**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

|  |  |
| --- | --- |
| PROGRAMA DE FORMACIÓN | DESARROLLO DE VIDEOJUEGOS Y ENTORNOS INTERACTIVOS |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA | 220501088 - Producir la lógica del videojuego de acuerdo con el diseño y técnicas de desarrollo. | RESULTADOS DE APRENDIZAJE | 220501088-1. Definir la estructura del código del videojuego de acuerdo con el documento de diseño. |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 8 |
| NOMBRE DEL 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. |
| PALABRAS CLAVE | Algoritmo, diagrama de flujo, metodologías ágiles, scrum, SUM, script. |

|  |  |
| --- | --- |
| ÁREA OCUPACIONAL | 5 - ARTE, CULTURA, ESPARCIMIENTO Y DEPORTES |
| IDIOMA | Español |

1. **TABLA DE CONTENIDO**

**Introducción**

**1. Metodologías de desarrollo**

1.1. Metodologías ágiles

1.2. Metodología SCRUM

1.2.1*. Pregame*

*1.2.2. Game*

*1.2.3. Porst game*

1.3. Metodología SUM

**2. Programación en Unity3D**

2.1. Conceptos básicos de algoritmo y programa

2.1.1. *Algoritmo.*

*2.1.2. Programa*

2.2. Diagramas de flujo

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

2.3.1. *Variables*

*2.3.2. Operadores*

*2.3.3. Funciones y métodos*

*2.3.4. Estructuras de control*

2.4. Funciones de eventos y control de acciones

2.5. Estructura básica de un script de Unity3D

2.6. Control mediante interfaz de usuario (Inspector)

1. **DESARROLLO DE CONTENIDOS**

**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.

****

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. | Variety of time objects and a man landing page Free Vector |
| Process concept illustration Free Vector | **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. | Product iteration concept illustration Free Vector |

* 1. **Metodologías ágiles**

Como se mencionó anteriormene, 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 1:

**Tabla 1**

*Metodologías ágiles vs Metodologías tradicionales*

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

**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:

1. Los individuos y las interacciones que se puedan realizar sobre los procesos y las herramientas.
2. La funcionalidad del software que se desarrolle sobre la excesiva documentación que se haga sobre este.
3. La colaboración con el cliente sobre los detalles de los aspectos contractuales.
4. 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:

|  |
| --- |
| **Infografía**  CF008\_1.1.\_manifiesto agil |

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.

|  |  |
| --- | --- |
| ¡Vamos a Practicar! | Boton Modal  Anexo: CF008\_1.1.vamos a practicar |

* 1. **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 conocimiento y la experiencia de las personas que participan en ellos.

* + 1. ***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.

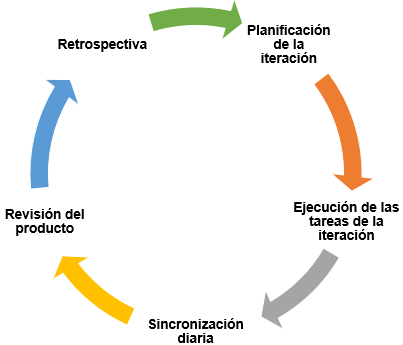
En este ciclo se seleccionan los Sprint (requisitos), se determinan los costos y los recursos para la construcción del producto, se hace un análisis general de lo que se debe entregar como producto final y se realiza un diseño del prototipo de este.

* + 1. ***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 1.

**Figura 1**

*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***: |
| 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? |
| 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. |
| 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. |
| 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. |

* + 1. ***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:

|  |
| --- |
| Gráfico interactivo  CF008\_1.2.3\_Post game |

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

* 1. **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:

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

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

|  |
| --- |
| Gráfico interactivo  CF008\_1.3\_Metodologia SUM |

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 productoyes 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.

1. **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. | Unity Cube logo |

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:

|  |
| --- |
| Gráfico interactivo  CF008\_2\_POO |

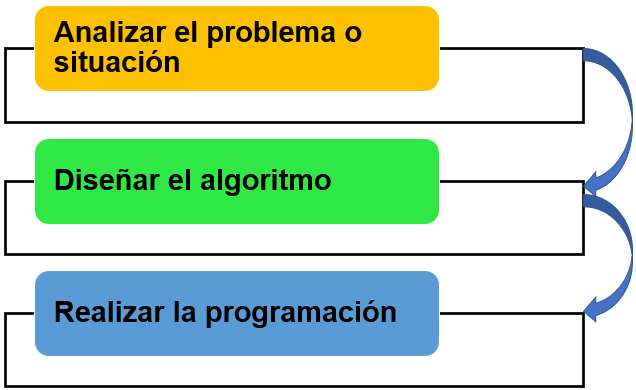
El lenguaje de programación que se utiliza en Unity para generar las líneas de código o *Scripts* es C#; el cual se compone de variables, funciones y clases que se explicarán más adelante. Para escribir el código, Unity tiene un editor de texto asociado llamado Visual Studio, aunque el programador puede 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:

|  |  |
| --- | --- |
| Consultar en la base de datos SENA, ingresando su usuario y contraseña ⇒ | Fundamentos de programación: algoritmos, estructura de datos y objetos  Joyanes Aguilar, L. (2020).  Propiedades fundamentales de orientación a objetos. *Fundamentos de programación: algoritmos, estructura de datos y objetos* (pp. 576-590). *McGraw-Hill.* <https://www-ebooks7-24-com.bdigital.sena.edu.co/?il=10409> |

* 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:



* **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 ejecutas por un ordenador del cual se obtienen los resultados esperados.

