**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

|  |  |
| --- | --- |
| PROGRAMA DE FORMACIÓN | PROGRAMACIÓN DE VIDEOJUEGOS |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA | 220501127 - Elaborar la lógica del contenido digital de acuerdo con métodos de desarrollo y propuesta de diseño. | RESULTADOS DE APRENDIZAJE | 220501127-03. Construir un prototipo funcional del videojuego de acuerdo con requerimientos del proyecto. |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 05 |
| NOMBRE DEL COMPONENTE FORMATIVO | Programación de videojuegos |
| BREVE DESCRIPCIÓN | Durante este componente, el aprendiz tendrá la oportunidad de desarrollar un prototipo de videojuego. El proceso comienza con la comprensión de la metodología de trabajo en un entorno real, utilizando métodos ágiles como Scrum. Posteriormente, el aprendiz adquirirá una comprensión de los conceptos fundamentales de la programación, lo que le permitirá, utilizando Unity como caso de estudio, implementar el prototipo del videojuego |
| PALABRAS CLAVE | Scrum, Algoritmos, Programación, Prototipo Funcional, Programación Orientada a Objetos |

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

|  |
| --- |
| **Para este componente se unen los componentes formativos y se agregan temas.**  Programación orientada a objetos  <https://sena.territorio.la/content/index.php/institucion/Titulada/institution/SENA/CienciasNaturales/228108/Contenido/OVA/CF8/index.html#/>  Integración de assets  <https://sena.territorio.la/content/index.php/institucion/Titulada/institution/SENA/CienciasNaturales/228108/Contenido/OVA/CF4/index.html#/> |

1. **TABLA DE CONTENIDOS:**

**Temas en azul son nuevos**

1. Metodologías de desarrollo

1.1 Metodologías ágiles

1.2 Metodología SCRUM

1.3 Metodología SUM

2. Programación en Unity3D

2.1 Conceptos básicos de algoritmo y programa

2.2 Diagramas de flujo

2.3 Conceptos de variables, operadores, funciones, métodos, 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)

2.7 Arreglos

2.8 Listas

3. Programación Orientada a Objetos

3.1 Clases

3.2 Atributos y cualificadores

3.3 Métodos

4. Matemáticas para Videojuegos

4.1 Vectores

4.2 Matrices

4.3 Operaciones sobre matrices

5. Interfaz y flujo de integración de Unity 3D

5.1 Instalación de Unity 3D

5.2 Navegación en Unity 3D

5.3 Escenario y ventanas de Unity 3D

6. Unity 3D importar elementos 3D - assets

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

8. Iluminación

8.1 Iluminación de escenarios

8.2 Luces en Unity 3D

9. Cámaras

10. Efectos visuales

11. Interfaz

11.1 Interfaz del videojuego

11.2 Diseño de interfaz del videojuego

1. **INTRODUCCIÓN**

**Dejar video:**

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

<https://www.youtube.com/watch?v=TE0TKx9dEtI>

1. **DESARROLLO DE CONTENIDOS**

|  |
| --- |
| **Se reúsa;** Programación orientada a objetos  <https://sena.territorio.la/content/index.php/institucion/Titulada/institution/SENA/CienciasNaturales/228108/Contenido/OVA/CF8/index.html#/> |

**Agregar información de los nuevos numerales:**

**2.7 ARREGLOS**

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

**Arreglos y matrices en c#**

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

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

|  |
| --- |
| **Ejemplos**  string[] arreglo = new string[8]; // Se declara un arreglo de string de 8 posiciones.  int[] numeros = { 2, 14, 17 }; // Se inicializa una matriz de enteros con 3 elementos.  string[] personajes = new string[] { "Paco", "Juan", "Mario", "Sabino" }; // Se inicializa una matriz de cadenas con 4 elementos. |

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Por ejemplo**  int [ ] números = { 2, 14, 17, 18, 15, 6 } ;   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **2** | **14** | **17** | **18** | **15** | **6** | | **0** | **1** | **2** | **3** | **4** | **5** |   En la posición 0 tenemos almacenado el valor de 2, mientras en la posición 5 tenemos el número 6, siendo la posición 0 el primer elemento del arreglo y la posición 5 el sexto elemento de este. |

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

Para el mismo ejemplo anterior:

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

**Tamaño de un arreglo.**

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

int x = numeros.Length

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

**Arreglos e iteraciones**

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

|  |
| --- |
| **Ciclo For**  for (int i = 1; i <= 10; i++)  {  Debug.Log(números[ i ] );  }  **Ciclo Foreach**  string[ ] ciudades= { "Pereira", "Armenia", "Bogotá", "Medellín", "Cali", "Barranquilla" };  foreach (string ciudad in ciudades ) {  Debug.Log(ciudad);  }  **Ciclo While**  string[ ] ciudades= { "Pereira", "Armenia", "Bogotá", "Medellín", "Cali", "Barranquilla" };  int i=0;  while (i < ciudades.Length) {  Debug.Log(ciudades[i]);  i++;  } |

