



Componente formativo

Programación orientada a objetos

Breve descripción:

El componente formativo se enfoca en la programación orientada a objetos para facilitar el desarrollo de líneas de código que pueda ser interpretadas por el motor de Unity bajo el cual se van a procesar y ejecutar.

Área ocupacional:

Ciencias Naturales.

Mayo 2023

Tabla de contenido

Introducción	3
1. Metodologías de desarrollo	4
1.1. Metodologías ágiles	5
1.2. Metodología SCRUM	9
1.3. Metodología SUM.....	12
2. Programación en Unity3D	15
2.1. Conceptos básicos de algoritmo y programa	17
2.2. Diagramas de flujo	22
2.3. Conceptos de variables, operadores, funciones, métodos, estructuras de control	26
2.4. Funciones de eventos y control de acciones	37
2.5. Estructura básica de un script de Unity3D	43
2.6. Control mediante interfaz de usuario (Inspector)	49
Síntesis	54
Material complementario.....	55
Glosario	57
Referencias bibliográficas.....	58
Créditos	58

Introducción

¿Qué es la Programación Orientada a Objetos? El siguiente video presenta un bosquejo general de lo que se aprenderá a lo largo del componente formativo.

Figura 1. Programación Orientada a Objetos



[Programación Orientada a Objetos](#)

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 video juego, por lo que es necesario comprender la estructura básica de un programa y como se debe iniciar su escritura en un 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 Csharp, utilizado en el motor gráfico 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

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.

a) 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.

b) 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.

c) 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 donde el cliente no tiene claro todos los requisitos de este y por lo tanto no se puede definir el alcance desde el principio, igualmente cuando durante el desarrollo se cambia de opinión o se adicionan 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, ver tabla:

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.
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.

Metodologías ágiles	Metodologías tradicionales
<p>Planifica los requisitos del proyecto en una lista de iteraciones (tareas) para realizar en corto tiempo, luego el desarrollo es iterativo.</p> <p>Hay retroalimentaciones por cada iteración para hacer mejoras al producto durante el proceso de desarrollo.</p> <p>Realiza entregas del proyecto al terminar cada iteración.</p>	<p>Planifica los requisitos del proyecto en un desarrollo que se lleva a cabo de manera lineal durante todo el proceso.</p> <p>No hay retroalimentaciones durante el proceso de desarrollo.</p> <p>Realiza solo una entrega cuando finaliza el proyecto.</p>

Manifiesto agile

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 “Manifiesto agile” 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:

- Los individuos y las interacciones que se puedan realizar sobre los procesos y las herramientas.
- La funcionalidad del “software” que se desarrolle sobre la excesiva documentación que se haga sobre este.
- 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:

- a) Priorizar la satisfacción del cliente.
- b) Aceptar los cambios.
- c) Entregar frecuentemente.
- d) Trabajar conjuntamente.
- e) Motivar a las personas.
- f) Conversar “face to face”.
- g) Medir progreso a través del “software” funcional.
- h) Promover el desarrollo sostenible.
- i) Mejorar la agilidad con excelencia técnica y atención.
- j) Lograr la simplicidad de los trabajos realizados.
- k) Promover la auto organización del equipo de trabajo.
- l) Reflexionar en equipo para mejorar la productividad.

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.

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.

Tarea #1

Paso 1: alista un cronómetro (celular o reloj).

Paso 2: toma 6 monedas de cualquier valor.

Paso 3: lánzalas al piso al mismo tiempo y activa el cronómetro.

Paso 4: anota en que cayó cada moneda (cara o cruz).

Paso 5: cuando termines con la última moneda debes para el cronometro.

Paso 6: registra el tiempo.

Tarea #2

Paso 1: igual a la Tarea#1.

Paso 2: igual a la Tarea#1

Paso 3: lanza la primera moneda y activa el cronómetro.

Paso 4: anota en que cayó (cara o sello) la primera moneda y después lanza la segunda moneda y repites el proceso hasta completar la última.

Paso 5: igual a la Tarea#1.

Paso 6: igual a la Tarea#1

Responde las preguntas:

¿Hubo diferencia de tiempo en la realización de las tareas?

¿Cuál fue la más eficiente?

¿Cuál tarea corresponde al método ágil? Explica por qué.

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, ver figura.

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:

1. 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:
 - A. ¿Qué he hecho desde ayer para aportar al cumplimiento de la tarea?
 - B. ¿Qué voy a hacer a partir de este momento para ayudar al equipo a cumplir el objetivo de la tarea?
 - C. ¿Qué obstáculos me impiden colaborar con el trabajo en equipo para conseguir el objetivo?
2. 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.
3. 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.
4. 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:

“PO 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.

“SM 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, auto organizarse como equipo y eliminar los impedimentos para maximizar la productividad.

“ST 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 Scrum, 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.

Las estructura de SUM según Acerenza (2009):

Fase 1 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. Allí se encuentra el desarrollo de concepto.

Fase 2 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. Acá se encuentra la planificación administrativa y especificación del videojuego.

Fase 3 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. Dentro de esta fase, está la planificación de la iteración, el seguimiento de la iteración, el desarrollo de características y finalmente el cierre de la iteración.

Fase 4 Beta: antes de liberar la versión del producto se verifican 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. Acá se encuentra la distribución de la versión Beta, verificación del videojuego y corrección del videojuego.

Fase 5 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. En la última fase, la liberación del videojuego y la evaluación del proyecto.