* + 1. ***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:

* **Preciso**: tener un objetivo claramente definido.
* **Secuencial**: llevar a cabo una serie de pasos en secuencia y siguiendo un orden lógico.
* **Finito**: poseer un número delimitado de pasos.
* **Determinado**: obtener un mismo resultado no importa cuántas veces se ejecute.
* **Llegar a un resultado**: solucionar el problema detectado o resolver la tarea que se ha planteado.
* **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.

* + 1. ***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 muestra a continuación:

|  |
| --- |
| Gráfico interactivo  CF008\_2.1.2\_ Ejecucion programa |

* 1. **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 ← 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 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 parte 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:

* Se deben elaborar en orden lógico que empieza arriba y termina abajo.
* Inician y finalizan con el mismo símbolo.
* Las flechas que se coloquen deben indicar el flujo o secuencia de las acciones a realizar.
* 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.**
* Lo que se escriba al interior de los símbolos debe tener el menor número de palabras posibles o contener operaciones claras: Z ← A – B
* El diagrama debe probarse con datos reales para verificar su funcionamiento. A esta verificación se le llama “prueba de escritorio”.

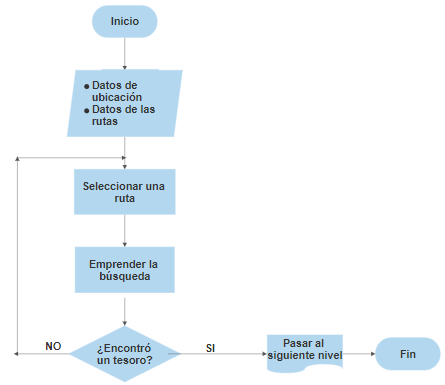
Las ventajas que tiene elaborar un algoritmo utilizando esta técnica, son las siguientes:

* Facilita la visualización general del problema, situación o tarea a resolver.
* Identifica con facilidad si los pasos que se han diseñado para la solución del problema tienen la secuencia lógica para lograr una solución adecuada.
* Permite verificar con datos reales si el diseño de las instrucciones logra la solución planteada.
* 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, ver figura 2, se ilustra la actividad que se le ha propuesto llevar a cabo al jugador.

**Figura 2**

*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!** | Botón Modal  Anexo: CF008\_2.2. a practicar |

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

|  |  |
| --- | --- |
| *Consultar en la base de datos SENA, ingresando su usuario y contraseña ⇒* | *Diseño y construcción de algoritmos*  *Mancilla Herrera, A., Capacho Portilla, J., Ebratt Gómez, R.(2016). Cap. 3. Primitivas algorítmicas. Diseño y construcción de algoritmos. Ediciones de la U.* [*https://www-ebooks7-24-com.bdigital.sena.edu.co/?il=6425*](https://www-ebooks7-24-com.bdigital.sena.edu.co/?il=6425) |

**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.

* + 1. ***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 si 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 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.
* 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#  bool A = false  int B = 80 | 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 viable se inicializó con “*false*”. La variable B es de tipo numérico que puede tomar valores enteros y se ha inicializado con el valor “80”. |

* + 1. ***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*** |

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**:

|  |
| --- |
| Pestañas A  Anexo: CF008\_2.3.2\_operadores |

* + 1. ***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):

* 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): \_\_\_\_\_\_\_\_\_**

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

|  |  |
| --- | --- |
| private int local (int A, int B)  {  Int total = A + B;  return total ;  } | 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*. |

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. |

* + 1. ***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  Linea = 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  Linea = 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 (consola) 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

* 1. **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:

* 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.
* 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:

* 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.
* Se mejora la experiencia del usuario ya que se tienen en cuanta 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.
* Se minimizan las líneas de código ya que es el usuario quien genera el evento y el sistema se encarga de responder.
* 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:

|  |
| --- |
| Pasos B  Anexo: CF008\_2.4\_crear evento |

* 1. **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:

|  |
| --- |
| Acordión A tipo A  Anexo: CF008\_2.5\_Estructura Script |

* 1. **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* quieres 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:

|  |  |
| --- | --- |
| * Clic en la opción de menú *GameObject*   + Clic en la opción UI     - Clic en la opción *Canvas* | *Canvas* |
| *Componentes* | 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. |

**Ejemplo**

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

|  |  |
| --- | --- |
| 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* | *Crear* |
| 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 | *Texto*  El área de *Canvas* está listo para adicionar otros componentes si se desea. |