**2.8 Listas**

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

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

**Crear listas en Unity**

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

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

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

**Por ejemplo:**

|  |
| --- |
| List<int> valores=new List<int>{4,1,3,7};  Igualmente se pueden agregar nuevos elementos a través del método Add  valores.Add(20) //agrega un nuevo elemento a la lista  Las listas igualmente se pueden adicionar una a otra a través del método AddRange  List<int> valores=new List<int>{4,1,3,7};  List<int> otrosvalores=new List<int>{5,9,2,8};  void Start()  {valores.AddRange(otrosvalores); //añade todos los elementos de la lista de otrosvalores a valores  } |

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

|  |
| --- |
| Slides  CF05\_2.8\_Listas |

**3. Programación Orientada a Objetos**

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

**3.1 Clases**

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



**Clases en Unity C#**

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

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

**¿De qué está compuesta una clase?**

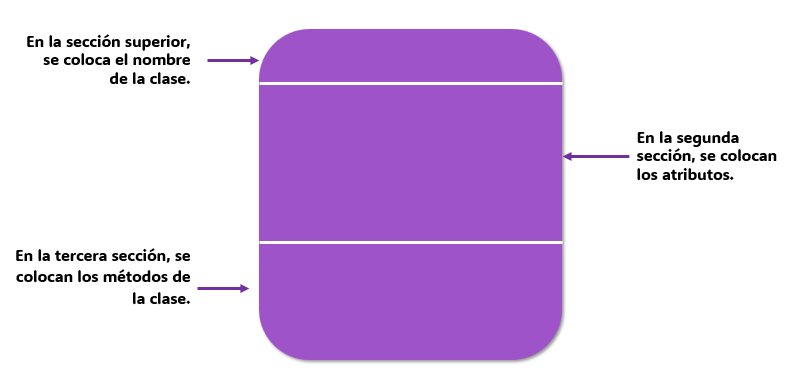
Las clases están compuestas por atributos y métodos:

**¿Cómo se representa una clase?**

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

**Figura 3.**

*Representación de una clase*



A continuación, se presenta un ejemplo:

|  |
| --- |
| **Ejemplo:**    En el ejemplo, "Enemigo" sería el nombre de la clase, con sus respectivos atributos (nombre, stamina, velocidad, ataque y defensa). Esta clase cuenta con métodos que identifican las acciones que el enemigo puede realizar, como son atacar, defender y perseguir.  Una forma sencilla de identificar qué es una clase, qué es un atributo y qué es un método es a través de su representación lingüística: generalmente, una clase se asocia con un sustantivo, los atributos con cualidades y los métodos con verbos. |

**3.2 Atributos y cualificadores**

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

|  |
| --- |
| Tarjetas  CF05\_3.2\_Atributos y cualificadores |

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

**3.3 Métodos**

Un método es similar a una función en la programación orientada a objetos (POO). Así como un sustantivo suele ser el indicativo de una clase, y una variable es el equivalente a un atributo dentro de esa clase, los métodos representan las acciones o comportamientos que la clase puede realizar. Un método puede aceptar parámetros como argumentos, procesarlos y, posteriormente, retornar un resultado cuando es invocado en una instancia de la clase. Aunque son similares a las funciones, los métodos se distinguen porque están asociados a una clase y frecuentemente operan sobre los atributos de dicha clase, modificando o utilizando su estado para realizar tareas.

**Métodos de interfaz**

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

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

Por ejemplo:

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

**Método constructor**

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

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

**Método de implementación**

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

|  |
| --- |
| class TestMotorcycle : Motorcycle  {  public override double GetTopSpeed()  {  return 108.4;  }  }  static void Main()  {  TestMotorcycle moto = new TestMotorcycle();  moto.StartEngine();  moto.AddGas(15);  moto.Drive(5, 20);  double speed = moto.GetTopSpeed();  Console.WriteLine("My top speed is {0}", speed);  } |

**4. Matemáticas para videojuegos**

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



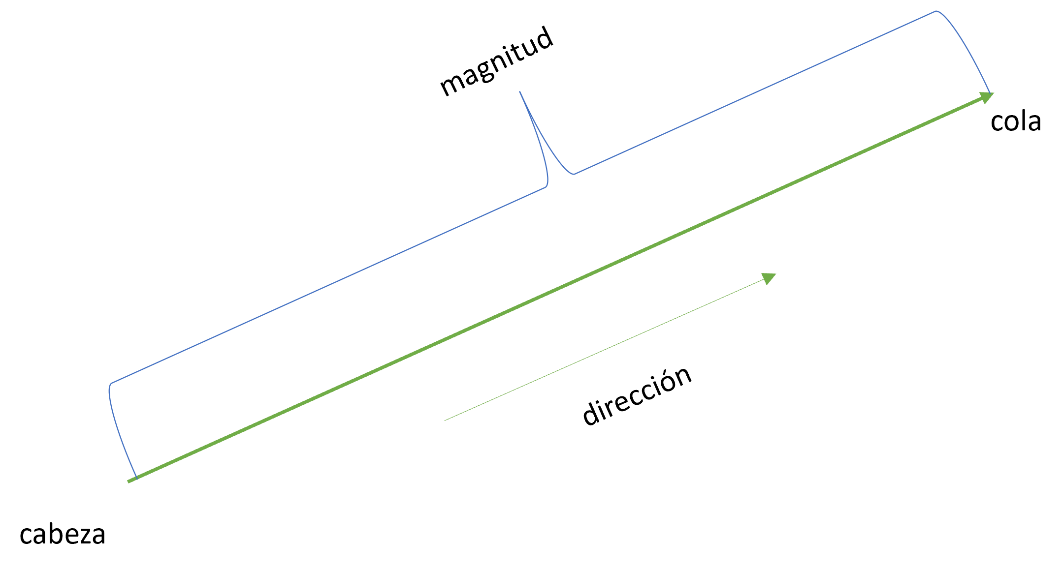
Por ejemplo, en los videojuegos 3D, el álgebra lineal es esencial para manejar las transformaciones geométricas y la proyección de escenas en 3D a la pantalla 2D. Se utilizan matrices y vectores para rotar, escalar y mover objetos en el espacio del juego.