Dentro de todo este proceso se encuentra la **Gestión de riesgos** que 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 presente 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 videos juegos, la cual se encuentra en el material complementario.

2. Programación en Unity3D

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 es que tenga un marco, bisagras y cerradura sin importar el tamaño, el color y el tipo de material entre otras.

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, 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.

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.

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, ver 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](#).

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:

1. Analizar el problema o la situación.
 2. Diseñar el algoritmo.
 3. Realizar la programación.
-
1. 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.
 2. Diseñar el algoritmo: consiste en definir una serie de acciones a ejecutar en el proceso y que deben culminar con la solución del problema.
 3. 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 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:

Pseudocódigo

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

- a) Poseen reglas de sintaxis y semántica que se debe seguir al pie de la letra para que pueda ser interpretado por la computadora.
- b) 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.
- c) 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:

- a) 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.
- b) 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 muestra a continuación:




1. **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.
2. **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.
3. **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.
4. **Entrega del programa objeto:** 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.
5. **Enlazador.**





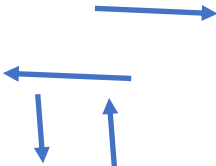
6. **Entrega del programa ejecutable:** es el programa que resulta de enlazar todas las funciones para convertir el programa objeto en un programa que la computadora pueda ejecutar.
7. **Ejecución del programa:** el programa está listo para ser ejecutado por el ordenador y producir los resultados que 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:

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

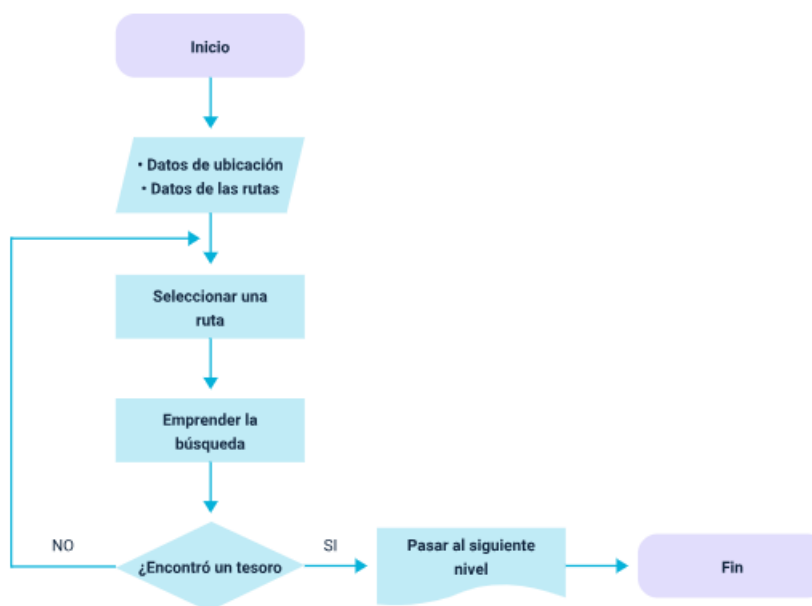
Los diagramas de flujo tienen unas reglas de elaboración, las cuales son:

- a) Se deben elaborar en orden lógico que empiece 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 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.
- e) Lo que se escriba al interior de los símbolos debe tener el menor número de palabras posibles o contener operaciones claras: $Z \leftarrow A - B$
- f) 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:

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

Tabla 3. 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.

A practicar

Diseñando el Algoritmo

Suponga que debe modificar el algoritmo diseñado anteriormente de tal forma que si el jugador no encuentra el tesoro, debe pedir una pista para lo cual se le descuentan 20 monedas y cuando encuentre el tesoro se le dará una recompensa de 50 monedas y podrá terminar esta fase del juego.

Tarea #1

Diseñar el algoritmo utilizando la técnica de Pseudocódigo.

Tarea #2

Diseñar el algoritmo utilizando la técnica de Diagrama de Flujo.

Para complementar el aprendizaje de la temática se recomienda hacer la siguiente lectura:

Diseño y construcción de algoritmos

Consultar en la base de datos SENA, [ingresando su usuario y contraseña](#).

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:

1. **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.

2. **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.

3. **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:

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

- f) El lenguaje hace diferencia entre letras mayúsculas y minúsculas; por ejemplo las variables X y x son diferentes.
- g) 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: es el elemento como tal, con el que se realiza la operación de adición, substracción, comparación:

+, -, <, >

Operandos: son los elementos sobre los cuales se aplican las operaciones:

variables, números.

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): _____

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:

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; \Rightarrow 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; \Rightarrow 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 ; \Rightarrow esto da como resultado falso (false).

Figura 3. Ejemplo bool total

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; ⇒ se ha declarado la variable a de tipo entero.

Int b = 20; ⇒ se ha declarado la variable b de tipo entero.

Int c = 8; ⇒ se ha declarado la variable c de tipo entero.

bool total; ⇒ 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 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):

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 para parámetros: una variable de tipo entera llamada `A` y otra del mismo tipo llamada `B`.

```
private int local (int A, int B)
```



```
{  
  
    Int total = A + B;  
  
    return total ;  
  
}
```

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.

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.

