia objetos específicos, con una intensidad lumínica alta (brillo) que los hace resaltar y que son fundamentales para la fluidez y avance en el juego, ayudando a reducir la frustración del jugador, además, es un elemento esencial en el desarrollo de efectos visuales especiales que buscan atraer, sorprender, asustar o entretenir.

8.1. La iluminación en los videojuegos

La iluminación e intensidad lumínica se usa para la creación del clima en un nivel de juego; una luminosidad alta se utiliza para describir escenas de día al aire libre, espacios cálidos o espacios interiores bien iluminados, por ejemplo, niños disfrutando de un día de verano en la playa o una mujer leyendo una revista en un salón de belleza, transmitiendo sensaciones alegres y optimistas; mientras que una luminosidad baja es utilizada para el diseño de escenarios nocturnos, espacios fríos o espacios poco iluminados, por ejemplo, una cueva iluminada por la luz de una antorcha o una caminata a la luz de la luna, que pueden transmitir sensaciones de tranquilidad,

inseguridad o temor; una luminosidad moderada (ni muy alta ni muy baja) puede ser usada para lograr otros efectos atmosféricos, por ejemplo, un día lluvioso, un día con neblina, haciendo buen uso de la paleta de colores, o una tarde de tormentas, en la que también cabe destacar el uso del brillo para la creación de rayos (efecto visual especial).

Para lograr lo anterior, es necesario conocer cómo se deben ubicar y orientar las luces en las diferentes zonas de un nivel para mejorar la apariencia general de una escena, buscando simular un ambiente más realista fortalecido por un adecuado contraste o combinación de colores, que logre transmitir diversas sensaciones y emociones al jugador, de manera que se sumerja profundamente en la historia, por ejemplo, en un tétrico callejón iluminado solo por la luna (objeto fuente de luz) con una luz tenue, debería ubicarse en la parte superior y orientarse hacia el callejón, de manera que se cree un ambiente sombrío, que transmita sensaciones de temor o peligro, debido a que no se puede percibir bien lo que hay en ese entorno; si por el contrario lo que se busca es simular un día soleado, entonces, se podría usar luz de ambiente (“ambient light”) que no provenga de un objeto específico, sino que esté presente en todo el entorno, creando una atmósfera más cálida y brillante que genere sensaciones de bienestar.

A continuación, se analizará la iluminación en las escenas representadas en las siguientes figuras.

Figura 22. Escenas del videojuego Pikmin 3



Nota. Muestra escena de videojuego. Vidaextra (2013).

En Pikmin 3 se presenta un escenario diurno con una luz ambiente que ilumina toda la escena directamente, ocasionando que la definición y contornos de los personajes y demás elementos del entorno se vean más nítidos y que sea fácil de distinguir cada uno de sus detalles, logrando una simulación bastante realista; además, gracias al buen uso de la paleta de colores, la iluminación hace que estos se vean más vivos y mejor contrastados, contribuyendo a la calidad estética de la escena, la cual logra que el jugador se ubique en ese entorno natural, transmitiendo sensaciones de calidez, armonía y belleza, como se presenta en la siguiente figura.

Figura 23. Escenas del videojuego Unravel 2



En contraste con la escena anterior, Unravel 2 presenta un escenario nocturno en donde la fuente de luz está dada por un objeto de alta intensidad lumínica, creando la ilusión de que el entorno se ilumina a causa de una gran chispa brillante, la cual permite observar con mayor claridad los detalles de los objetos cercanos; sin embargo, al alejar la vista del foco de luz se pierde por completo los detalles, logrando percibir solamente algunas sombras, siluetas y contornos de los árboles y montañas que se muestran al fondo; aun así, aunque se trate de una escena no tan clara, los efectos de contrastes, luces y sombras logrados gracias al posicionamiento y direccionamiento de la luz hace que se genere una simulación bastante realista.

Figura 24. Escenas videojuegos Silent hill 2 y Resident evil 2 remake



En estos dos escenarios diseñados con una iluminación tenue, con el objetivo de generar una atmósfera de temor es fácil destacar cómo la luz tiene un papel funcional en el desarrollo y avance de un videojuego, pues por un lado permite la exploración del nivel, dejándole conocer al jugador qué es lo que sucede a su alrededor, pues es gracias a la fuente de luz representada por una linterna que se pueden percibir las posibles amenazas que se esconden en ese ambiente sombrío; por otro lado, esa misma linterna es de gran ayuda en la búsqueda de información y objetos útiles para el progreso, como se puede identificar en la imagen de la derecha, en donde al iluminar la ventana se ve que un objeto resalta gracias a sus colores cálidos, que al ser iluminados parecen más vivaces (saturados) y contrastan con los colores (fríos) que están a su alrededor, llamando inmediatamente la atención del jugador.

Como se ve, el diseño de estas fuentes de luz en escenarios oscuros es fundamental, ya que permite progresar en el juego, de manera que sirve como guía o le da indicios al jugador de lo que tiene que hacer, evitando que se sienta estancado, pues esto genera emociones negativas respecto al juego.