**4.1 Vectores**

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

**Figura 4**

*Vector.*

****

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

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

Los vectores, representados gráficamente, poseen las siguientes características:

|  |
| --- |
| Infografía interactiva  CF05\_4.1\_Vectores |

**4.2 Matrices**

Una matriz es una disposición rectangular compuesta por números, variables, símbolos o expresiones, organizados en filas y columnas. Esta estructura permite efectuar diversas operaciones matemáticas como la suma, resta, multiplicación y transposición. Cada unidad de la matriz se conoce como “**elemento**”. Las secuencias de elementos en **horizontal constituyen las filas y en vertical, las columnas**. El tamaño u orden de una matriz se especifica por el número de filas y columnas que posee, denotándose como 'm x n', donde 'm' representa el número de filas y 'n' el de columnas. Por ejemplo, una matriz B de 3 filas y 4 columnas se escribe como B3×4 y se denomina matriz de tres por cuatro.

**Notación de matrices**

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

**4.3** **Operaciones sobre matrices**

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

**Adición de matrices**

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

**Figura 5.**

*Ejemplo suma de matrices.*

Diagrama, Esquemático

Descripción generada automáticamente

**Resta de matrices**

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

**Figura 6.**

*Ejemplo resta de matrices.*

Diagrama

Descripción generada automáticamente con confianza baja

**Multiplicación escalar**

El producto de una matriz A con cualquier número 'c' se obtiene multiplicando cada entrada de la matriz A por c, se llama multiplicación escalar, es decir, (cA)ij=c(Aij)

**Propiedades de la multiplicación escalar en matrices.**

Las diferentes propiedades de las matrices para la multiplicación escalar de cualquier escalar K y l, con las matrices A y B se dan como:

K(A + B) = KA + KB

(K + l)A = KA + lA

(Kl)A = K(lA) = l(KA)

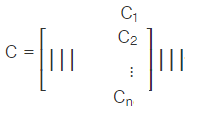
(-K)A = -(KA) = K(-A)

1·A = A

(-1)A = -A

**Multiplicación de matrices**

La multiplicación de matrices se define solo si el número de columnas de la primera matriz y de filas de la segunda matriz son iguales. Para entender cómo se multiplican las matrices, consideremos primero un vector fila R = [r1 r2 ... rn] y un vector de columna:



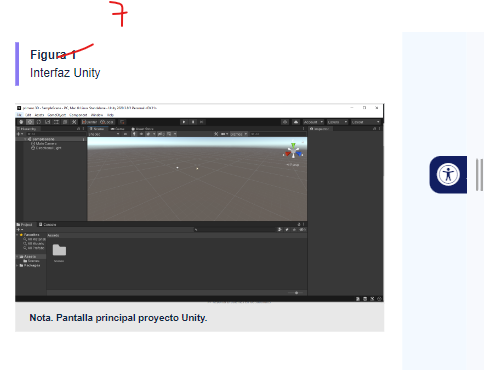
Entonces el producto de R y C se puede definir como:

Imagen que contiene Texto

Descripción generada automáticamente

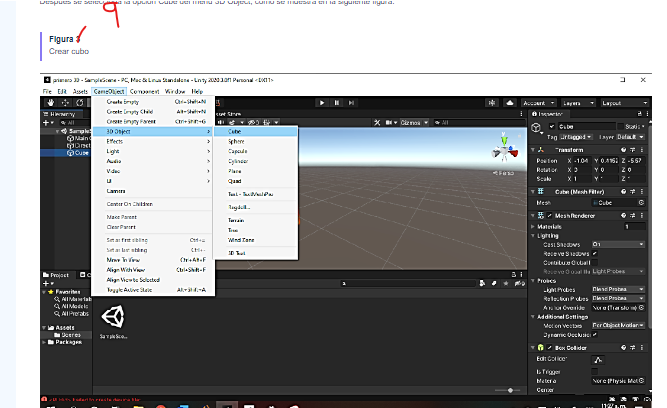
|  |
| --- |
| **Se reúsa, se reenumera y se ajustan el número de figuras y tablas;**  Integración de assets  <https://sena.territorio.la/content/index.php/institucion/Titulada/institution/SENA/CienciasNaturales/228108/Contenido/OVA/CF4/index.html#/>  5. Interfaz y flujo de integración de Unity 3D  5.1 Instalación de Unity 3D  5.2 Navegación en Unity 3D  5.3 Escenario y ventanas de Unity 3D  6. Unity 3D importar elementos 3D - assets  7. Componer los escenarios (personajes, props, fondos)  8. Iluminación  8.1 Iluminación de escenarios  8.2 Luces en Unity 3D  9. Cámaras  10. Efectos visuales  11. Interfaz  11.1 Interfaz del videojuego  11.2 Diseño de interfaz del videojuego |