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

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 línea;                          ⇒ a continuación los datos que se  
escribirán son de tipo texto
```

```
    Console.Write ("Ingrese primer valor: ");    ⇒ escribe en pantalla (consola)  
ese texto
```

```
    Línea = Console.ReadLine ();              ⇒ capturamos en pantalla el  
número introducido
```

```
    Numero1 = int.Parse (línea);              ⇒ convierte la línea ingresada en  
un número entero
```

```
    Console.Write ("Ingrese segundo valor:");    ⇒ escribe en pantalla (consola)  
ese texto
```

```
    Línea = Console.ReadLine ();              ⇒ capturamos en pantalla el número  
introducido
```

```
    Numero2 = int.Parse (línea);              ⇒ 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
(console) ese texto

    } else {                          ⇒ si es falso
(else)

        Console.WriteLine ("No aplicar descuento");         ⇒ convierte la línea ingresada
en un número entero
```

```
    }  
}
```

Estructura repetitiva

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:

1. 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.
2. 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:

- a) Mouse.
- b) Teclado.
- c) Interfaz de usuario.
- d) 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:

1. 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.
2. 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.
3. Se minimizan las líneas de código ya que es el usuario quien genera el evento y el sistema se encarga de responder.
4. La interfaz del programa es mas amigable ya que ofrece opciones como botones, menús, ventanas, casillas de verificación, entre otras y los usuarios pueden escoger cual de estos utilizar para navegar en él.

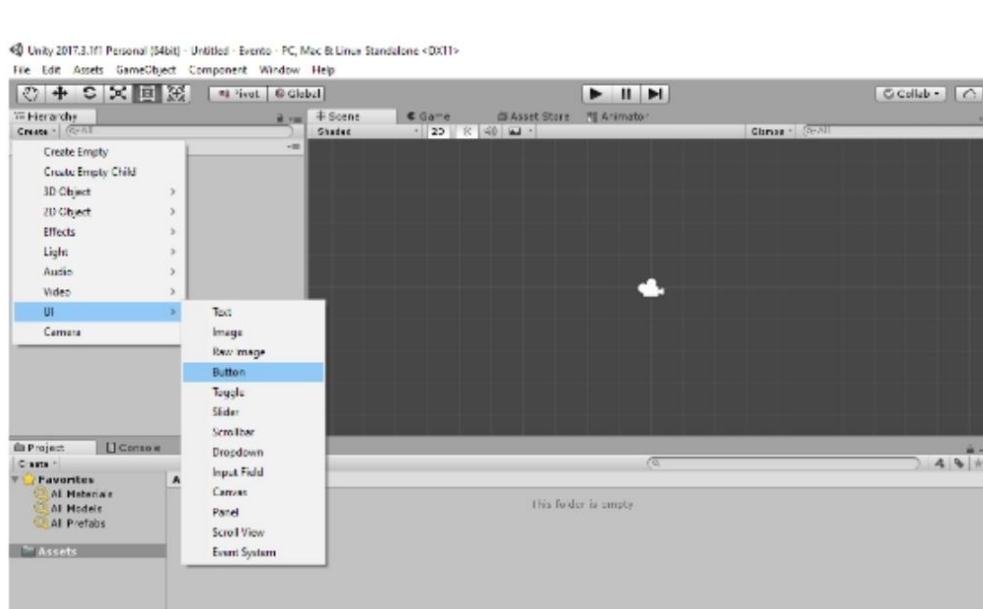
Ejemplo

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:

Acción 1

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

Figura 4. Acción 1



Acción 2

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.

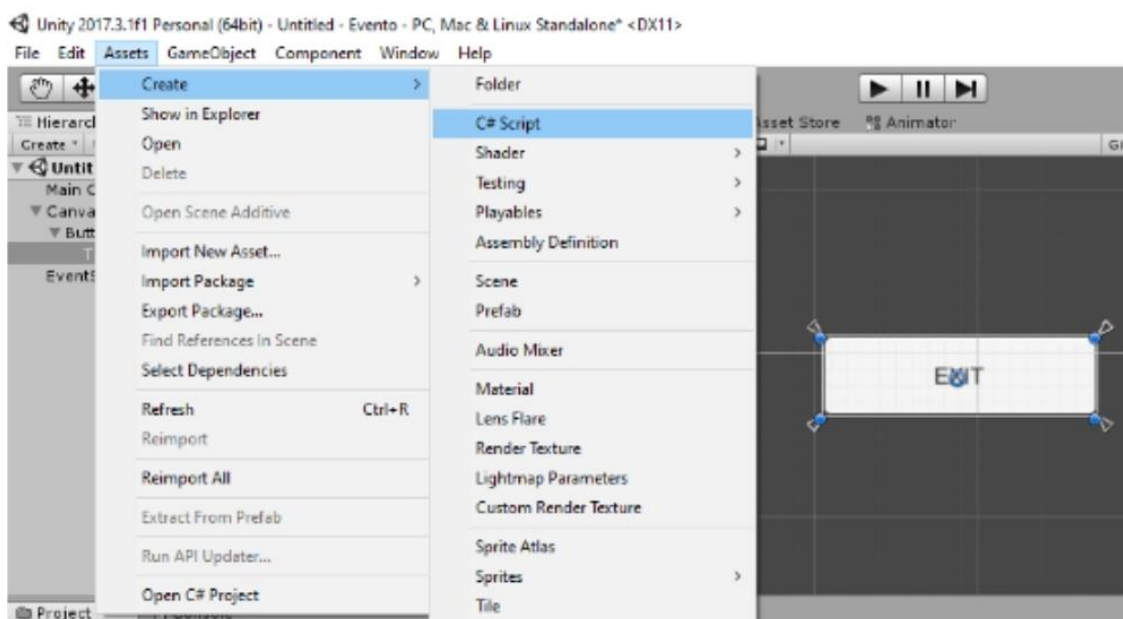
Figura 5. Acción 2



Acción 3

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.



Acción 4

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:

Figura 6. Acción 4

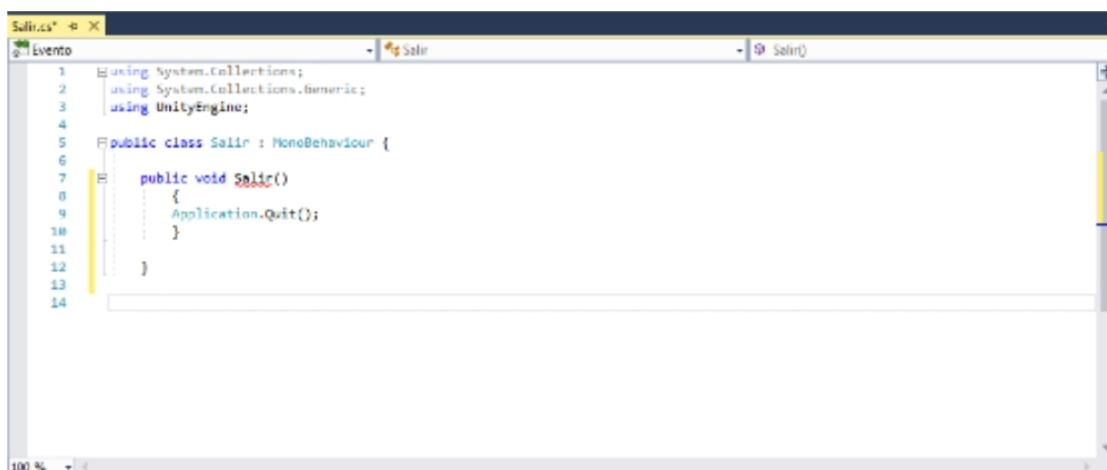


Acción 5

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.

Figura 7. Acción 5