8.2. Luces en Unity

La iluminación y en general las luces en los videojuegos influye sobre todos los elementos presentes en la escena, por ejemplo, si se usa una fuente de luz directa, los colores de los objetos se verán mucho más vivos o saturados, con un mayor contraste y definición, generando una sensación de claridad, nitidez y de entorno seguro, transmitiendo un estado de ánimo positivo; por otro lado, si se usa una fuente de luz indirecta, los colores de los objetos se verán más opacos o menos saturados, con menor contraste y definición, dando la

sensación que puede ser de frescura, melancolía o de un ambiente sombrío, dependiendo de la combinación de colores y la intensidad lumínica.

Aquí cabe destacar que el contraste de colores especialmente entre fríos y cálidos hace que sobresalgan más los cálidos, lo que los hace más atrayentes a la vista.

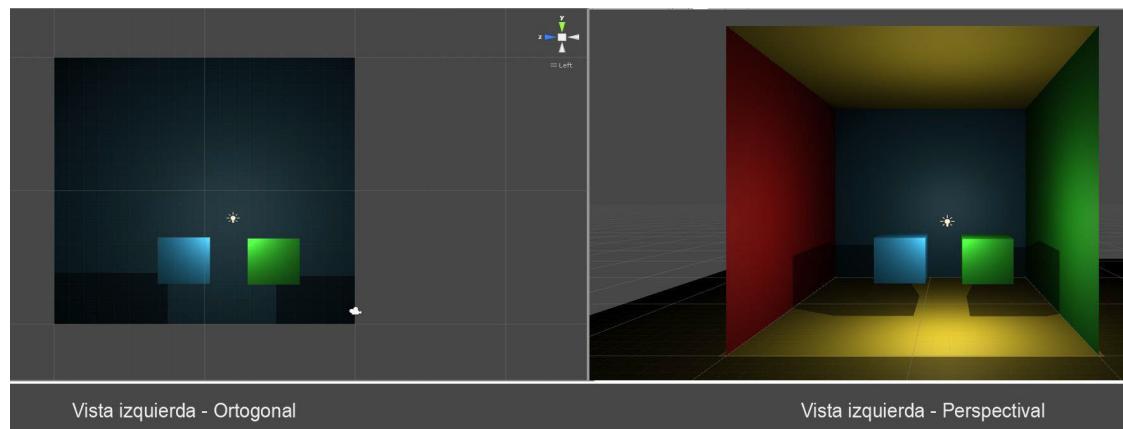
Unity Luces

En el Anexo de “Unity Luces” encontrará información complementaria de los tipos de luces en Unity.

9. Cámara

La cámara principal es la que se utiliza por defecto, que es la que muestra el visor de la escena general, la cual se puede rotar e interactuar con ella, desde la posición de perspectiva y el modo ortográfico, con el gizmo ubicado en la parte superior derecha se puede observar que tiene unos ejes y un cuadro amarillo. Con ellos clicando encima se puede navegar perspectiva y vista ortogonal, esto permite poder diseñar de una forma más cómoda y precisa, como se presenta la siguiente figura.

Figura 25. Vista ortogonal y perspectiva

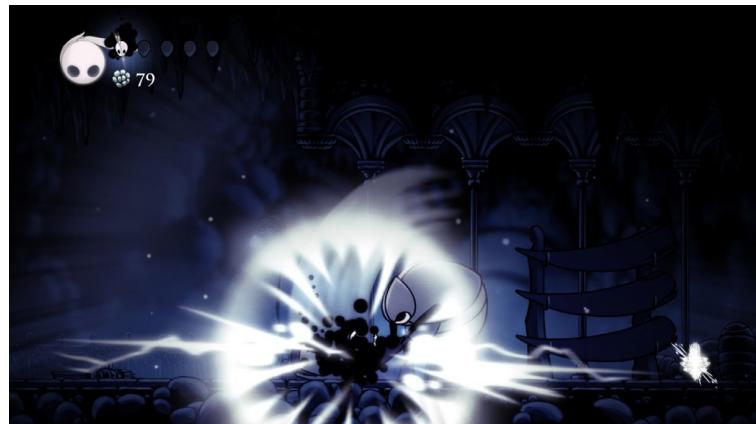


“Unity Camara”. Para la información adicional ingrese al anexo **“Unity Camara”** relacionado con las propiedades de cámaras, ejemplos de perspectivas, entre otros, puede consultarla en este documento.

10. Efectos visuales

Hacen que las acciones durante el juego se destaque mucho más, dándole más impacto visual a las escenas, por ejemplo, en un combate los golpes se hacen más visibles con efectos, el piso al combatir puede botar partículas, si hay un disparo se puede generar humo, puede haber un entorno con neblina, lluvia, fuego, entre muchos más. La siguiente figura presenta un ejemplo:

Figura 26. “Explosion hollow knight”



11. Interfaz

La interfaz en los videojuegos es muy importante, ya que es la antesala del juego en proceso, esta permite al usuario navegar, buscar información, tener objetivos claros, ajustar diversas opciones como el sonido, los gráficos, los efectos, los controles, pero también está presente durante el desarrollo del juego. Todo depende del tipo de historia que se quiera contar.

11.1. Interfaz del videojuego

Hay unos objetivos claros a tener en cuenta para el desarrollo de una interfaz. Es necesario tener en cuenta la narrativa que es la historia que se cuenta en el juego, y lo que se puede encontrar en él, como elementos externos que apoyan la jugabilidad. Para esto hay que preguntarse si estos componentes estarán presentes en el desarrollo del juego; hay cuatro elementos súper importantes a tener en cuenta: interfaz no diegética, diegética, espacial y meta.