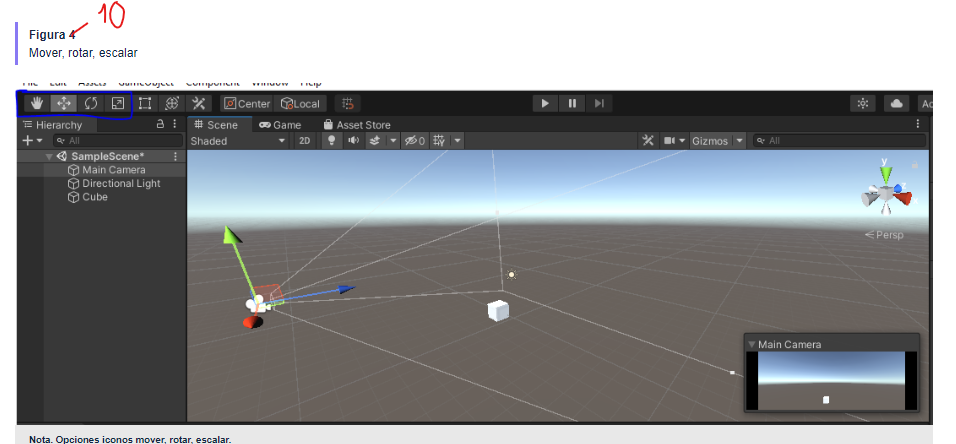
**REMUNERACIÓN DE FIGURAS:**



A screenshot of a computer

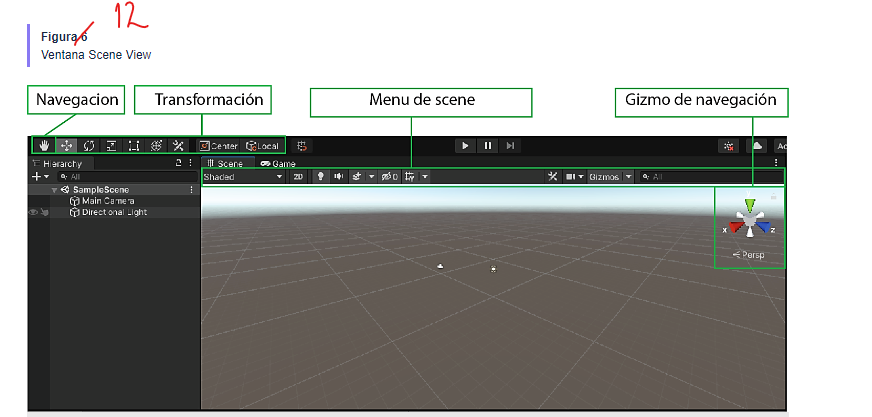
Description automatically generated

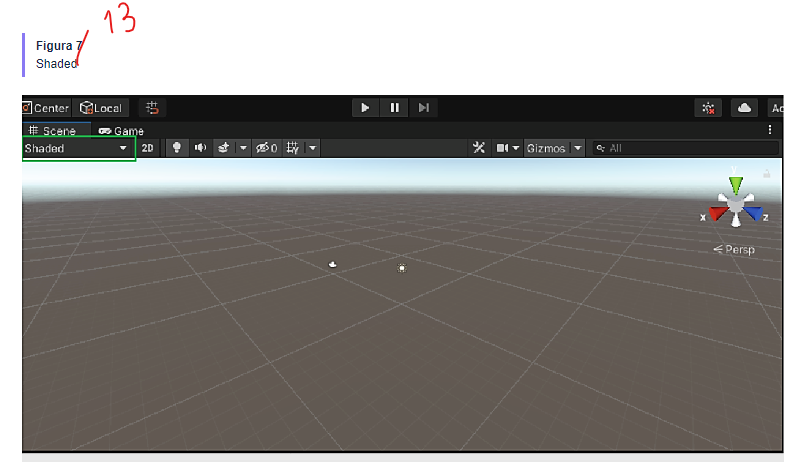