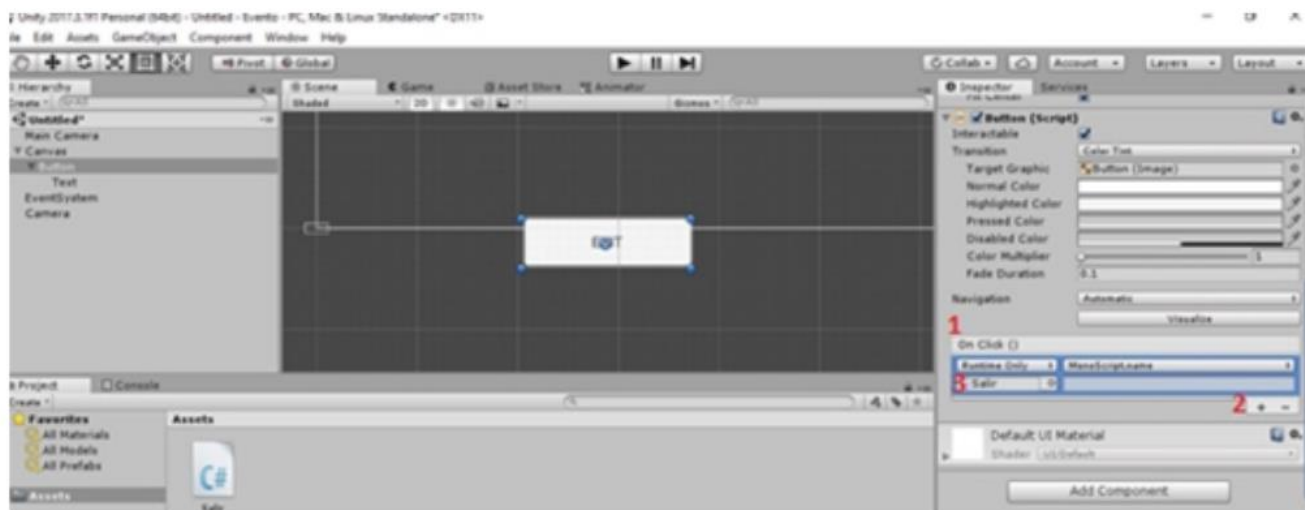


Acción 6

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).

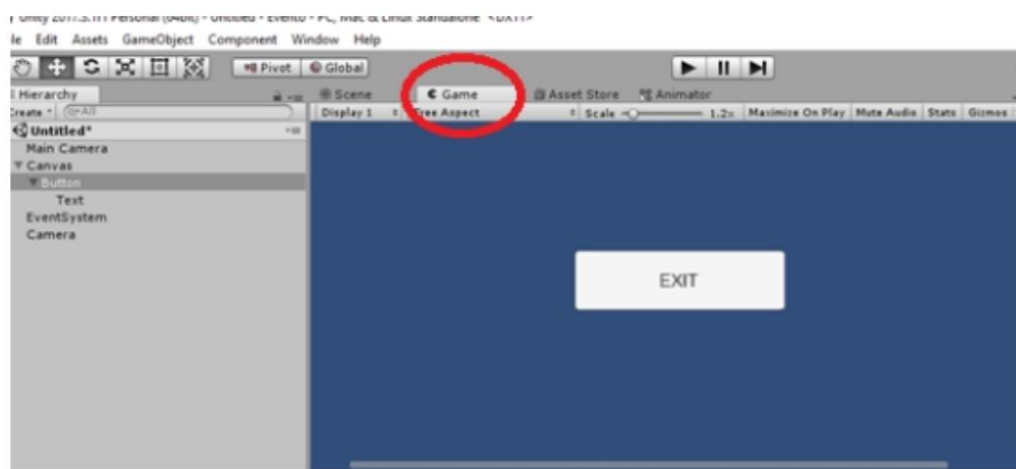
Figura 8. Acción 6



Acción 7

Cuando la función está lista para ser utilizada se observa en modo Game y no en modo escena para visualizar como se verá el botón del evento en la interfaz del usuario para que pueda ser accionado por éste.

Figura 9. Acción 7



2.5. Estructura básica de un script de Unity3D

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.

Figura 10. Variables



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.

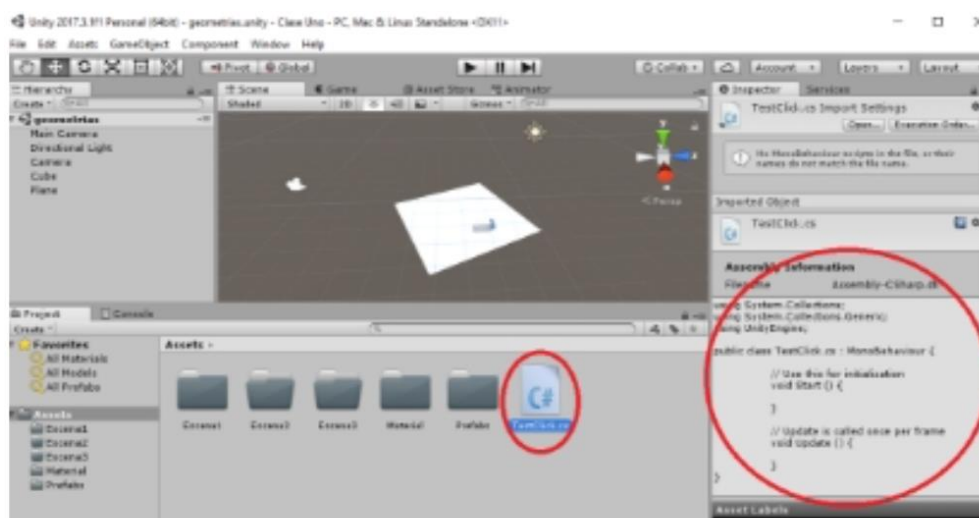
Update: función que se utiliza para el movimiento de los objetos sin utilizar el método física de Unity y se repite cada vez que se necesite para actualizar el brestado de este.

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*.

Figura 11. Funciones



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

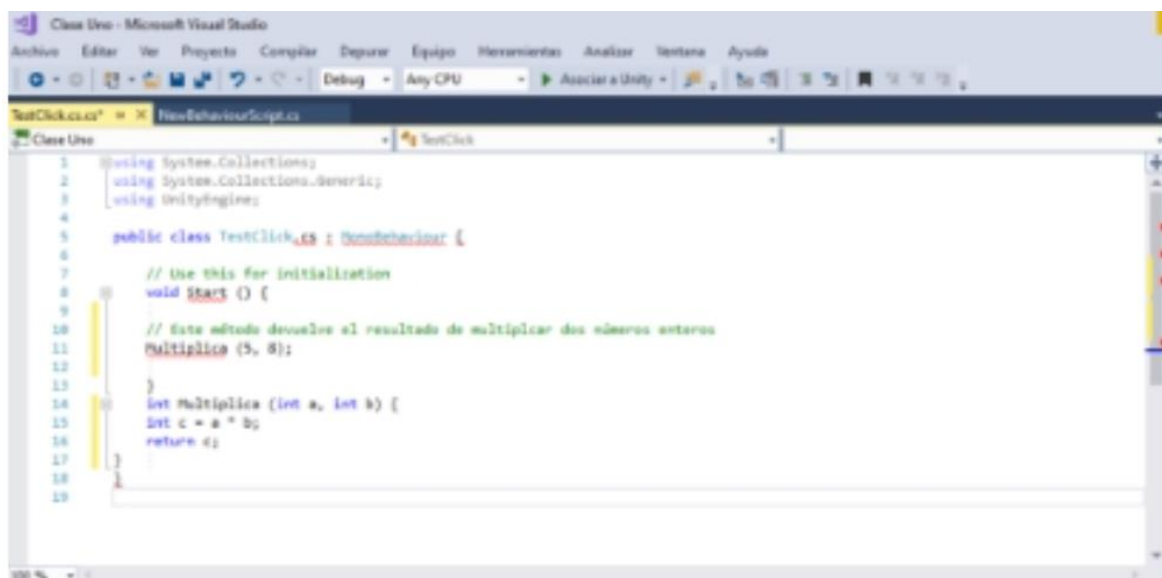
- Puede ser pública o privada
- Debe tener un identificador (nombre)
- Debe Incluir parámetros
- Debe retornar un tipo de dato así sea nulo

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 {
```