Unity Interfaz. En el anexo “Unity Interfaz” descargue el documento en él puede consultar información y ejemplos de los tipos de interfaz enunciados previamente.

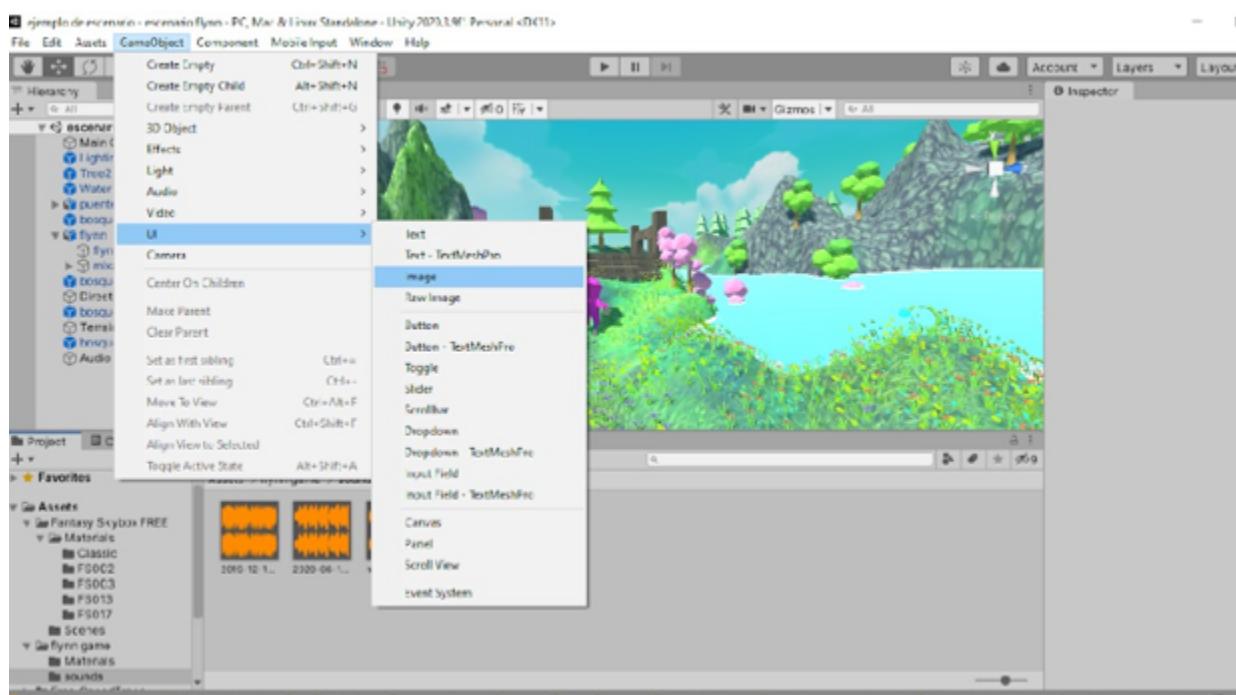
11.2. Diseño de interfaz del videojuego

Los elementos de información e interactividad dentro del juego, tales como barra de energía, menús de opciones y demás son llamados interfaz de usuario o UI (User interface). Es así como la interfaz es muy importante en el proyecto de videojuego, puesto que es la que brinda información del estatus del personaje.

Paso 1: image

Para empezar, lo primero que se debe hacer es ir a la pestaña “GameObject – UI – Image”, como se presenta en la siguiente figura. En la pestaña “hierarchy” aparece un nuevo elemento llamado “canvas”, el cual contiene el elemento “image” y el evento “system”.

Figura 27. Ejemplo creación interfaz.