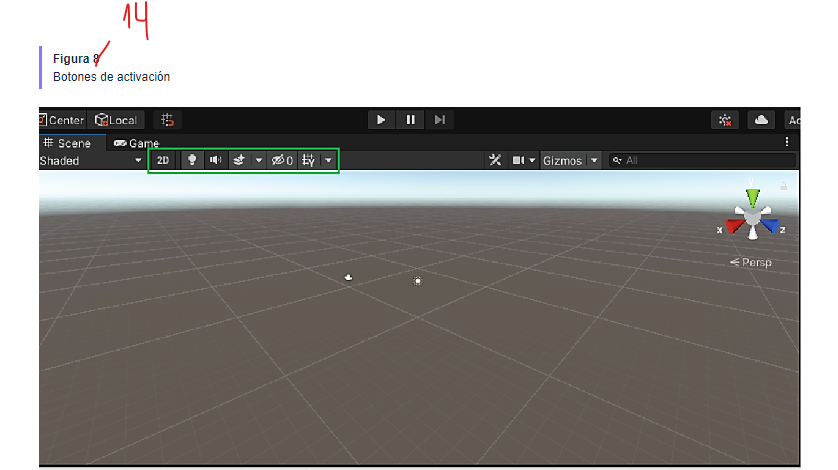


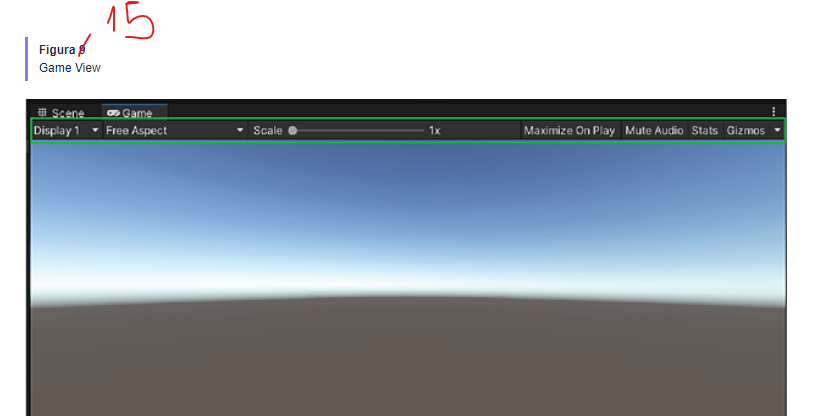
A screenshot of a computer

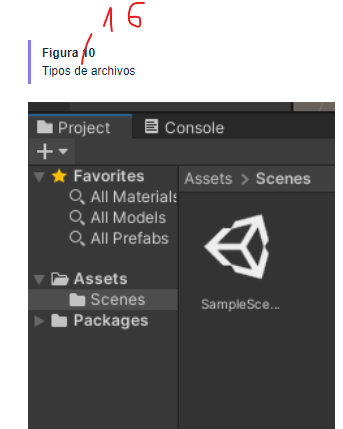
Description automatically generated

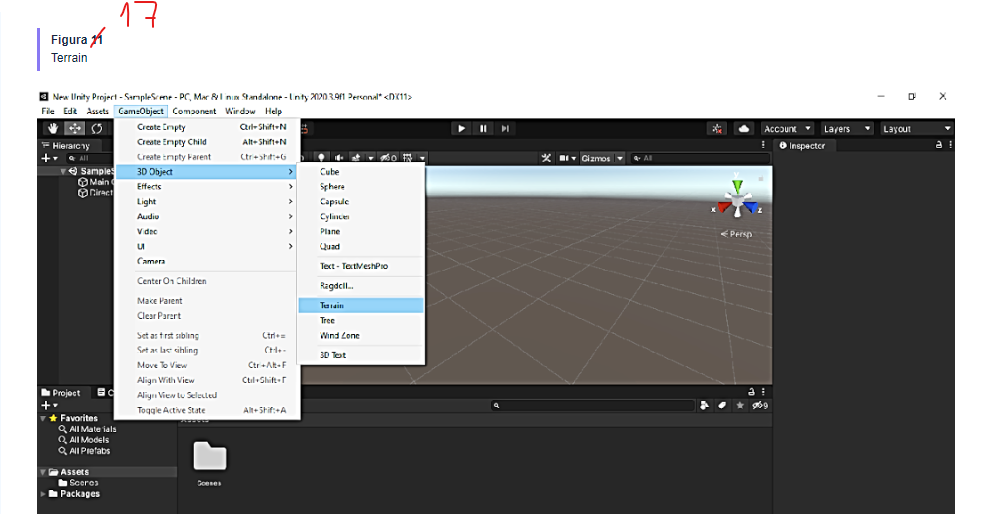


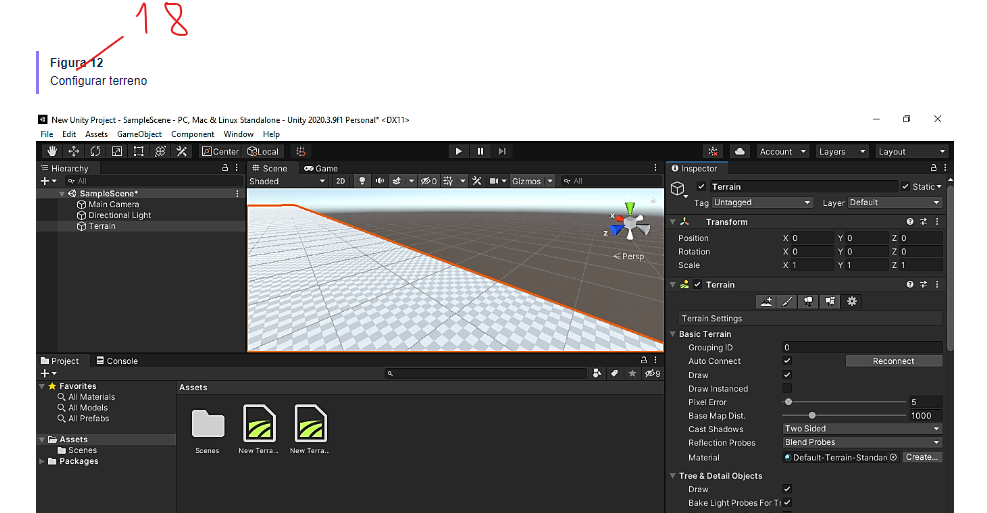


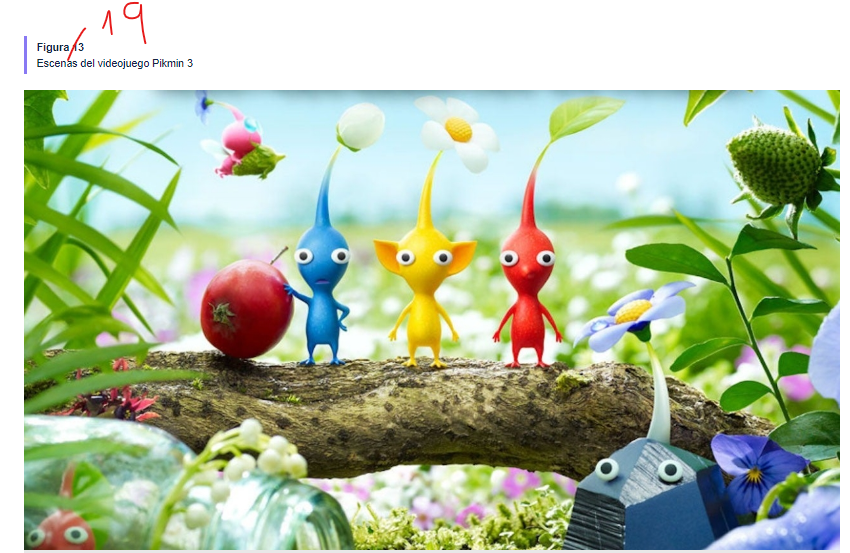






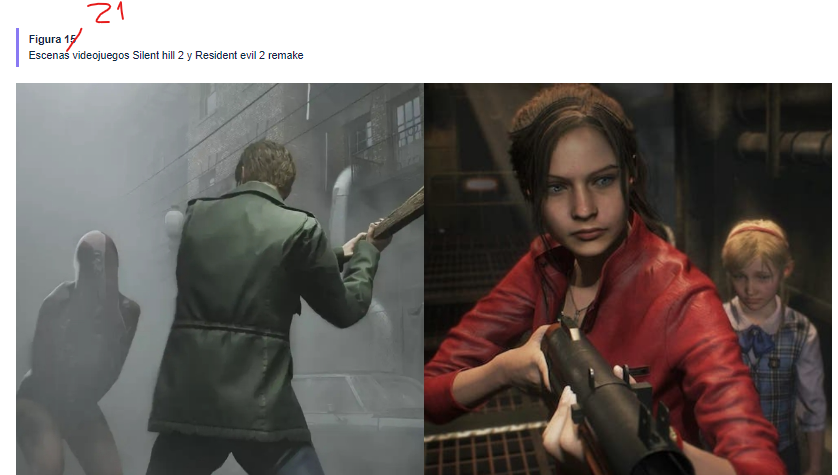


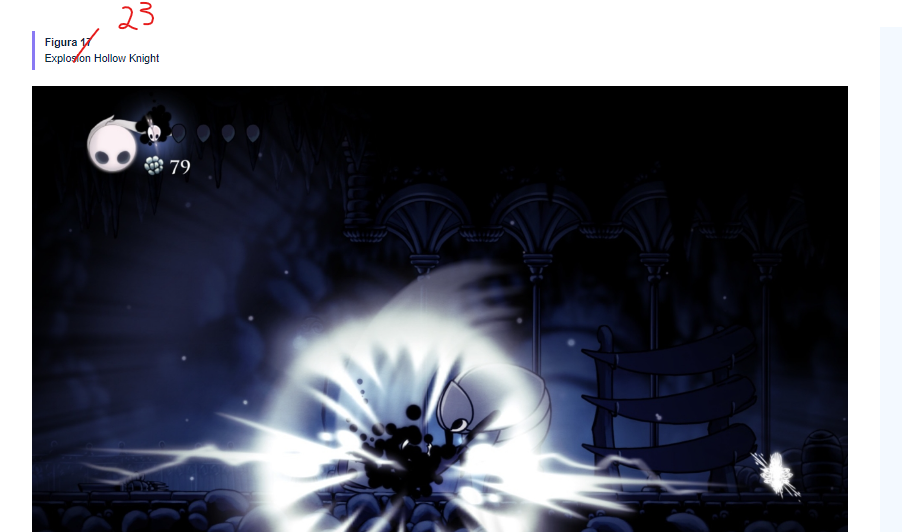
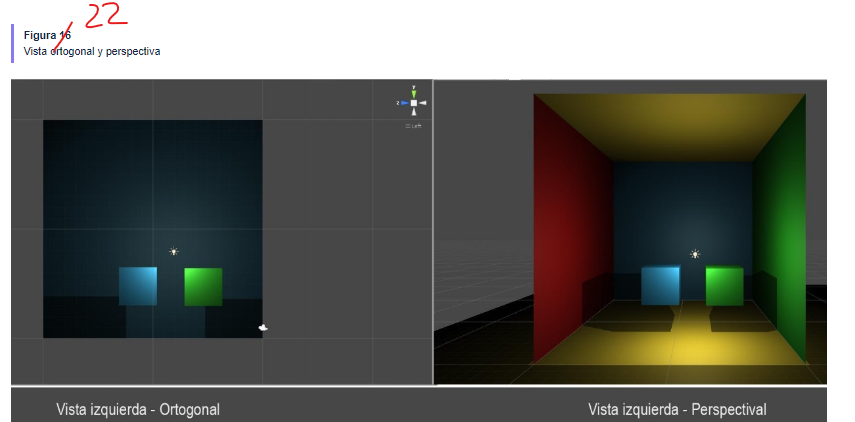




A screenshot of a video game

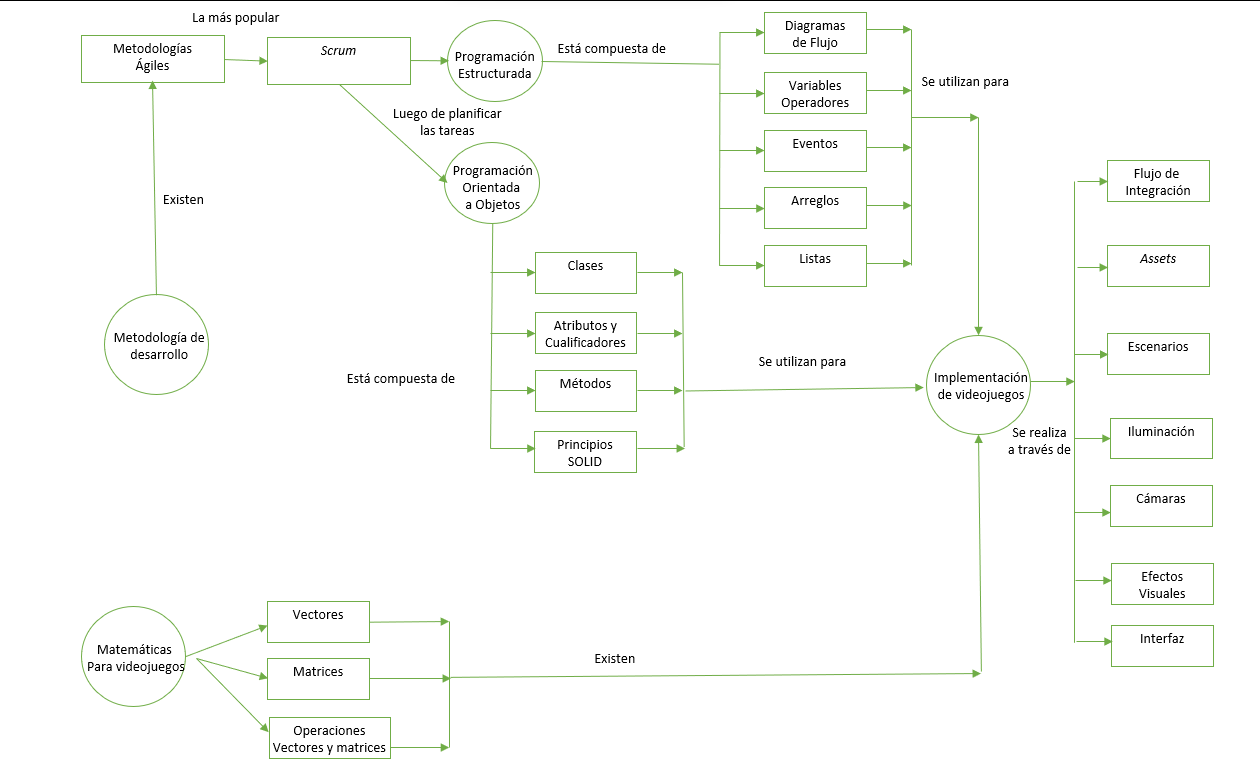
Description automatically generated





1. **SÍNTESIS**

Crear nueva síntesis (adjunta editable)

****

1. **ACTIVIDADES DIDÁCTICAS**

Utilizar las dos actividades de los componentes reusados

1. **MATERIAL COMPLEMENTARIO:**

**Añadir recursos**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tema** | **Referencia APA del Material** | **Tipo de material**  **(Video, capítulo de libro, artículo, otro)** | **Enlace del Recurso o**  **Archivo del documento o material** |
| 1.2 Metodología SCRUM | Mejorar La Productividad (2022, marzo 24). SCRUM + KANBAN [metodología ágil] / EJEMPLO /. Youtube. | Video | <https://www.youtube.com/watch?v=6ZBIE0XJU1M> |
| 3. Programación Orientada a Objetos | Cómo te explico? (2021). Programación estructurada. Youtube. | Video | <https://www.youtube.com/playlist?list=PLHxyqMuBkJ3yyIwbNSt1GuqLFccq2YYTx> |
| 3. Programación Orientada a Objetos | yacklyon (2019, septiembre 24). CURSO de PYTHON 2020 Programación Orientada a Objetos (POO). Youtube. | Video | <https://www.youtube.com/watch?v=AqjTfmPitTQ&list=PLg9145ptuAigw5pV_DRznXdOsX19dorDs> |
| 4.1 Vectores | Matematicas profe Alex. (2019). Vectores. Youtube. | Video | <https://www.youtube.com/playlist?list=PLeySRPnY35dEaZT3iJUNdDkgLEZE5x-Jd> |
| 4.2 Matrices | Matematicas profe Alex. (2019, mayo 31). Qué es una matriz | Sistemas de ecuaciones. Youtube. | Video | <https://www.youtube.com/watch?v=RJ96S2Pt3qU&list=PLeySRPnY35dEr2XewNdOjOl7Ft0tLIlKI> |
| Diagrama de Clases | Nicosiored. (2017, octubre 25). Introduccion a UML- 1- Tutorial UML en español. Youtube. | Video | <https://www.youtube.com/watch?v=-OWd0tJAK10&list=PLM-p96nOrGcaw5dhv8wOA5tVVWEmXtA2F> |
| 11.2 Diseño de interfaz del videojuego | Antony Morsas]. (2020, junio 26). Crea tu primer juego 2D en Unity desde cero. Youtube. | Video | <https://www.youtube.com/watch?v=4XvfpCz_vh8&t=755s> |

1. **GLOSARIO:**

|  |  |
| --- | --- |
| **TÉRMINO** | **SIGNIFICADO** |
| **Arreglos (Programación)** | son estructuras de datos que nos permiten almacenar otros datos dentro de este tipo de datos. Es decir, es un contenedor que nos permite tener varios datos al mismo tiempo almacenados. |
| **Atributo** | son las características individuales que diferencian un objeto de otro y determinan su apariencia, estado u otras cualidades. |
| **Caso de uso** | es la descripción de una acción o actividad. Un diagrama de caso de uso es una descripción de las actividades que deberá realizar alguien o algo para llevar a cabo algún proceso. |
| **Ciclo** | es una secuencia de instrucciones de código que se ejecuta repetidas veces, hasta que la condición asignada a dicho bucle deja de cumplirse. |
| **Clase** | una clase es un elemento de la programación orientada a objetos que actúa como una plantilla y va a definir las características y comportamientos de una entidad. |
| **Condicional** | los condicionales son estructuras que permiten elegir entre la ejecución de una acción u otra. |
| **Instrucción** | conjunto de instrucciones definidas, ordenadas y finitas que aplicadas permiten realizar una acción con el fin de resolver un problema o realizar una acción. |
| **Método** | es un conjunto de instrucciones definidas dentro de una clase, que realizan una determinada tarea y a las que podemos invocar mediante un nombre. |
| **Programación Orientada a Objetos** | la programación orientada a objetos se basa en el concepto de crear un modelo del problema de destino en sus programas. |
| **Variable** | sirven para guardar y recuperar datos, representar valores existentes y asignar unos nuevos. |

1. **REFERENCIAS BIBLIOGRÁFICAS:**

Trigas, M. (2012). *Gestión de proyectos informáticos.* Universidad Oberta de Catalunya. <https://openaccess.uoc.edu/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf>

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

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

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

1. **CONTROL DEL DOCUMENTO**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Nombre** | **Cargo** | **Dependencia** | **Fecha** |
| **Autor (es)** |  |  |  |  |

1. **CONTROL DE CAMBIOS**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Nombre** | **Cargo** | **Dependencia** | **Fecha** | **Razón del Cambio** |
| **Autor (es)** | Carlos Andrés Cortes | Experto temático | Centro de Diseño e Innovación Tecnológica Industrial- Regional Risaralda | Noviembre 2023 | Actualización |
|  | Paola Alexandra Moya Peralta | Diseñadora instruccional | Centro Industrial del Diseño y la Manufactura - Regional Santander | Noviembre 2023 | Actualización |
|  | Rafael Neftalí Lizcano Reyes | Responsable de Línea de Producción | Centro Industrial del Diseño y la Manufactura - Regional Santander | Noviembre 2023 | Actualización |