Figura 12. Editor



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 abstraccionismo, 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 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.

Figura 13. Clases

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 objetivos 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.

Ejemplo: crear un Script e identificar los espacios de nombres.

Para crear el Script hay dos formas:

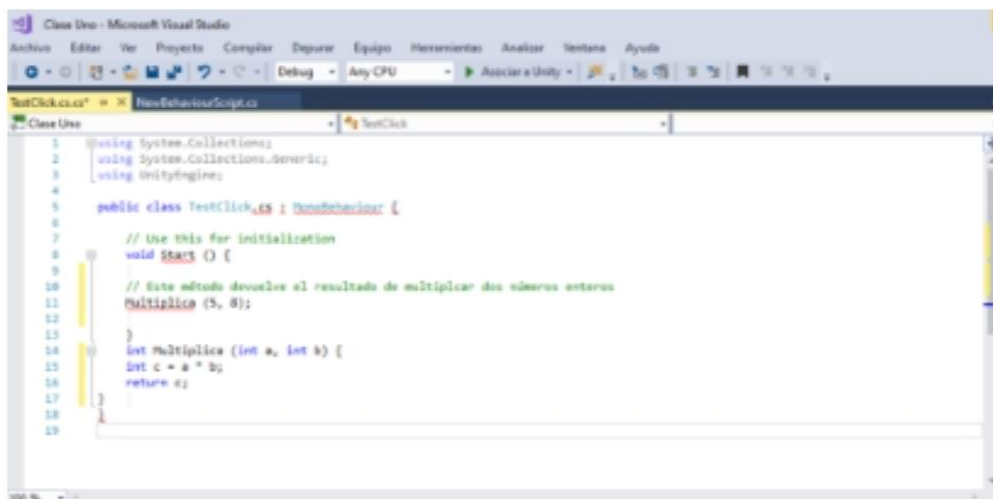
Assets > Create > C# Script

Project > Create > C# Script

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 agrupara 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.

Figura 14. Interfaz



2.6. Control mediante interfaz de usuario (Inspector)

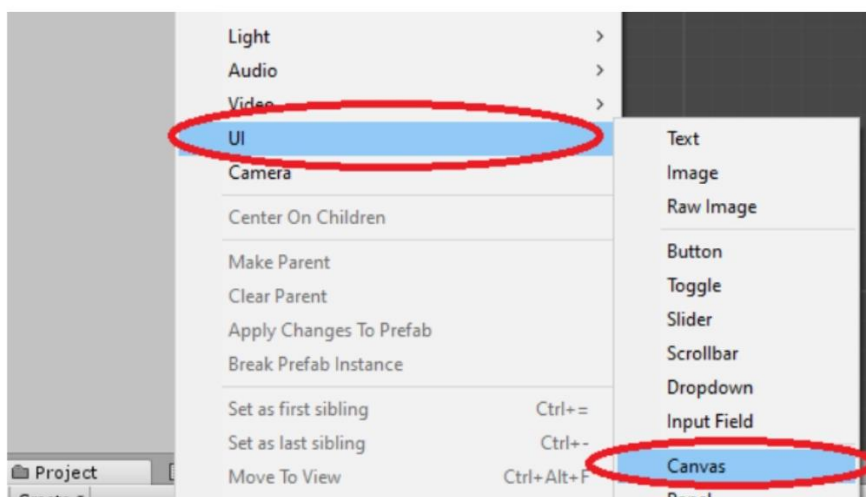
La interfaz de usuario es (UI) es la manera en que este interacciona con el programa; en este caso la forma en que un jugador se comunica con el videojuego interactuando con él. El motor de Unity posee un componente llamado Canvas que actúa como un contenedor o plantilla donde se pueden colocar todos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

Acción 1

- Clic en la opción de menú GameObject.
- Clic en la opción UI.
- Clic en la opción Canvas.

Figura 15. Acción 1 Inspector



Acción 2

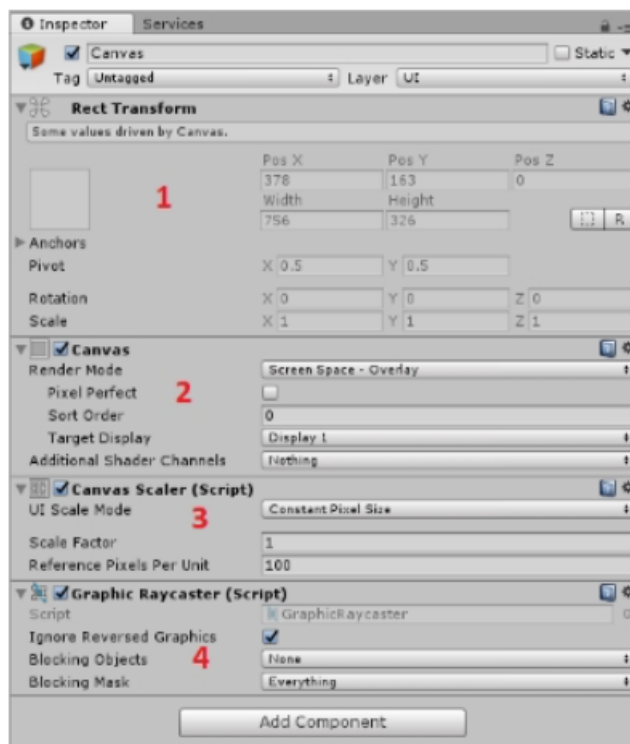
Cuando se ha añadido Canvas a la escena en la ventana del Inspector se observa cada uno de sus componentes:

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 tratara 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.