Para profundizar en la ventana del Inspector del Canvas y en las temáticas de programación en Unity3D se recomienda consultar los siguientes sitios web:

1. **ACTIVIDADES DIDÁCTICAS (OPCIONALES SI SON SUGERIDAS)**

|  |  |
| --- | --- |
| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA | |
| Nombre de la Actividad | Responsabilidades Scrum según los roles |
| Objetivo de la actividad | Identificar las actividades que ejecuta cada rol en un equipo de trabajo SCRUM. |
| Tipo de actividad sugerida | Interfaz de usuario gráfica  Descripción generada automáticamente |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | CF008\_228108\_Actividad 1 |

1. **MATERIAL COMPLEMENTARIO:**

|  |  |  |  |
| --- | --- | --- | --- |
| Tema | Referencia APA del Material | Tipo de material  (Video, capítulo de libro, artículo, otro) | Enlace del Recurso o  Archivo del documento o material |
| Metodologías ágiles | RobertoTouza.com. (s.f). *Gestión Ágil de proyectos.* <https://robertotouza.com/agile/gestion-agil-de-proyectos/> | Artículo | <https://robertotouza.com/agile/gestion-agil-de-proyectos/> |
| Agilemanifiesto.org. (2001). *Manifiesto para el desarrollo de software ágil.* <http://agilemanifesto.org/> | Documento | <http://agilemanifesto.org/> |
| Metodología SCRUM | Trigas, M. (s.f). *Metodología Scrum. 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). *Introducción a SUM.* <http://www.gemserk.com/sum/> | Artículo | <http://www.gemserk.com/sum/> |
| Programación en Unity3D | Joyanes Aguilar, L. (2020).  Propiedades fundamentales de orientación a objetos. *Fundamentos de programación: algoritmos, estructura de datos y objetos* (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 construcción de algoritmos. Ediciones de la U. | Libro | <https://www-ebooks7-24-com.bdigital.sena.edu.co/?il=6425> |
| Estructuras de control | Microsoft. (2021). *Documentación de C#.* <https://docs.microsoft.com/en-us/dotnet/csharp/> | Manual | <https://docs.microsoft.com/en-us/dotnet/csharp/> |
| C#Sharp. Com.es. (2021). *Fundamentos del lenguaje C#.* <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). *Canvas: creación de interfaz de usuario en Unity.* <https://academiaandroid.com/canvas-creacion-de-interfaz-de-usuario-en-unity/> | Blog | <https://academiaandroid.com/canvas-creacion-de-interfaz-de-usuario-en-unity/> |
| Unity. (2018). *Documentation.* <https://docs.unity3d.com/es/2018.4/Manual/UnityManual.html> | Manual] | <https://docs.unity3d.com/es/2018.4/Manual/UnityManual.html> |
| López, B. (s.f). *Introducción a la programación en C#.* <http://www.itnuevolaredo.edu.mx/takeyas/Apuntes/Fundamentos%20de%20Programacion/Apuntes/03.-ElLenguajeCSharp.pdf> | PDF | <http://www.itnuevolaredo.edu.mx/takeyas/Apuntes/Fundamentos%20de%20Programacion/Apuntes/03.-ElLenguajeCSharp.pdf> |

1. **GLOSARIO**

|  |  |
| --- | --- |
| TÉRMINO | SIGNIFICADO |
| 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). |

1. **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>

Colombiaaprende.edu.co (s.f). *Lección 1: ¿Qué es un algoritmo?.* <https://aprende.colombiaaprende.edu.co/sites/default/files/naspublic/curriculos_ex/n1g10_fproy/nivel1/programacion/unidad1/leccion1.html>

Colombiaaprende.edu.co (s.f). *Lección 2: Los diagramas de flujo.* <https://aprende.colombiaaprende.edu.co/sites/default/files/naspublic/curriculos_ex/n1g10_fproy/nivel1/programacion/unidad1/leccion1.html>

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/>

1. **CONTROL DEL DOCUMENTO**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha |
| Autor (es) | Olga Lucía Mogollón Carvajal | Experto Temático | Regional Distrito Capital - Centro para la Industria de la Comunicación Gráfica | Septiembre 2021 |
| Luz Aída Quintero Velásquez | Diseñadora instruccional | Regional Distrito Capital - Centro de Gestión Industrial. | Septiembre 2021 |
| Ana Catalina Córdoba Sus | Revisora metodológica y pedagógica | Regional Distrito Capital – Centro para la Industria de la Comunicación Gráfica. | Octubre 2021 |
| Rafael Neftalí Lizcano Reyes | Asesor pedagógico | Regional Santander - Centro Industrial del Diseño y la Manufactura. | Octubre 2021 |
|  | Jhon Jairo Rodríguez Pérez | Diseñador y evaluador instruccional | Regional Distrito Capital - Centro para la Industria de la Comunicación Gráfica | Octubre 2021 |

1. **CONTROL DE CAMBIOS**

**(Diligenciar únicamente si realiza ajustes a la Unidad Temática)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
| Autor (es) |  |  |  |  |  |