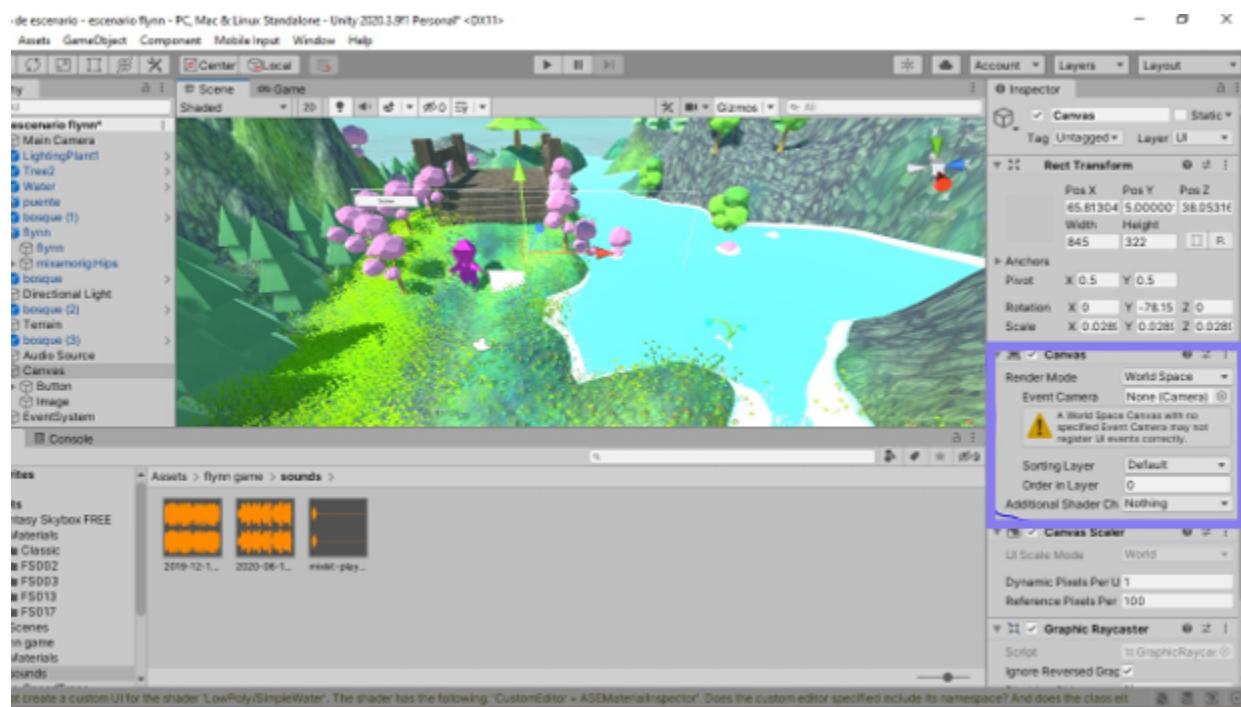


Los pasos anteriores crearán un rectángulo transparente, el cual es el “canvas” de la UI, así por ende este es el espacio donde se van a poner los elementos, luego se crea el botón desde “GameObjects” – UI – “button”.

Paso 2: canva

El canva en render “mode” se asigna a “world space” para que la interfaz aparezca siempre delante de los objetos que están en escena, como se presenta en la siguiente figura.

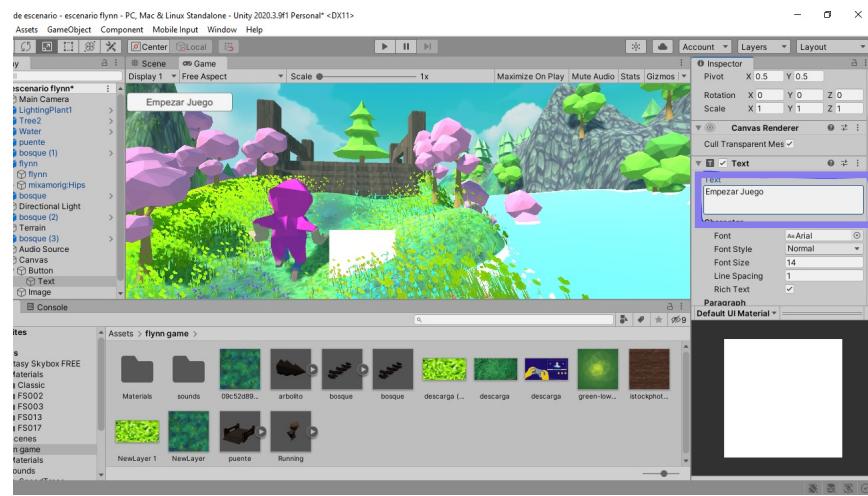
Figura 28. Ejemplo creación interfaz – render mode.



Paso 3: texto botón

Seleccionando en “Hierarchy” el elemento botón, se selecciona el Inspector y se agrega el texto que se quiere que aparezca en el botón, en este caso será el de comenzar juego, como se presenta a continuación:

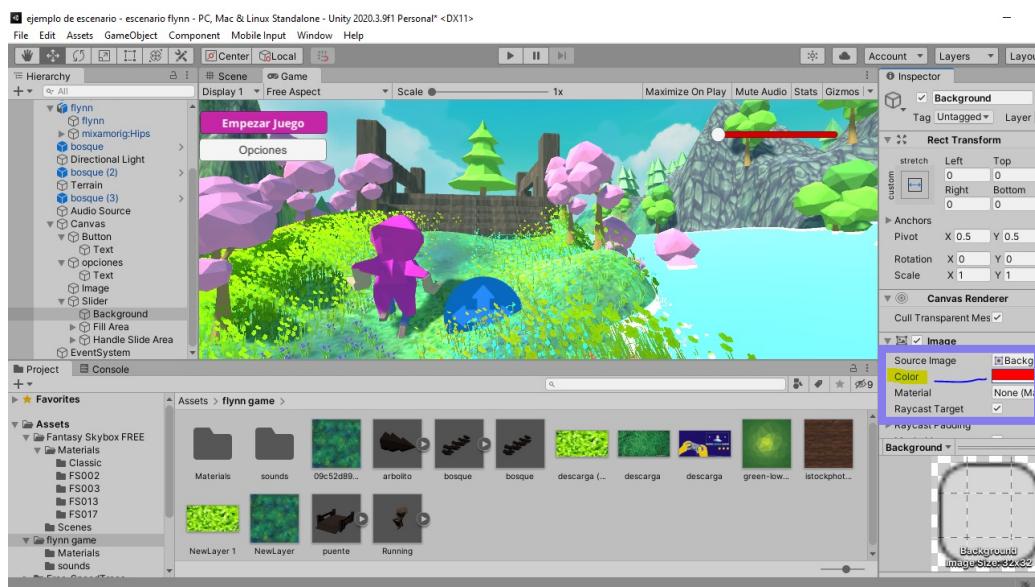
Figura 29. Ejemplo creación interfaz – texto botón



Paso 4: barra vida

Para poner una barra de vida se crea el objeto “Slider” accediendo a la barra de menú principal “GameObject - UI - Slider”. Una vez puesto y seleccionado aparecerá una barra “Slider”, se despliega y tendrá “background”, luego se selecciona el Inspector, se da clic en “image – color” y se asigna rojo, como se presenta en la siguiente figura.