Figura 16. Acción 2 Inspector



Ejemplo

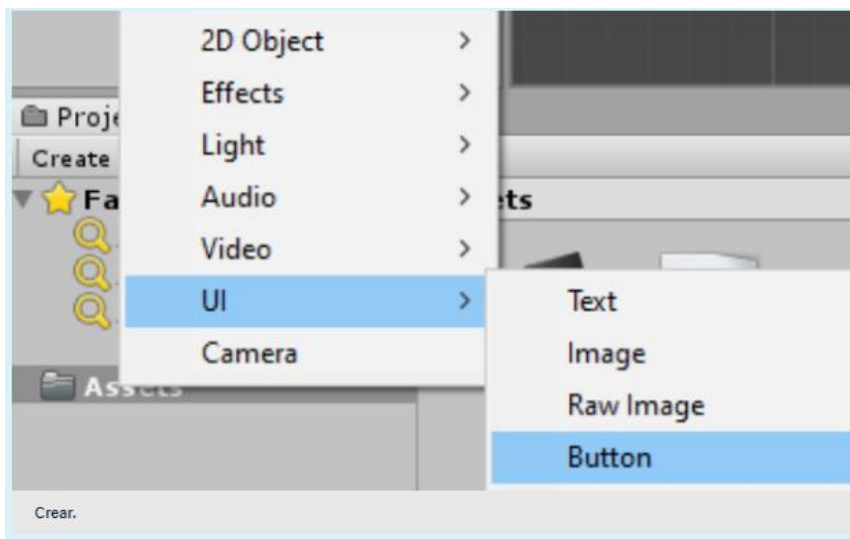
Construir un botón de inicio de sesión utilizando interfaz de usuario.

Paso 1

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.

Figura 17. Paso 1

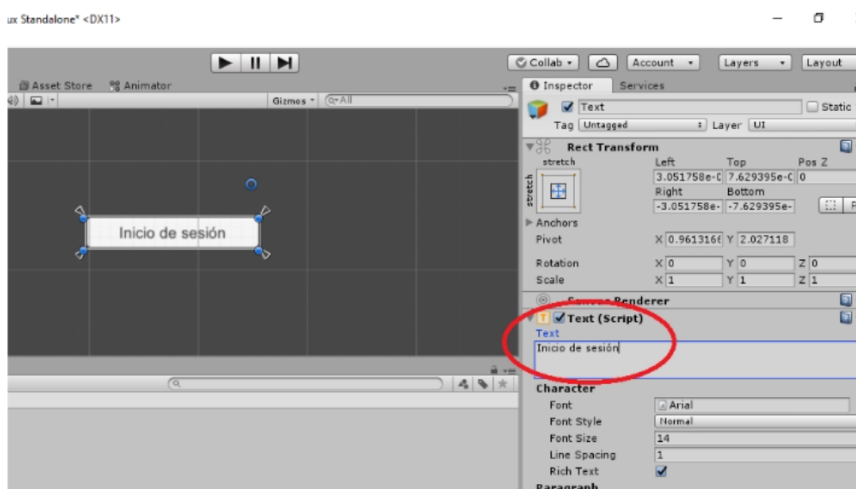


Paso 2

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.

Figura 18. Paso 2



Para complementar el aprendizaje de las temáticas de programación en Unity3D se recomienda consultar lo siguiente:

López, B. (s.f.). [Fundamentos de Programación.](#)

Unity. (2018). [Documentation.](#)

Síntesis

La programación orientada a objetos facilita el desarrollo de líneas de código que posteriormente son interpretadas por el motor de Unity, que finalmente se encarga de procesar y ejecutar, esto requiere de diversas metodologías abordadas en este componente formativo, el que se esquematiza en el siguiente mapa conceptual:



Material complementario

Tema	Referencia APA del Material	Tipo de material	Enlace del Recurso o Archivo del documento material
Metodologías ágiles	RobertoTouza.com. (s.f). <i>Gestión Ágil de proyectos</i> . https://robertotouza.com/agile/gestion-agil-de-proyectos/	Artículo	https://robertotouza.com/agile/gestion-agil-de-proyectos/
Metodologías ágiles	Agilemanifesto.org. (2001). <i>Manifiesto para el desarrollo de software ágil</i> . http://agilemanifesto.org/	Documento	http://agilemanifesto.org/
Metodología SCRUM	Trigas, M. (s.f). <i>Metodología Scrum</i> . TFC. http://openaccess.uoc.edu/webapps/o2/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf	PDF	http://openaccess.uoc.edu/webapps/o2/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf
Metodología SUM	SUM. (s.f). <i>Introducción a SUM</i> . http://www.gemserk.com/sum/	Artículo	http://www.gemserk.com/sum/
Programación en Unity3D	Joyanes Aguilar, L. (2020). Propiedades fundamentales de orientación a objetos. <i>Fundamentos de programación: algoritmos, estructura de datos y objetos</i> (pp. 576-590). McGraw-Hill.	Libro	https://www-ebooks7-24-com.bdigital.sena.edu.co/?il=10409
Diagramas de flujo	Mancilla Herrera, A., Capacho Portilla, J., Ebratt Gómez, R.(2016). Cap. 3. Primitivas algorítmicas. Diseño y	Libro	https://www-ebooks7-24-com.bdigital.sena.edu.co/?il=6425