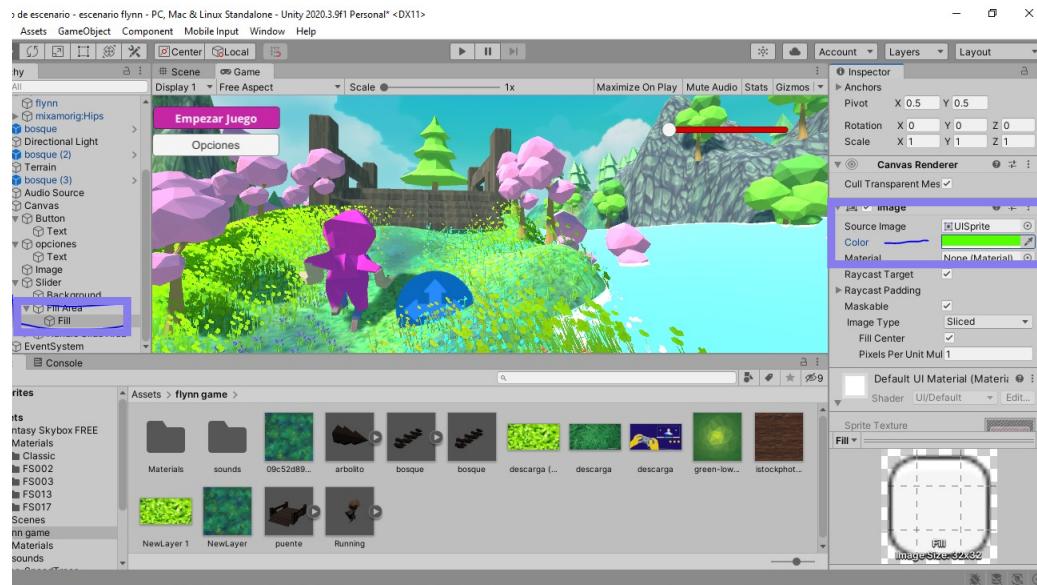
Figura 30. Ejemplo creación interfaz – barra vida.



Paso 5. “Slider” – fill area

Luego, en el “Slider” se despliega fill area – fill y se cambia el color a verde para mostrar la barra de vida cuando está llena, como se presenta en la siguiente figura.

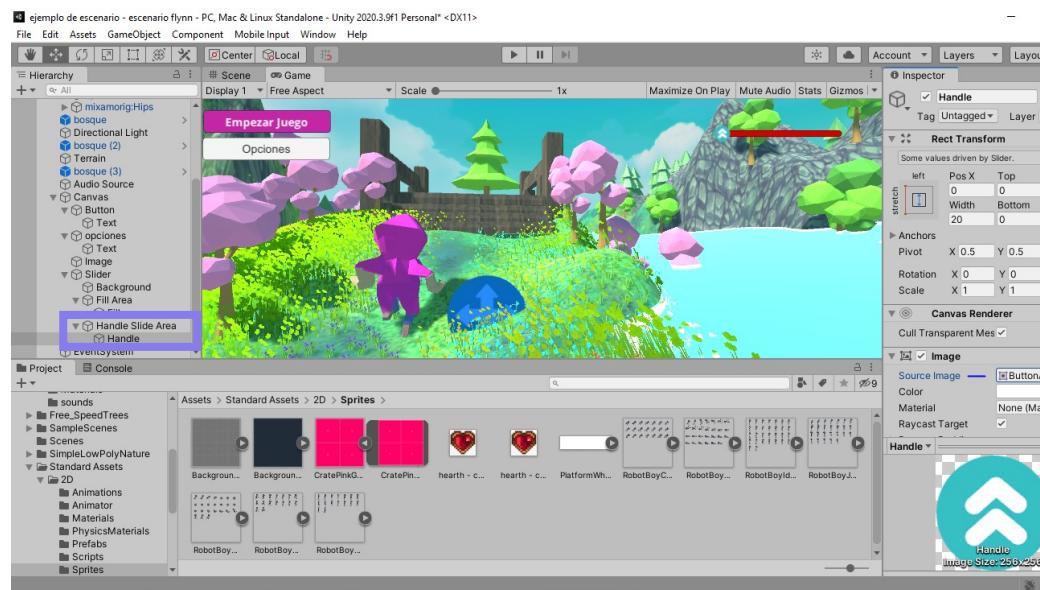
Figura 31. Ejemplo creación interfaz – fill área.



Paso 6: “Slider” – “handle slide area”

Ahora, en la tercera opción “handle slide area” se despliega y en “handle” se agrega un ícono en “file source” para la barra de vida, como se presenta a continuación:

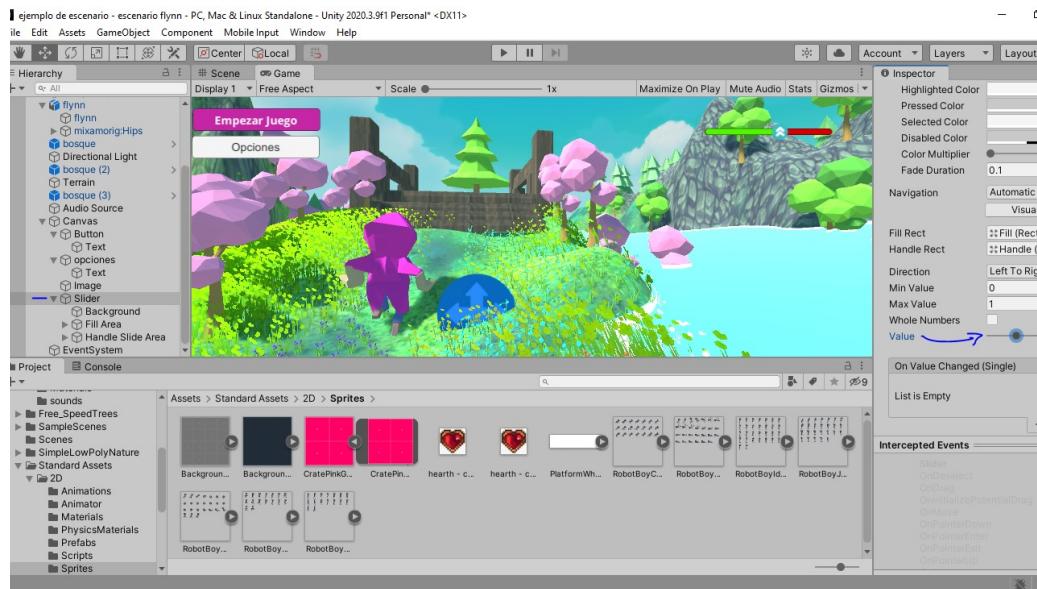
Figura 32. Ejemplo creación interfaz – “handle slide” área.



Paso 7: “Slider” – value”

Ahora para ver el “Slider” en funcionamiento se puede volver a “Slider” y en Inspector aumentar el valor para ver cómo se va llenando la barra de vida, como se presenta a continuación:

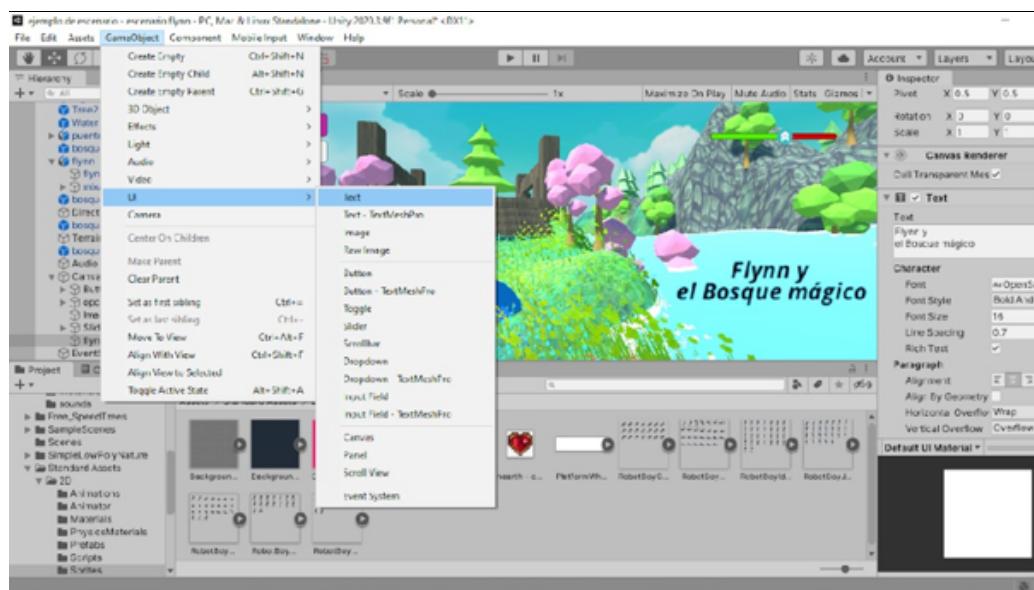
Figura 33. Ejemplo creación interfaz – “Slider”.



Paso 8: título

Finalmente, se va a GameObject – UI - Text para crear el título que acompañará la interfaz, como lo presenta la figura a continuación:

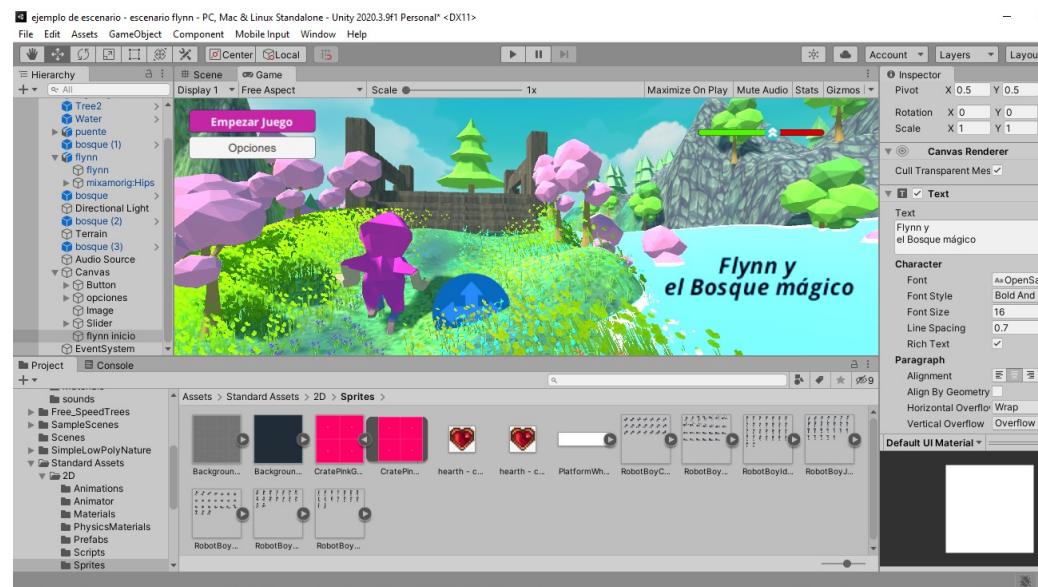
Figura 34. Ejemplo creación interfaz – título.



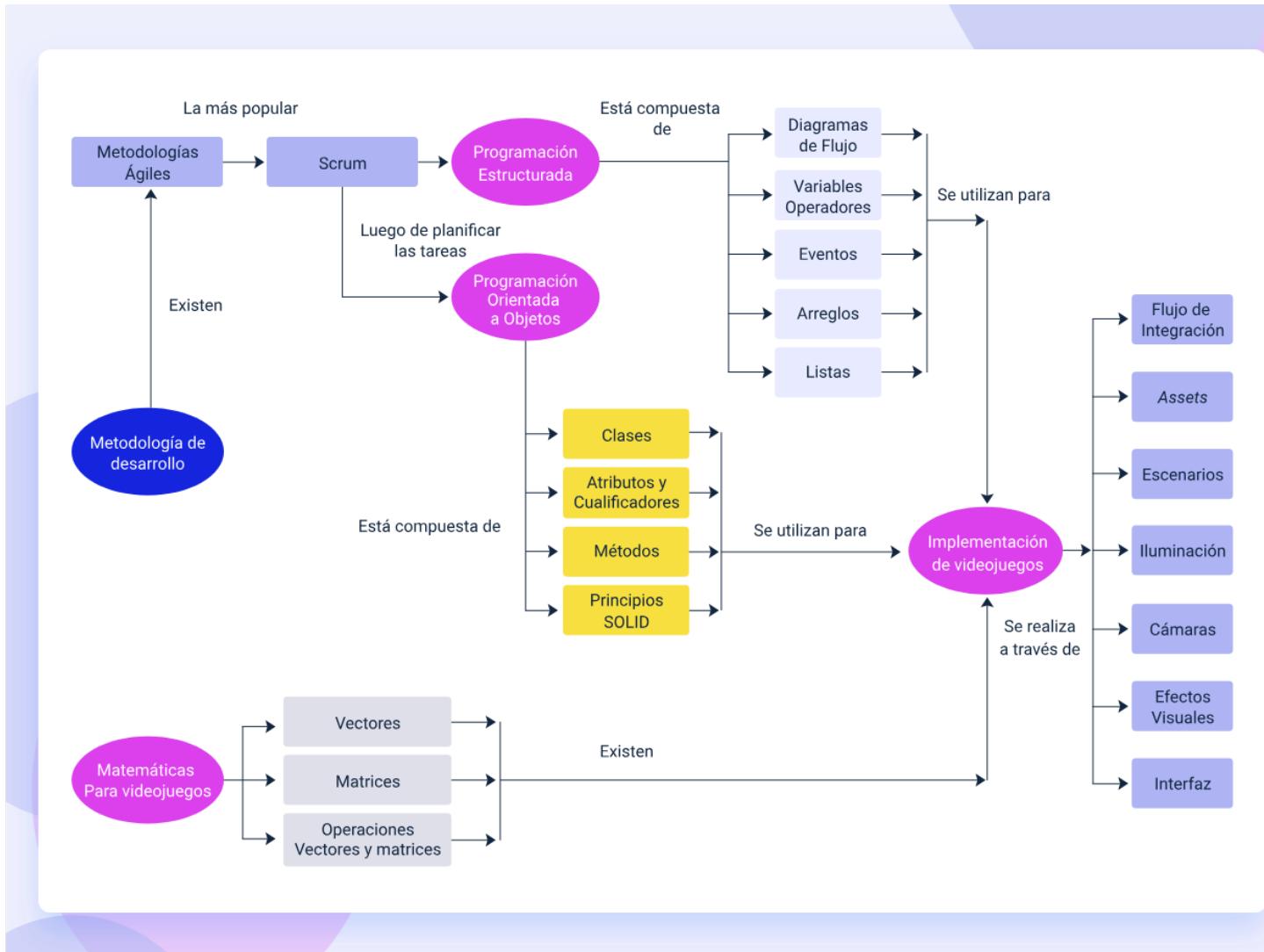
Paso 9: interfaz final

Con esto se tienen listos los elementos de la interfaz para iniciar el juego, como se ilustra en la siguiente figura:

Figura 35. Ejemplo creación interfaz – elementos agregados.



Síntesis



Material complementario

Tema	Referencia	Tipo de material	Enlace del recurso
1.2 Metodología SCRUM	Mejorar La Productividad (2022, marzo 24). SCRUM + KANBAN [metodología ágil] / EJEMPLO /.	Video	https://www.youtube.com/watch?v=6ZBIEOXJU1M
3. Programación Orientada a Objetos	¿Cómo te explico? (2021). Programación estructurada.	Video	https://www.youtube.com/playlist?list=PLHxyqMuBkJ3yyIwbNSt1GuqLFccq2YYTx
3. Programación Orientada a Objetos	yacklyon (2019, septiembre 24). CURSO de PYTHON 2020 Programación Orientada a Objetos (POO).	Video	https://www.youtube.com/watch?v=AqjTfmPitTQ&list=PLg9145ptuAigw5pV_DRznXdOsX19dorDs
4.1 Vectores	Matemáticas profe Alex. (2019). Vectores.	Video	https://www.youtube.com/playlist?list=PLeySRPnY35dEaZT3iJUNdDkgLEZE5x-Jd
4.2 Matrices	Matemáticas profe Alex. (2019, mayo 31). ¿Qué es una matriz? Sistemas de ecuaciones.	Video	https://www.youtube.com/watch?v=RJ96S2Pt3qU&list=PLeySRPnY35dEr2XewNdOI7Ft0tLIIKI
Diagrama de Clases	Nicosiore. (2017, octubre 25). Introducción a UML-1- Tutorial UML en español.	Video	https://www.youtube.com/watch?v=-OWd0tJAK10&list=PLMp96nOrGcaw5dhv8wOA5tVWWEmXtA2F
11.2 Diseño de interfaz del videojuego	Antony Morsas]. (2020, junio 26). Crea tu primer juego 2D en Unity desde cero.	Video	https://youtu.be/4XvfpCz_vh8?si=swea_6nC-y9npSgo

Glosario

Arreglos (Programación): son estructuras de datos que nos permiten almacenar otros datos dentro de este tipo de datos. Es decir, es un contenedor que nos permite tener varios datos al mismo tiempo almacenados.

Atributo: son las características individuales que diferencian un objeto de otro y determinan su apariencia, estado u otras cualidades.

Caso de uso: es la descripción de una acción o actividad. Un diagrama de caso de uso es una descripción de las actividades que deberá realizar alguien o algo para llevar a cabo algún proceso.

Ciclo: es una secuencia de instrucciones de código que se ejecuta repetidas veces, hasta que la condición asignada a dicho bucle deja de cumplirse.

Clase: una clase es un elemento de la programación orientada a objetos que actúa como una plantilla y va a definir las características y comportamientos de una entidad.

Condicional: los condicionales son estructuras que permiten elegir entre la ejecución de una acción u otra.

Instrucción: conjunto de instrucciones definidas, ordenadas y finitas que aplicadas permiten realizar una acción con el fin de resolver un problema o realizar una acción.

Método: es un conjunto de instrucciones definidas dentro de una clase, que realizan una determinada tarea y a las que podemos invocar mediante un nombre.

Programación Orientada a Objetos: la programación orientada a objetos se basa en el concepto de crear un modelo del problema de destino en sus programas.

Variable: sirven para guardar y recuperar datos, representar valores existentes y asignar unos nuevos.

Referencias bibliográficas

Ibargüengoitia, G. (2005). El lenguaje de modelado UM. Posgrado de Ciencia e Ingeniería en Computación, IIMAS, UNAM. México

Joyanes, L. (2008). Fundamentos de la programación. Algoritmos y Estructura de Datos, 4^a Edición. Madrid: McGraw-Hill.

López Sandoval, Carlo (2019). Unity. Aprende a Desarrollar Videojuegos. Rc Libros.

Trigas, M. (2012). Gestión de proyectos informáticos. Universidad Oberta de Catalunya.

<https://openaccess.uoc.edu/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf>

Créditos

Nombre	Cargo	Centro de Formación y Regional
Claudia Patricia Aristizábal	Líder del Ecosistema	Dirección General
Rafael Neftalí Lizcano Reyes	Responsable de Línea de Producción	Centro Industrial del Diseño y la Manufactura - Regional Santander
Carlos Andrés Cortes	Experto temático	Centro de Diseño e Innovación Tecnológica Industrial- Regional Risaralda
Paola Alexandra Moya Peralta	Diseñadora instruccional	Centro Industrial del Diseño y la Manufactura - Regional Santander
Yerson Fabian Zarate Saavedra	Diseñador de Contenidos Digitales	Centro Industrial del Diseño y la Manufactura - Regional Santander
Edward Leonardo Pico Cabra	Desarrollador Fullstack	Centro Industrial del Diseño y la Manufactura - Regional Santander
Emilsen Alfonso Bautista	Actividad Didáctica	Centro Industrial del Diseño y la Manufactura - Regional Santander
Zuleidy María Ruiz Torres	Validador de Recursos Educativos Digitales	Centro Industrial del Diseño y la Manufactura - Regional Santander
Luis Gabriel Urueta Alvarez	Validador de Recursos Educativos Digitales	Centro Industrial del Diseño y la Manufactura - Regional Santander
Daniel Ricardo Mutis Gómez	Evaluador para contenidos inclusivos y accesibles	Centro Industrial del Diseño y la Manufactura - Regional Santander