	construcción de algoritmos. Ediciones de la U.		
Estructuras de control	Microsoft. (2021). <i>Documentación de C#</i> . https://docs.microsoft.com/en-us/dotnet/csharp/	Manual	https://docs.microsoft.com/en-us/dotnet/csharp/
Estructuras de control	C#Sharp. Com.es. (2021). <i>Fundamentos del lenguaje C#</i> . https://csharp.com.es/fundamentos-del-lenguaje-c/	Artículo	https://csharp.com.es/fundamentos-del-lenguaje-c/
Control mediante interfaz de usuario (Inspector)	Academia Android. (2015). <i>Canvas: creación de interfaz de usuario en Unity</i> . https://academiaandroid.com/canvas-creacion-de-interfaz-de-usuario-en-unity/	Blog	https://academiaandroid.com/canvas-creacion-de-interfaz-de-usuario-en-unity/
Control mediante interfaz de usuario (Inspector)	Unity. (2018). <i>Documentation</i> . https://docs.unity3d.com/es/2018.4/Manual/UnityManual.html	Manual	https://docs.unity3d.com/es/2018.4/Manual/UnityManual.html

Glosario

Algoritmo: es una secuencia que se lleva a cabo de forma lógica y secuencial que tiene como objetivo la solución de un problema (colombiaaprende.edu.co, s.f).

Diagrama de flujo: forma gráfica de representación de un algoritmo utilizando simbología (colombiaaprende.edu.co, s.f).

Función: es un método en programación que permite dividir un programa en bloques más pequeños (lenguajesdeprogramador.net, s.f).

Metodologías ágiles: conjunto de buenas prácticas y técnicas para que el desarrollo de proyectos se realice en menos tiempo y de forma más dinámica (Da silva, 2021).

Script: es un conjunto de instrucciones escritas en un lenguaje de programación siguiendo reglas de sintaxis para que se puedan traducir a lenguaje de máquina (gamedevtraum, 2021)

Scrum: metodología de trabajo ágil que está basada en ciclos con periodos de tiempo cortos que pueden ser semanas o meses para entregar una parte del proyecto (Da silva, 2021).

SUM: metodología utilizada para llevar a cabo el ciclo de vida de un software desarrollado secuencialmente en fases iterativas e incrementales que se ejecutan en corto y por equipos pequeños (Acerenza, 2009).

Variable: espacio de memoria de un ordenador o dispositivo electrónico destinado al almacenamiento de datos (lenguajesdeprogramador.net, s.f).

Referencias bibliográficas

Acerenza, N. et al. (2009). *Una Metodología para Desarrollo de Videojuegos*. [PDF].
https://www.fing.edu.uy/sites/default/files/biblio/22811/asse_2009_16.pdf

Da Sila, D. (2021). *¿Qué es la metodología ágil? ¿Para qué sirve?*
<https://www.zendesk.com.mx/blog/metodologia-agil-que-es/>

GameDevTraum. (2021). *Scripts en programación*.
<https://gamedevtraum.com/es/programacion-informatica/introduccion-a-la-programacion/que-es-script-programacion/>

Lenguajesdeprogramador.net. (s.f.). *¿Qué es una FUNCION (o un método) en programación?* <https://lenguajesdeprogramacion.net/diccionario/que-es-una-funcion-o-metodo-en-programacion/>

Lenguajesdeprogramador.net. (s.f.). *¿Qué es una variable en programación?*
<https://lenguajesdeprogramacion.net/diccionario/que-es-una-variable-en-programacion/>

MinTIC, MinEducación. (s.f.). *Lección 1: ¿Qué es un algoritmo?*
<http://contenidos.sucerman.com/nivel1/programacion/unidad1/leccion1.html>

MinTIC, MinEducación. (s.f.). *Lección 2: Los diagramas de flujo*.
<http://contenidos.sucerman.com/nivel1/programacion/unidad1/leccion2.html>

Créditos

Nombre	Cargo	Regional y Centro de Formación
Claudia Patricia Aristizábal Gutiérrez	Responsable del equipo	Dirección General
Liliana Victoria Morales Gualdrón	Responsable de línea de producción	Centro de Gestión De Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Rafael Neftalí Lizcano Reyes	Asesor Pedagógico Ecosistema RED	Regional Santander - Centro Industrial del Diseño y la Manufactura
Olga Lucía Mogollón Carvajal	Experto Temático	Centro para la Industria de la Comunicación Gráfica -CENIGRAF-
Luz Aida Quintero Velásquez	Diseñadora instruccional	Regional Distrito Capital - Centro de Gestión Industrial
Ana Catalina Córdoba Sus	Revisora Metodológica y Pedagógica	Regional Distrito Capital- Centro de Diseño y Metrología
Jhon Jairo Rodríguez Pérez	Diseñadora instruccional	Regional Distrito Capital - Centro para la Industria de la Comunicación Gráfica
Gloria Amparo López Escudero	Adecuadora Instruccional 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Andrés Felipe Velandia Espitia	Metodólogo para la formación virtual-2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Yuly Andrea Rey Quiñones	Diseñador web	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Jhon Edison Castañeda Oviedo	Desarrollador Fullstack	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Laura Gisselle Murcia Pardo	Animación y Producción audiovisual	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital

Carolina Coca Salazar	Evaluadora de contenidos inclusivos y accesibles	Centro de Gestión De Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Castaño Pérez Leyson Fabian	Validador de recursos educativos	Centro de Gestión De Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital