Universidad Politécnica de Cartagena

Escuela Técnica Superior de Ingeniería de Telecomunicación

TRABAJO FIN DE GRADO

Aplicación de laboratorio virtual de experimentos de física para estudiantes

Autor: Casas Delgado, José

Director: Egea López, Esteban

Cartagena, septiembre de 2021

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

El proyecto realizado pretende proporcionar a los estudiantes de física, más concretamente para estudiantes de la ESO, bachillerato e incluso primeros cursos de grados universitarios, una herramienta a través de la cual puedan probar y visualizar con detalle experimentos de física básica, con el objetivo de facilitar el aprendizaje y entendimiento de la cinemática y la dinámica.

Con la aplicación, los estudiantes pueden seleccionar diferentes escenarios de simulación en 3D y, mediante la configuración de diferentes parámetros de los cuerpos sólidos que intervienen, experimentar los efectos que se producen como cambios de trayectorias o variación de magnitudes y, conseguir así, una mejor comprensión de los problemas. Cabe mencionar también que, aunque muchos escenarios son elementales, se pueden añadir configuraciones que hagan que la solución no se pueda obtener de forma analítica, por lo que la herramienta permite avanzar en el estudio más allá de los conceptos básicos.

La aplicación consta de cinco escenarios de simulación, los cuales son: tiro parabólico, plano inclinado, ley de Hooke, fuerza de tensión y colisiones elásticas e inelásticas.

El desarrollo del programa se ha realizado sobre el motor de videojuegos Unity3D. El motivo por el que se ha decidido utilizar esta herramienta es porque trae integrado un motor de físicas ya definido, el cual es NVIDIA PhysX. Esto permite que toda la parte de la física que afecta a los cuerpos sólidos como la gravedad, las colisiones, el rozamiento y otras fuerzas, sean tratadas de forma automática por la propia herramienta de Unity, por ello, es ideal para desarrollar de manera sencilla el proyecto planteado. Además, Unity cuenta con una serie de modelos 3D primitivos por defecto, que pueden ser escalados y rotados, que también se han utilizado para la aplicación por su sencillez. Los modelos en concreto son el cubo, la esfera, la cápsula, el cilindro, el plano y el cuadrante.

2. TECNOLOGÍAS UTILIZADAS

2.1. UNITY

Unity es un motor de videojuegos creado por la compañía Unity Technologies y que tuvo su lanzamiento en 2005. Está disponible como plataforma de desarrollo para sistemas operativos Windows, Mac y Linux.[1]

Figura 2.1: *Logotipo de Unity*

El motor de desarrollo de Unity está totalmente integrado y rico en funciones para la creación de contenido interactivo en 2D y 3D. Proporciona una funcionalidad completa y lista para usar y ensamblar contenido de alta calidad y rendimiento y publicarlo en múltiples plataformas. Unity ayuda a los desarrolladores y diseñadores independientes, a los grandes y pequeños estudios, a las corporaciones multinacionales, a los estudiantes y a los aficionados a reducir drásticamente el tiempo, el esfuerzo y el costo de crear juegos.[2]

2.1.1. Física en Unity

El motor de físicas integrado de Unity permite que los objetos de la simulación sean controlados por una aproximación de las fuerzas que existen en el mundo real, de manera que dichos objetos se verán acelerados por la gravedad, responderán a colisiones y se verán afectados por cualquier otro tipo de fuerza que sea ejercida sobre los mismos. Este motor de físicas es el motor de NVIDIA PhysX, cuya tecnología está optimizada para la aceleración de la física por hardware mediante procesadores masivamente paralelos.[3]

A cada objeto que aparece en la escena del proyecto se le denomina en Unity como GameObject. Por defecto, un GameObject no puede hacer nada por sí mismo, por lo que requerirá de una serie de propiedades que definan su comportamiento. Estas propiedades se conocen en Unity como componentes.

---

[1] Wikipedia. Obtenido de https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)
[2] Mod DB. Obtenido de https://www.moddb.com/engines/unity
[3] Gomar, Juan. (09 de Octubre de 2018). Obtenido de https://www.profesionalreview.com/2018/10/09/que-es-nvidia-physx/

Unity no utiliza directamente la API de PhysX, sino que proporciona su propia API para el acceso a la funcionalidad. Los componentes que define Unity para simular la física de los objetos son: Colliders, Rigidbodies, Physics Materials, Joints y Scripts.

2.1.1.1. Collider

El componente Collider se puede describir como un volumen no visible que se asocia a un objeto y define el espacio físico que ocupa dicho objeto, de manera que, cuando dos o más Colliders colisionan o se cruzan entre sí queda registrado. Para que se produzcan las colisiones, los objetos deben tener, además, un componente Rigidbody asociado.

Dado que el objetivo de este proyecto es crear una herramienta para simular experimentos de física, se han utilizado Colliders que se ajustan lo máximo posible a los objetos 3D que se visualizan, para así conseguir una aproximación lo más cercano posible a la realidad.

Por defecto, Unity incluye algunos tipos de Colliders:

- **Box Collider:** colisionador primitivo con forma de cubo. Las caras del cubo pueden ser escaladas para formar el volumen de un prisma cuadrangular.

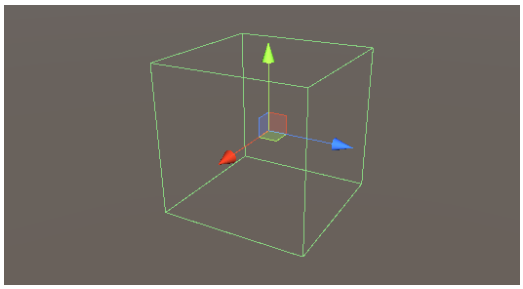

Figura 2.2: *Box Collider*

- **Sphere Collider:** colisionador primitivo con forma de esfera. El radio de la esfera puede ser escalado.

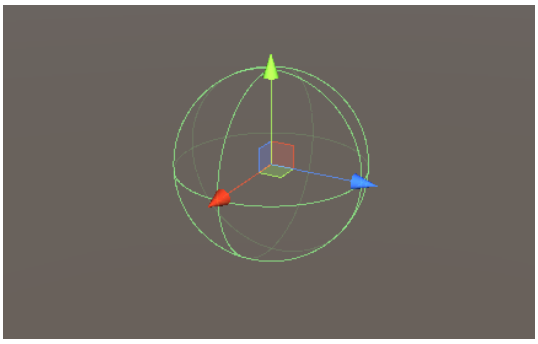

Figura 2.3: *Sphere Collider*

- **Capsule Collider:** colisionador primitivo con forma de cápsula. Está formado por dos semiesferas unidas por un cilindro. Pueden ser modificados tanto el radio de las semiesferas como la altura del cilindro.

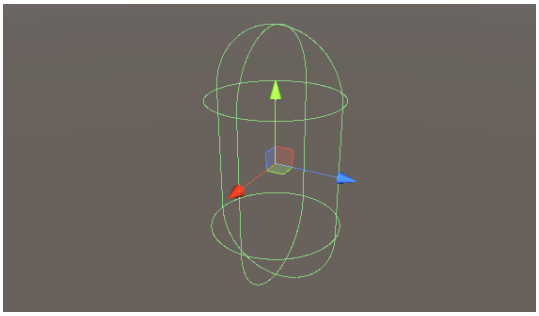

Figura 2.4: Capsule Collider

- **Mesh Collider:** colisionador con forma de malla que se adapta a la forma del modelo 3D del objeto al que se le quiera asignar. Está formado por triángulos, lo que permite que se puedan formar volúmenes con formas más complicadas y detalladas. Sin embargo, este tipo de colisionador se convierte en una opción computacionalmente más costosa que los colisionadores primitivos a medida que contengan más triángulos.

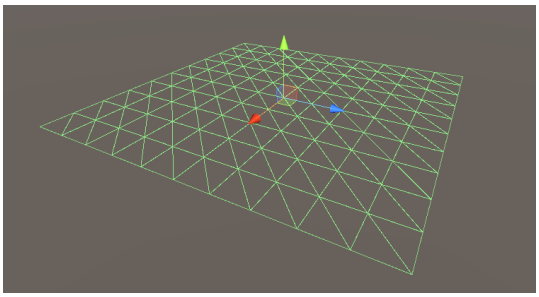

Figura 2.5: Mesh Collider

2.1.1.2. Rigidbody

El componente Rigidbody representa un sólido rígido, lo que permite que los objetos sean afectados por propiedades físicas, como la gravedad. Entre los atributos del Rigidbody se encuentran la masa del objeto, el vector de velocidad, la resistencia al aire o el arrastre angular (el cual afecta a la velocidad angular), entre otros.

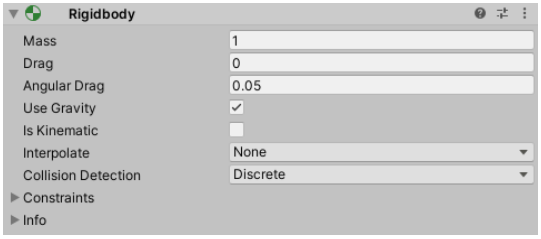

Figura 2.6: Características del componente Rigidbody

2.1.1.3. Physic Material

Unity permite la creación de materiales físicos que pueden ser asociados como atributo del componente Collider de un objeto y, con los cuales, se puede ajustar el rozamiento que se produce en las superficies de los objetos cuando entran en contacto.

Un material físico queda definido principalmente por las características de rozamiento estático, rozamiento dinámico y capacidad para rebotar.

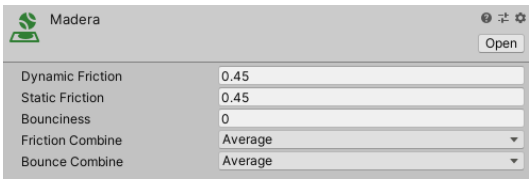

Figura 2.7: *Ejemplo de material físico*

2.1.1.4. Joint

El componente Joint permite conectar un Rigidbody a otro Rigidbody o a un punto fijo del espacio. Unity define algunos Joints por defecto y, en función de cual se utilice se aplicarán diferentes fuerzas entre las uniones y se limitará el movimiento para los cuerpos conectados.

- **Fixed Joint :** restringe el movimiento de un objeto, de manera estricta, al movimiento del objeto o punto del espacio al que esté conectado.
- **Hinge Joint :** adjunta un objeto a otro permitiendo que los cuerpos giren alrededor de un eje específico desde un punto origen común.
- **Spring Joint :** mantiene los objetos unidos entre sí pero permite que la distancia entre ellos se estire ligeramente.

2.1.1.5. Script

Si se quiere obtener un comportamiento específico en los objetos, Unity permite la creación de scripts de programación que se pueden añadir como componentes de dichos objetos.

Para la programación de scripts se utiliza el lenguaje C#, y cada script crea su propia conexión con los procesos internos de Unity al implementar una clase que deriva de la clase integrada, llamada `MonoBehaviour`.[4] Esta clase implementa una serie de funciones que se ejecutan por eventos y en un orden estrictamente definido, como se muestra en la Figura 2.8.

[4] UNITY. Obtenido de https://unity.com/es/how-to/programming-unity

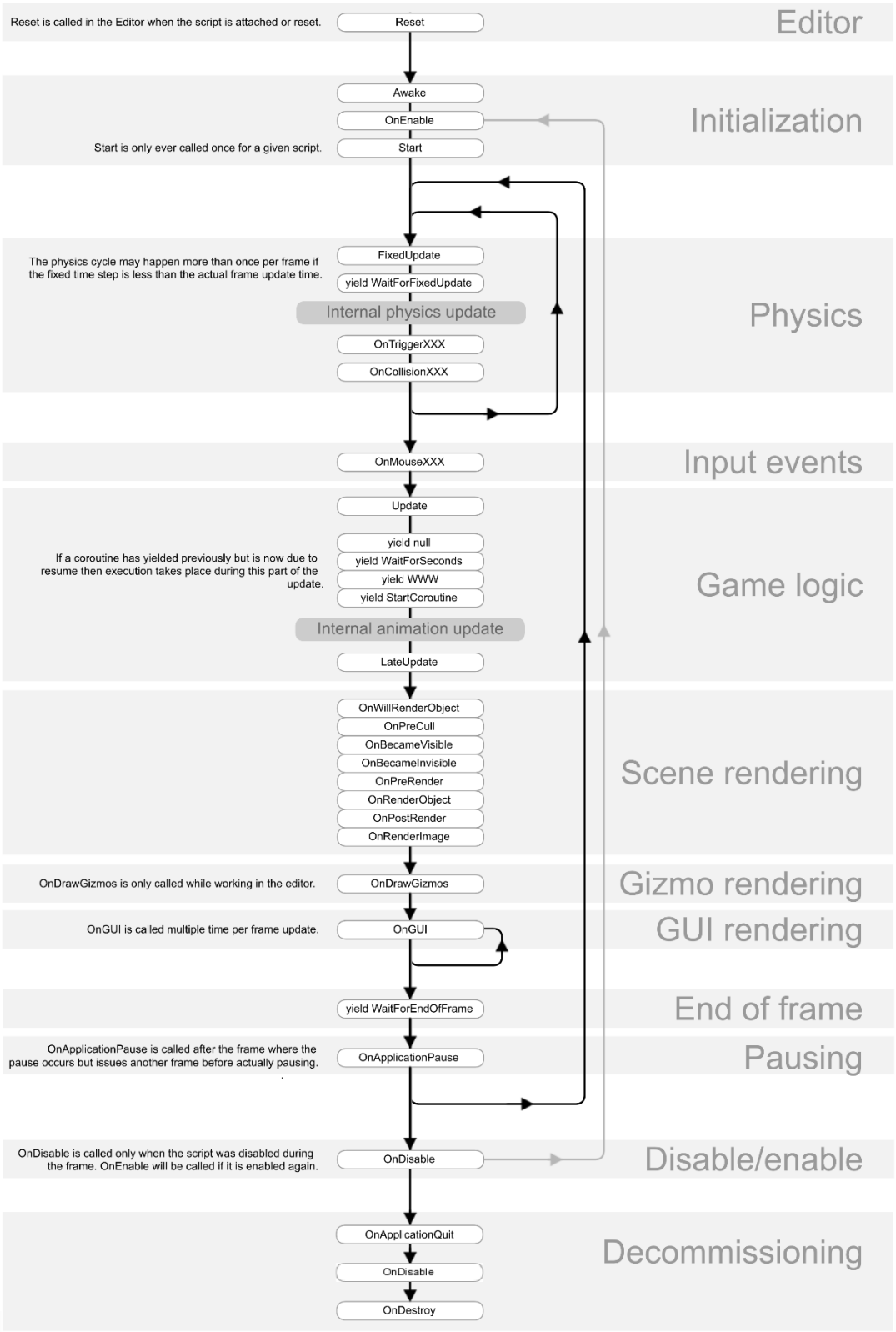

***Figura 2.8**: Jerarquía de ejecución de funciones*

Algunos métodos a destacar son:[5]

- **Awake:** se llama una sola vez cuando un GameObject activo que contiene el script se inicializa al cargar la escena, o cuando un GameObject previamente inactivo se establece como activo. Comúnmente se utiliza para inicializar variables o estados antes del inicio de la aplicación.
- **Start :** se llama justo antes de la primera llamada de los métodos de actualización. Al igual que la función Awake, se llama una sola vez.
- **FixedUpdate :** es llamado de forma periódica independientemente de la velocidad de fotogramas de la aplicación. El periodo que establece el número de llamadas por segundo puede ser modificado a través de la variable Time.fixedDeltaTime. Las actualizaciones del sistema físico ocurren después de cada llamada a FixedUpdate.
- **Update:** es la función más utilizada para implementar cualquier lógica dentro de un juego. Las llamadas a esta función se producen una vez por cada fotograma.

2.2. MICROSOFT VISUAL STUDIO

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE) para Windows y Mac OS. Es compatible con múltiples lenguajes de programación, entre los que se encuentra C#.[6]

Figura 2.9: *Logotipo de Microsoft Visual Studio*

Visual Studio Tools incluye una extensión que permite la integración de Visual Studio con el editor de Unity, de manera que se obtiene una herramienta completa con la que desarrollar aplicaciones y juegos multiplataforma. La herramienta proporciona características de edición de código y depuración para crear scripts de editor y juego para Unity mediante C#. Además, el componente IntelliSense hace que resulte rápido y sencillo implementar mensajes de la API de Unity.[7]

[5] Unity3D. Obtenido de https://docs.unity3d.com/Manual/index.html
[6] Wikipedia. Obtenido de https://es.wikipedia.org/wiki/Microsoft_Visual_Studio
[7] Microsoft. Obtenido de https://docs.microsoft.com/es-es/visualstudio/gamedev/unity/get-started/visual-studio-tools-for-unity

2.3. INKSCAPE

Para la creación de imágenes y elementos visuales que aparecen en la interfaz de usuario de la aplicación se ha utilizado el editor de gráficos vectoriales libre y de código abierto, Inkscape. Este programa permite crear y editar diagramas, líneas, gráficos, logotipos e ilustraciones complejas, utilizando el formato *Scalable Vector Graphics* (SVG). Se encuentra disponible para la plataforma Windows, Mac OS y otros derivados de Unix.[8]

Figura 2.10: *Logotipo de Inkscape*

[8] Wikipedia. Obtenido de https://es.wikipedia.org/wiki/Inkscape

3. DESARROLLO DEL PROYECTO

El proyecto que se ha desarrollado consiste en una aplicación de escritorio a través de la cual se puedan simular experimentos de cinemática y dinámica. Con la aplicación, se pueden visualizar tanto las variaciones de las magnitudes de los distintos parámetros de interés de los objetos que intervienen en la simulación como los cambios en la dirección, sentido y módulo de los vectores.

La aplicación consta de cinco escenarios de simulación, los cuales son: tiro parabólico, plano inclinado, ley de Hooke, fuerza de tensión y colisiones elásticas e inelásticas. Para cada una de las distintas simulaciones, la aplicación permite introducir datos de entrada, los cuales utiliza posteriormente el motor de físicas de Unity para llevar a cabo el movimiento de los objetos.

3.1. ESTRUCTURA DE SCRIPTS DE LA APLICACIÓN

En Unity, las escenas son elementos que contienen los objetos y componentes que forman un juego y son usadas para crear menús o niveles del juego, entre otros.

Para este proyecto se han creado un total de seis escenas, donde cada uno de los cinco escenarios de simulación desarrollados tienen su propia escena, y para el menú principal hay otra escena más que se utiliza como pasarela para cambiar entre los distintos escenarios de simulación. Las escenas de la aplicación poseen elementos que son comunes a todas y se comportan de la misma manera, de modo que los scripts de programación pueden ser reutilizados en todas las escenas. Sin embargo, existen otros muchos elementos que, ya sean o no comunes en todas las escenas, se comportan de forma distinta, por lo que necesitan un script específico para cada uno de dichos elementos.

En primer lugar, se enumeran aquellos scripts comunes a todas las escenas y que tienen siempre el mismo comportamiento:

- **ColorPickerPanel.cs :** controla el panel para la modificación de color para el material y el contorno de los objetos del juego.
- **LabelButton.cs :** actualiza el tamaño de las etiquetas del panel general, en función de la etiqueta que en ese momento esté seleccionada.
- **LabelsPanel.cs :** controla el estado de selección de cada una de las tres etiquetas del panel general.
- **OptionsPanel.cs :** controla la visualización del menú de opciones, así como del menú interno de ajustes de vídeo. Además, contiene las funciones para cerrar la aplicación y para cargar la escena del menú principal.
- **StepByStepPanel.cs :** controla la visualización del panel de simulación paso a paso en función del estado de la casilla de verificación que activa y desactiva dicho panel.
- **Vector.cs :** obtiene como atributos de entrada un cilindro, un cono y un punto de referencia para el cono. El script hace que el cono se posicione constantemente,

en la posición de referencia, la cual, a su vez, está posicionada en una de las bases del cilindro, de modo que se obtiene el modelo 3D de un vector. El motivo por el que se crea el vector de esta forma y no como una pieza única es porque de este modo se puede realizar el escalado del cilindro sin que se deforme el vector.

- **VideoSettings.cs:** se encarga de gestionar los ajustes gráficos en función de los valores que el usuario seleccione desde el menú de opciones.

Por otro lado, se enumeran los scripts que tienen ligeras variaciones para cada una de las escenas:

- **CameraBehaviour.cs:** controla el posicionamiento de la cámara y le aplica filtros en función de modo de cámara que esté seleccionado.
- **ClickObject.cs:** permite detectar si se ha hecho clic sobre un objeto del juego, marcándolo como seleccionado.
- **GeneralPanel.cs:** recibe como parámetros de entrada los objetos que contiene el panel principal de la aplicación y lleva a cabo un control de la visualización de dichos objetos en función de la etiqueta que esté seleccionada.
- **ParametersPanel.cs:** permite el acceso a los componentes de texto que forman el panel de parámetros.
- **ObjectPanel.cs:** permite el acceso a los componentes de texto y campos de entrada que forman el panel de objeto.
- **UICanvas.cs:** permite el acceso a los componentes gráficos de la aplicación.
- **VectorPickerPanel.cs:** controla la visualización del panel de vectores, así como de los vectores que hayan sido marcados desde dicho panel.
- **SimManager.cs:** es el encargado de gestionar gran parte de las funcionalidades de las simulaciones. Tiene definido tres estados de simulación: Play, Pause y Stop.
    - En el estado Stop, se hace un reinicio de todas las variables y, en cada FixedUpdate, se actualizan los parámetros de entrada que el usuario introduzca para la simulación en cuestión.
    - En el estado Play, se deja el que motor de físicas de Unity actúe en cada FixedUpdate en base a los parámetros configurados por el usuario. También se lleva a cabo la actualización del panel de parámetros y de la rotación y escalado de los vectores.
    - En el estado Pause, se detiene la simulación y se almacena el estado que tenía para cuando se reanude.

Como ejemplo, en la Figura 3.1 se visualiza la jerarquía de objetos que compone la escena de "*Projectile Motion*", la cual consta de los siguientes elementos:

- La cámara principal de la escena.
- La luz global que ilumina la escena y proyecta la sobra de los objetos.
- El plano sobre el que se desarrolla la simulación.
- La esfera que es lanzada en el experimento.
- Un objeto que contiene el componente LineRenderer para dibujar la trayectoria de la esfera al ser lanzada.
- Los vectores de velocidad, aceleración, peso y rozamiento que se pueden visualizar durante la simulación.
- La interfaz de usuario.
- Un objeto vacío que contiene el script de gestión de la simulación.

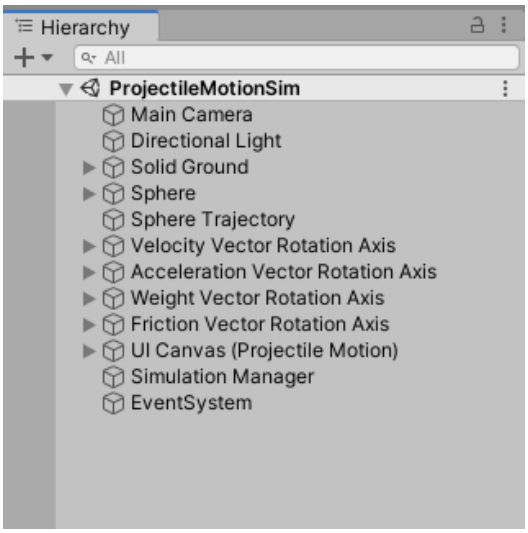

***Figura 3.1**: Jerarquía de objetos de la escena "Projectile Motion"*

3.2. INTERFAZ DE USUARIO

El diseño de la interfaz se ha procurado que sea lo más sencillo y visual posible haciendo uso de iconos y fuentes de texto atractivas para que resulte fácil de entender.

La interfaz de los escenarios de simulación consta de los siguientes apartados:

1. Encabezado de la simulación.
2. Botón de menú de opciones.
3. Tipo de cámara.
4. Panel principal.
5. Botones de control de la simulación.
6. Panel de simulación paso a paso.

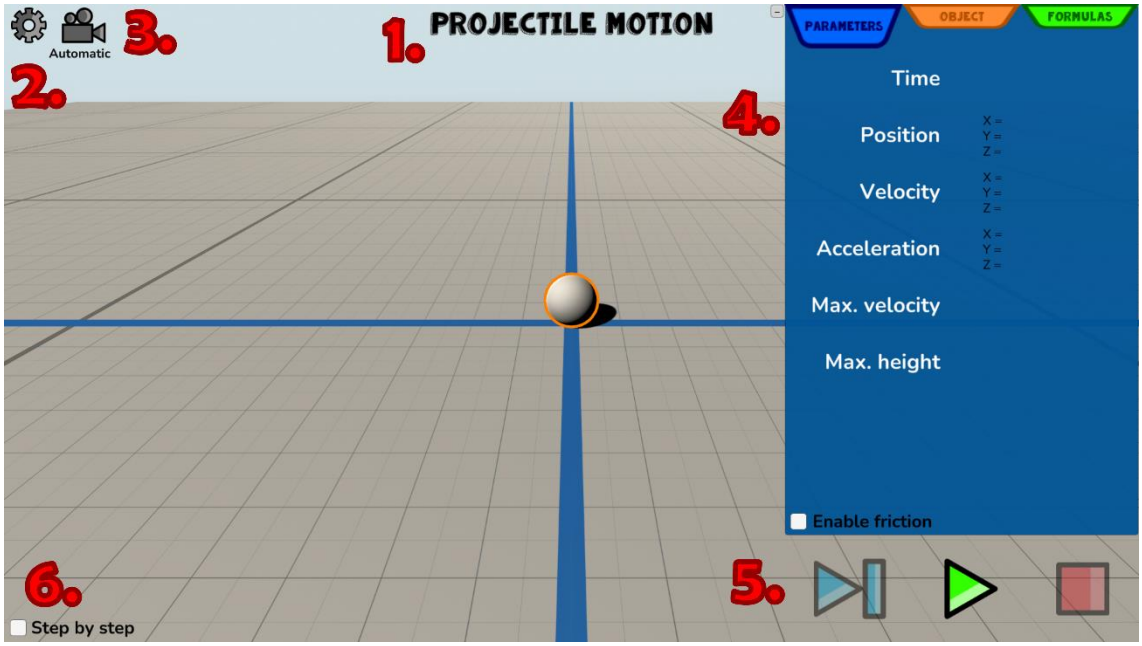

Figura 3.2: *Interfaz de la simulación "Projectile Motion"*

3.2.1. Menú principal

Al entrar en la aplicación se muestra una pantalla de inicio con un menú de selección simple con tres opciones: elegir simulación, ajustes de vídeo y cerrar aplicación.

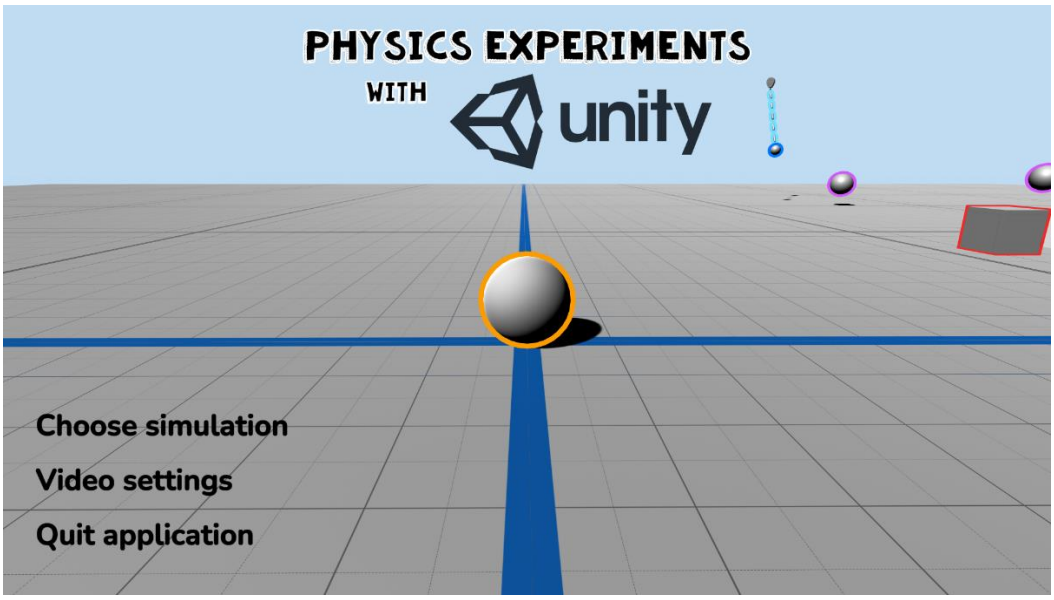

Figura 3.3: Interfaz del menú principal

Si se pulsa sobre la opción *"Video settings"* se abre un panel donde se puede establecer una configuración de vídeo que mejor se adapte a las capacidades de la pantalla y del ordenador que esté ejecutando la aplicación. Desde este panel de configuración se puede ajustar la resolución de pantalla, el modo de visualización de la ventana, la calidad gráfica de la aplicación y la tasa límite de fotogramas por segundo.

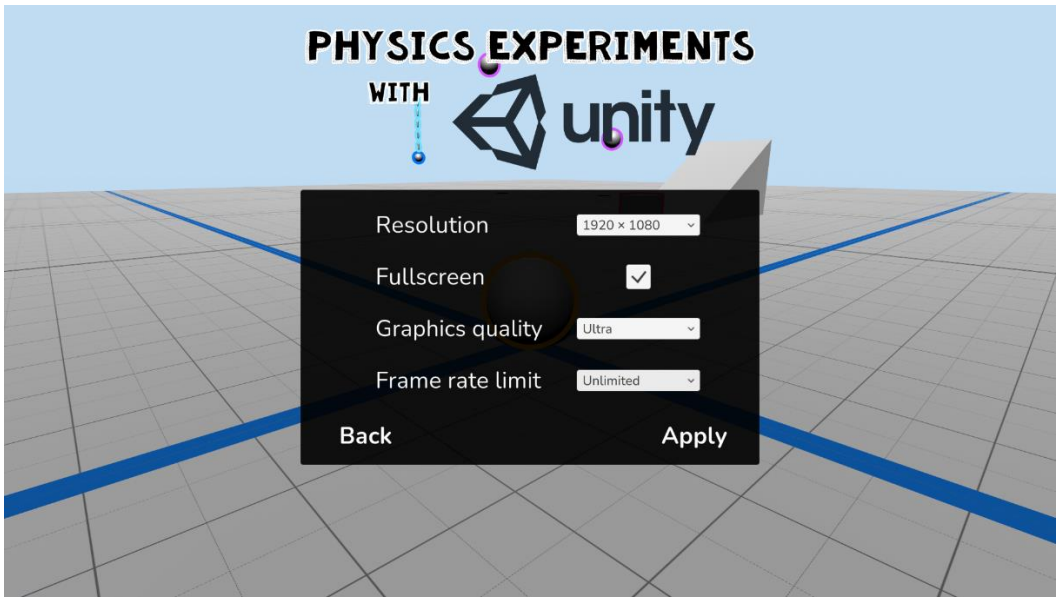

Figura 3.4: Interfaz del panel de configuración de vídeo

Al hacer clic sobre *"Quit application"*, la aplicación se cierra inmediatamente.

Por último, desde la opción *"Choose simulation"* se despliega un panel que puede ser deslizado verticalmente para mostrar y seleccionar el escenario de simulación que se desee ejecutar al hacer clic sobre la fila correspondiente. Cada fila consta de una imagen en miniatura con el escenario que se va a ejecutar acompañado del nombre de la simulación.

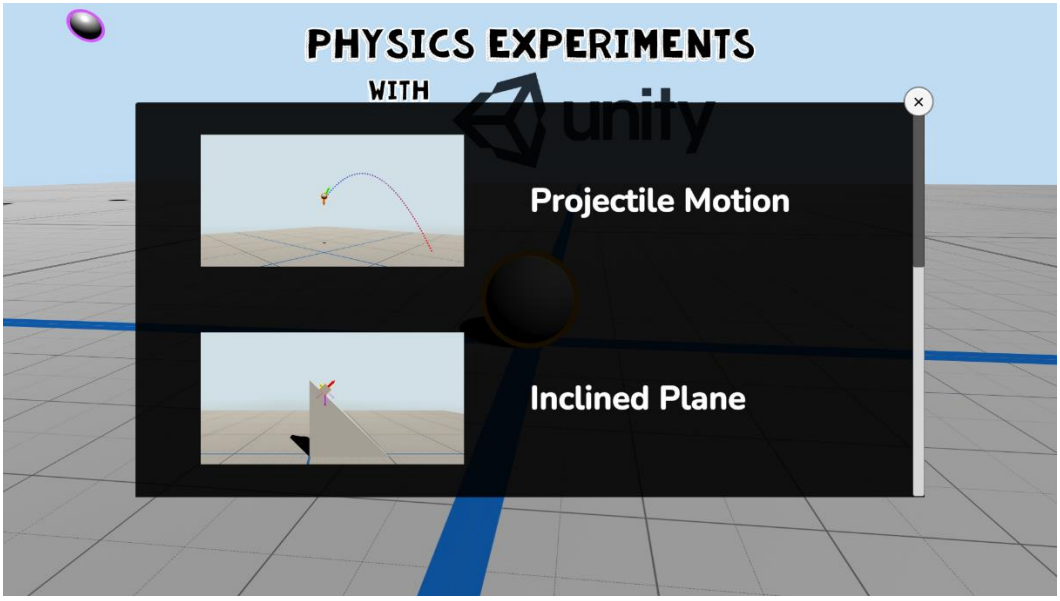

Figura 3.5: *Interfaz del panel de selección de simulación*

3.2.2. Menú de opciones

Dentro de cualquier escena de simulación, haciendo clic sobre el icono del engranaje que aparece en la parte superior izquierda de la pantalla (punto 2 de la Figura 3.1), se abre un menú de opciones que consta de cuatro apartados de selección.

Seleccionando el apartado *"Resume"* se cierra el menú de opciones para continuar con la simulación ejecutada.

Pulsando en *"Video Settings"* se muestra un panel de configuración de vídeo muy similar al que se muestra en la Figura 3.3.

Pinchando en *"Back to main menu"* se cierra la simulación que se estaba ejecutando para cargar la pantalla del menú principal que aparece al abrir la aplicación.

Finalmente, la opción de *"Quit application"* cierra la aplicación por completo.

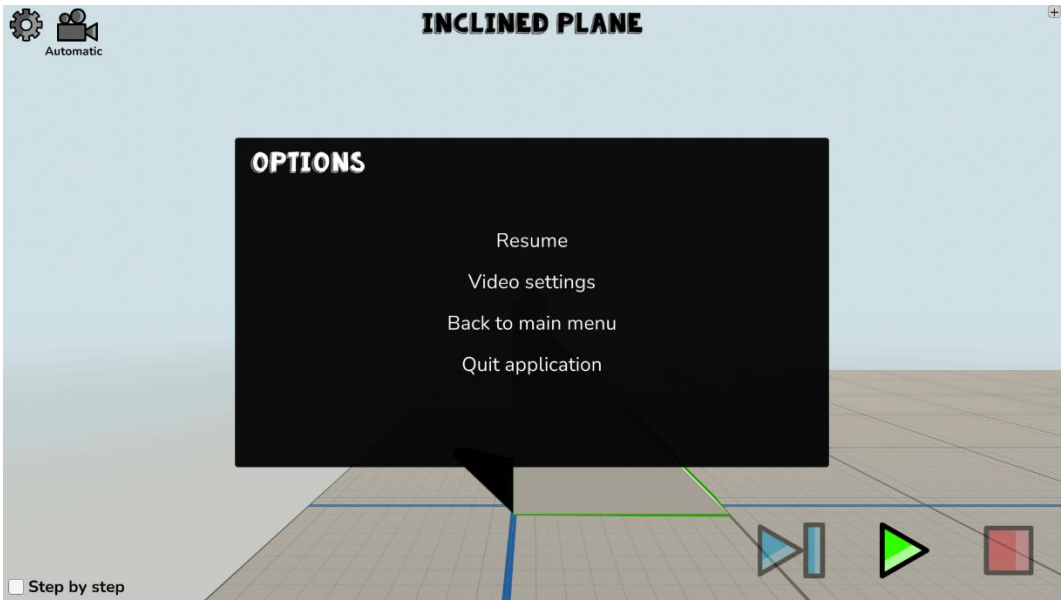

Figura 3.6: Interfaz del menú de opciones

3.2.3. Cámara

En la parte superior izquierda de la pantalla, a la derecha del icono del engranaje, aparece el icono de una cámara acompañada de un texto (punto 3 de la Figura 3.1). Pinchando sobre el icono de la cámara permite conmutar el comportamiento de la cámara entre tres modos de funcionamiento: Automatic, Slow Motion y Manual.

- **Automatic:** en el modo automático, la cámara hace un seguimiento automático de la simulación, de manera que, realiza su posicionamiento en función del movimiento de los objetos de la escena.
- **Slow Motion:** el modo cámara lenta tiene el mismo funcionamiento que el modo automático, solo que se ralentiza la velocidad de la simulación al 20% de la velocidad normal. Además, para dar una mayor sensación de ralentización del tiempo, en este modo se le aplica a la cámara un filtro de color azul cian en los bordes de la pantalla y un filtro de aberración cromática como se muestra en el Figura 3.6.
- **Manual:** en el modo manual, la cámara apunta constantemente en dirección al objeto de la simulación, pero el posicionamiento de la cámara es controlado por entrada de teclado siendo, la tecla W para acercarse al objeto, la tecla S para alejarse del objeto, la tecla A para girar en sentido izquierdo en torno al objeto, la tecla D para girar en sentido derecho en torno al objeto, la tecla Q para girar hacia arriba en torno al objeto, la tecla E para girar hacia abajo en torno al objeto y la tecla SHIFT (combinado con cualquiera de las teclas anteriores) para realizar un movimiento más rápido. La cámara manual permite alejarse del objeto de la simulación hasta un máximo de 200 metros.

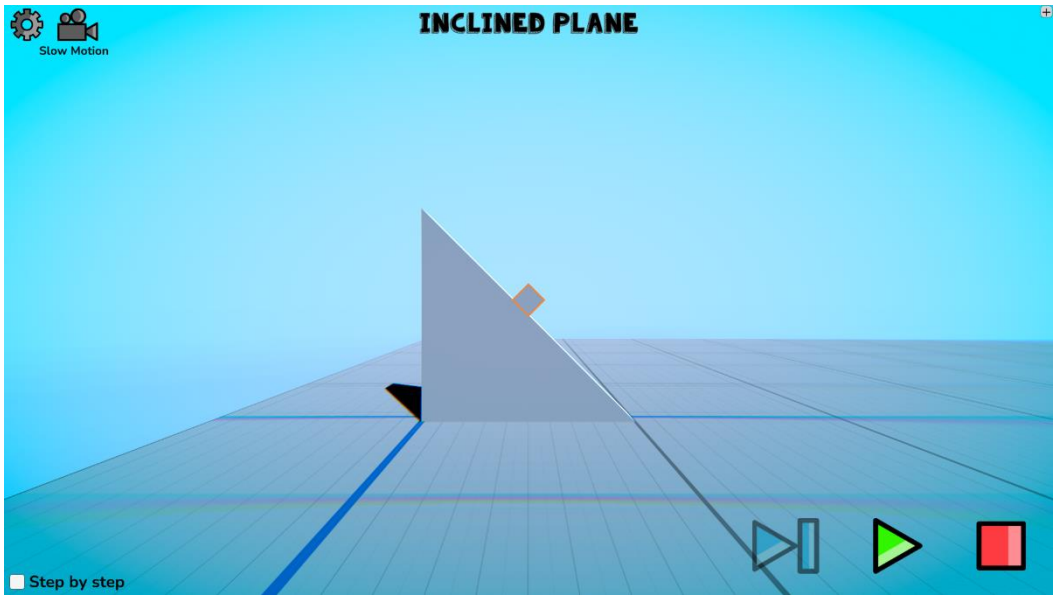

Figura 3.7: Simulación "Inclined Plane" con el modo de cámara "Slow Motion"

3.2.4. Panel principal

En el lateral derecho de la pantalla se muestra el panel principal de la interfaz, desde donde se realiza la monitorización y configuración de los parámetros más relevantes de la simulación. Utilizando las etiquetas de colores que aparecen en la parte superior del panel se puede conmutar entre los distintos paneles de parámetros, objeto y fórmulas.

Justo debajo del panel se encuentran los botones de control de la simulación, permitiendo pausar, reanudar, reiniciar o avanzar un paso de simulación.

Figura 3.8: Panel principal con los botones de control de la simulación

3.2.4.1. Panel de parámetros

El panel de parámetros, representado de color azul, monitoriza los valores de las magnitudes más relevantes de cada simulación, como pueden ser el tiempo, la velocidad, la aceleración, etc.

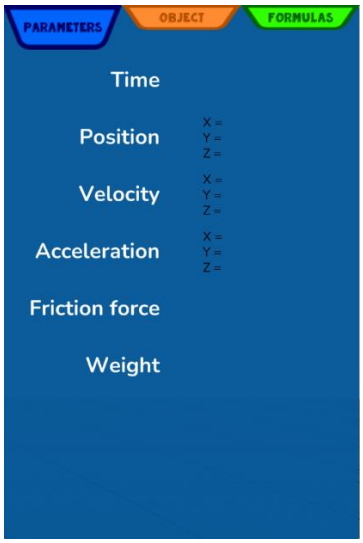

Figura 3.9: *Panel de parámetros de la simulación "Inclined Plane"*

3.2.4.2. Panel de objeto

El panel de objeto, representado de color naranja, contiene campos de entrada de datos para realizar la configuración de los diferentes objetos que participan en la simulación. Para realizar la configuración de un objeto se debe hacer clic sobre dicho objeto para que así, aparezca en el panel de objeto sus correspondientes campos, como pueden ser la masa, velocidad inicial, coeficiente de rozamiento, etc. También puede ser modificado el color del material y contorno de los objetos.

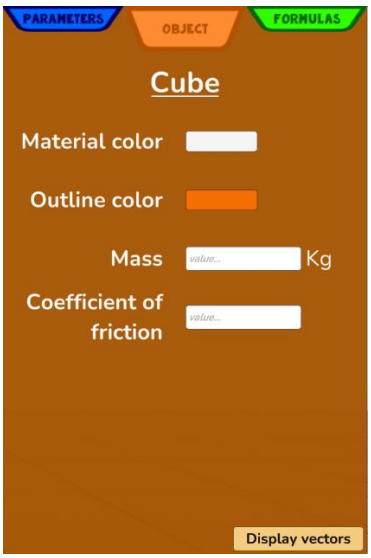

Figura 3.10: *Panel del objeto "Cube" de la simulación "Inclined Plane"*

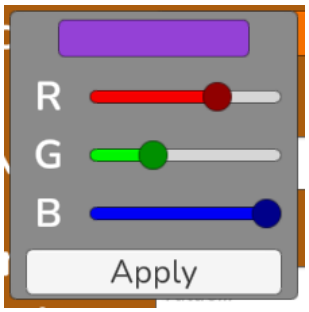

Figura 3.11: Panel de modificación de color para material y contorno de objeto

Si se introducen valores no válidos en una entrada de datos, ésta se marca de color rojo como se muestra en la Figura 3.11 y, generalmente, el programa va a considerar que tiene valor cero.

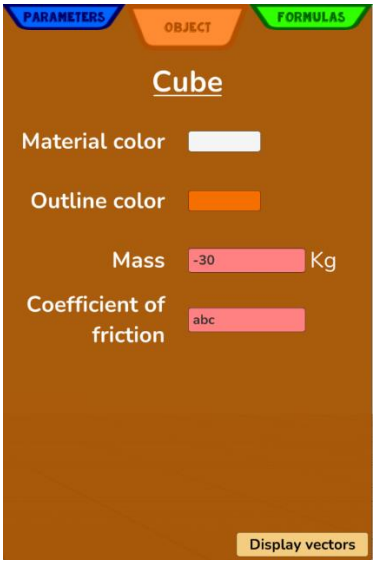

Figura 3.12: Valores de entrada no válidos para la masa y el coeficiente de rozamiento

Además, si se realizan cambios en la configuración de un objeto mientras que la simulación se está ejecutando, éstos no se aplicarán hasta que no se pulse el botón de reinicio de la simulación.

Este panel también incluye en la parte inferior derecha el botón *"Display vectors"* que, al ser pulsado, muestra un panel como el de la Figura 3.12 para seleccionar aquellos vectores que se deseen visualizar durante la simulación.

Figura 3.13: Panel de selección para visualizar vectores en simulación "Inclined Plane"

3.2.4.3. Panel de fórmulas

El tercer panel que forma parte del panel principal es el panel de fórmulas, representado de color verde, el cual contiene un dibujo esquemático de la simulación correspondiente junto con un listado de las fórmulas expresadas de manera analíticamente simple.

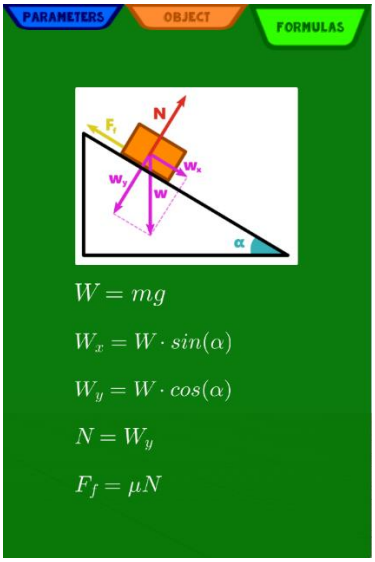

Figura 3.14: Panel de fórmulas de la simulación "Inclined Plane"

3.2.5. Panel de simulación paso a paso

Se ha implementado una funcionalidad a la aplicación que permite realizar la ejecución de las simulaciones mediante pasos de tiempo, a través de los cuales, se pueden realizar pausas más precisas de la ejecución en instantes de tiempo determinados.

Para iniciar el modo de simulación por paso de tiempo hay que activar la casilla que aparece en la parte inferior izquierda de la pantalla (punto 6 de la Figura 3.1). Activando esta casilla, la simulación pasa a estado de pausa y se muestra un nuevo panel para la simulación paso a paso, representado de color amarillo. Al mismo tiempo, en los

botones de control, el botón de ejecutar un paso de simulación pasa a estar habilitado mientras que el botón de reanudar/pausar pasa a estar deshabilitado.

El nuevo panel contiene un campo de entrada para introducir el intervalo de tiempo que se desea ejecutar hasta realizar una nueva pausa automática.

Figura 3.15: *Panel de simulación paso a paso*

3.3. ESCENARIOS DE SIMULACIÓN

3.3.1. Tiro parabólico

La simulación *"Projectile Motion"* posibilita realizar el lanzamiento de una esfera en cualquier dirección del espacio 3D y visualizar la trayectoria seguida por dicha esfera, así como los cambios que sufre su velocidad, aceleración, rozamiento con el aire, la altura y velocidad máximas, entre otros.

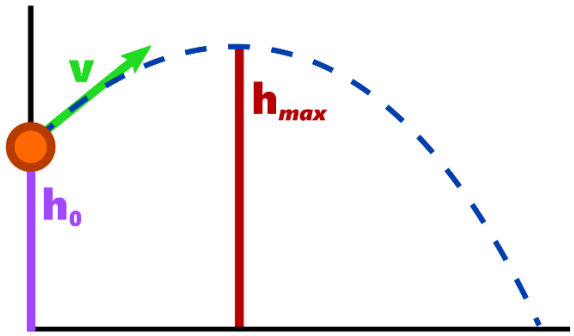

Figura 3.16: *Tiro parabólico*

En los parámetros de la esfera se puede configurar la altura inicial, la velocidad inicial de cada uno de los ejes de coordenadas cartesianas, la masa y el rozamiento con el aire. Así mismo, entre los vectores que pueden ser visualizados se encuentran la velocidad, aceleración, peso y rozamiento.

En el caso de que no exista rozamiento con el aire, la esfera va a seguir la trayectoria que se visualiza durante la configuración. Sin embargo, en el caso de que sí lo haya, la esfera se va a desviar más o menos de la trayectoria en función de la masa y el coeficiente de rozamiento configurados.

La simulación se detendrá cuando la esfera haya alcanzado una altura de cero.

Ecuaciones aplicables en la simulación:

Si no hay rozamiento:

$$\boldsymbol{a_i = g_i}$$

$$\boldsymbol{v_i = v_{0_i} + a_i \cdot t}$$

$$\boldsymbol{h_i = h_{0_i} + v_{0_i} \cdot t + \frac{1}{2} \cdot a_i \cdot t^2}$$

donde,

$i = x, y \; o \; z$

$a \equiv aceleración \; de \; la \; esfera$

$g \equiv aceleración \; de \; la \; gravedad$

$v \equiv velocidad \; de \; la \; esfera$

$h \equiv altura \; de \; la \; esfera$

$t \equiv tiempo \; de \; ejecución$

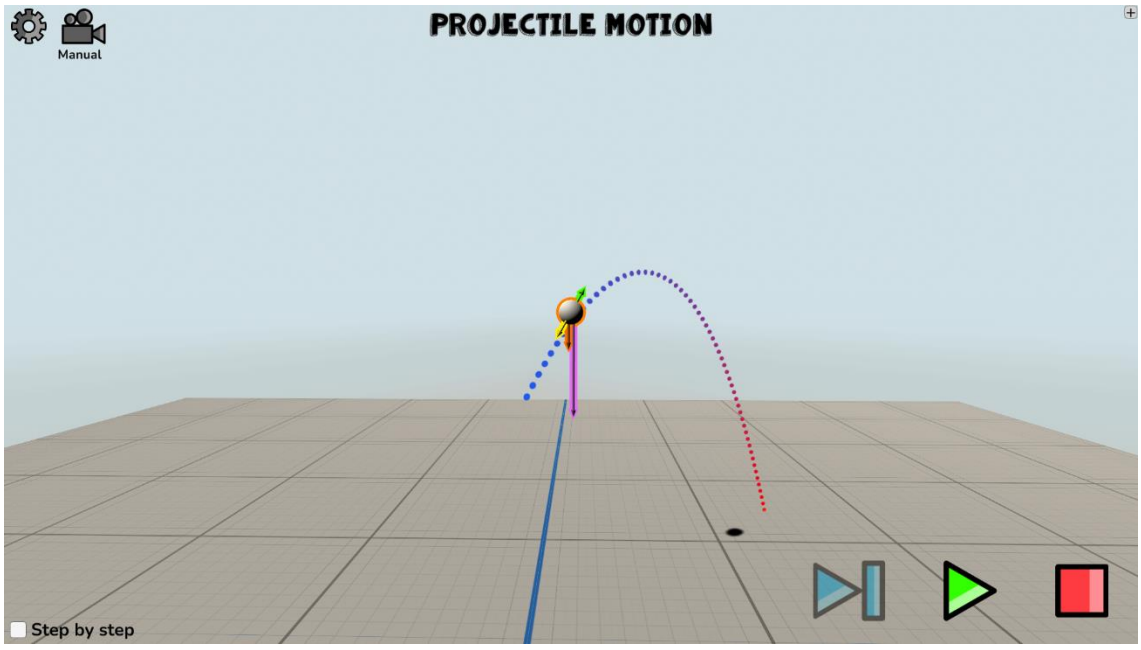

Figura 3.17: *Visualización de la simulación "Projectile Motion"*

3.3.2. Plano inclinado

La simulación *"Inclined Plane"* permite colocar un cubo que tiene una superficie caracterizada por un coeficiente de rozamiento sobre un plano con una cierta inclinación, de manera que, se puede visualizar los cambios de las distintas fuerzas que intervienen como el peso, el rozamiento o la normal y monitorizar magnitudes como la velocidad o la aceleración del cubo.

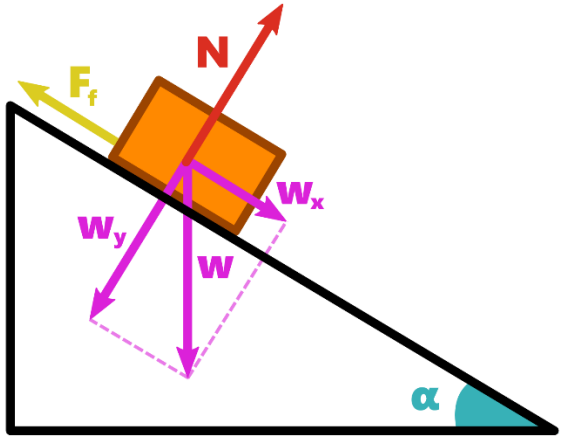

Figura 3.18: *Plano inclinado*

En los parámetros del cubo se puede establecer la masa y el coeficiente de rozamiento, mientras que en los parámetros del plano inclinado se puede configurar el ángulo de inclinación del mismo. Los vectores que pueden ser visualizados son la velocidad, aceleración, peso, rozamiento, normal.

La simulación se detendrá si el cubo alcanza el final del plano.

Ecuaciones aplicables en la simulación:

$$W = m \cdot g$$

$$W_x = W \cdot \sin(\alpha)$$

$$W_y = W \cdot \cos(\alpha)$$

$$N = W_y$$

$$F_f = \mu \cdot N$$

$$\sum F_x = W_x - F_f = m \cdot a \;\rightarrow\; a = g \cdot [\sin(\alpha) - \mu \cdot \cos(\alpha)]$$

donde,

$W \equiv$ *peso del cubo*

$m \equiv masa\ del\ cubo$

$g \equiv aceleración\ de\ la\ gravedad$

$W_x \equiv componente\ x\ del\ peso\ respecto\ al\ plano\ inclinado$

$W_y \equiv componente\ y\ del\ peso\ respecto\ al\ plano\ inclinado$

$\alpha \equiv ángulo\ de\ inclinación\ del\ plano$

$N \equiv fuerza\ normal$

$F_f = fuerza\ de\ rozamiento$

$\mu = coeficiente\ de\ rozamiento$

$a \equiv aceleración\ del\ cubo$

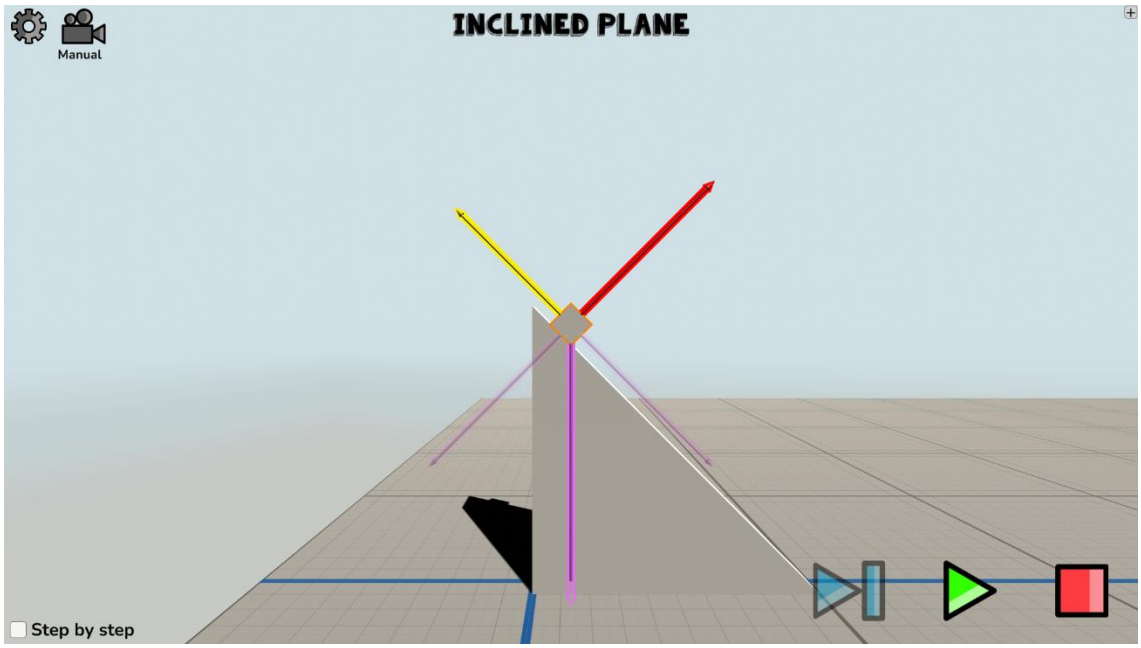

Figura 3.19: *Visualización de la simulación "Inclined Plane"*

3.3.3. Ley de Hooke

La simulación *"Hooke's Law"* posibilita hacer el estudio de la ley de elasticidad de Hooke, la cual establece que el alargamiento unitario que experimenta un cuerpo elástico es directamente proporcional a la fuerza aplicada sobre el mismo. En este caso, partimos de una esfera que tiene cierta masa y está colgando de un cuerpo elástico definido por su capacidad límite de elasticidad. Desde el panel de parámetros se monitorizan magnitudes como del peso, la fuerza de tensión ejercida por el resorte o la elongación del muelle, entre otras.[9]

[9] Wikipedia. Obtenido de https://es.wikipedia.org/wiki/Ley_de_elasticidad_de_Hooke

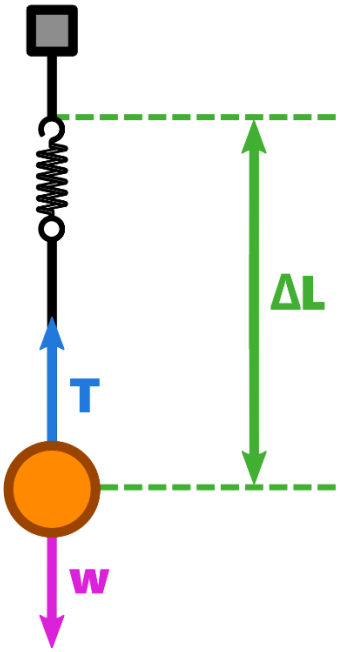

Figura 3.20: Ley de Hooke

Puede ser configurados, por tanto, la masa de la esfera y la constante elástica del resorte, y pueden ser visualizados los vectores de velocidad, aceleración, peso y tensión.

Una vez que la esfera logra detenerse, se visualiza cómo la fuerza de la tensión y del peso se han igualado.

Ecuaciones aplicables en la simulación:

$$\boldsymbol{W = m \cdot g}$$

$$\boldsymbol{F_k = k \cdot \Delta x}$$

$$\sum \boldsymbol{F_y = W + F_k = m \cdot a \ \rightarrow \ a = g + \frac{k \cdot \Delta x}{m}}$$

donde,

$W \equiv peso\ de\ la\ esfera$

$m \equiv masa\ de\ la\ esfera$

$g \equiv aceleración\ de\ la\ gravedad$

$F_k \equiv fuerza\ ejercida\ por\ el\ resorte$

$k \equiv constante\ de\ elasticidad\ del\ resorte$

$\Delta x \equiv elongación\ del\ resorte$

$a \equiv aceleración\ de\ la\ esfera$

Figura 3.21: Visualización de la simulación "Hooke's Law"

3.3.4. Fuerza de tensión

La fuerza de tensión aparece cuando se ejerce en los extremos de una cuerda, cable, cadena u otro objeto simular, dos fuerzas iguales y de sentido contrarias, produciendo que la cuerda se tense.

Para la simulación de *"Tension Force"* tenemos dos cubos, cada uno con su masa y rozamiento, unidos por una cuerda ideal. Una cuerda se considera ideal cuando su masa es despreciable, no se rompe y no se estira. Para que aparezca la tensión de la cuerda, a uno de los cubos se le aplica una fuerza que tire de todo el sistema, con el fin de visualizar si dicha fuerza es capaz de vencer el rozamiento ejercido por ambos cubos.

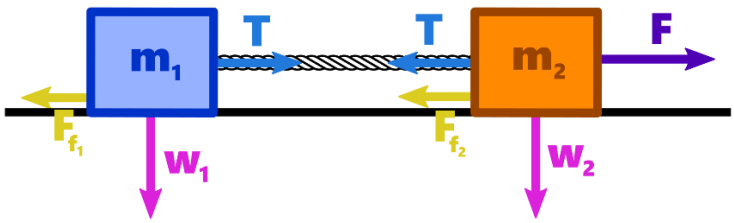

Figura 3.22: Fuerza de tensión

En la configuración se puede definir las masas y coeficientes de rozamiento de ambos cubos, además de la magnitud de la fuerza que tira de ellos. Están disponibles para ser mostrados los vectores de velocidad, aceleración, peso, tensión, rozamiento y de la fuerza.

Ecuaciones aplicables en la simulación:

$$\boldsymbol{W_i = m_i \cdot g}$$

$$\boldsymbol{F_{f_i} = \mu_i \cdot W_i}$$

$$\boldsymbol{\sum F_x = T - F_{f_1} - T - F_{f_2} + F = (m_1 + m_2) \cdot a}$$

$$\boldsymbol{v = v_0 + a \cdot t}$$

$$\boldsymbol{x = x_0 + v_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2}$$

donde,

$i = 1\ o\ 2$

$W \equiv peso\ del\ cubo$

$m \equiv masa\ del\ cubo$

$g \equiv aceleración\ de\ la\ gravedad$

$F_f \equiv fuerza\ de\ rozamiento$

$\mu \equiv coeficiente\ de\ rozamiento$

$T \equiv fuerza\ de\ tensión$

$F \equiv fuerza\ que\ tira\ del\ sistema$

$a \equiv aceleración\ del\ sistema$

$v \equiv velocidad\ del\ sistema$

$x \equiv posición\ del\ sistema$

$t \equiv tiempo\ de\ ejecución$

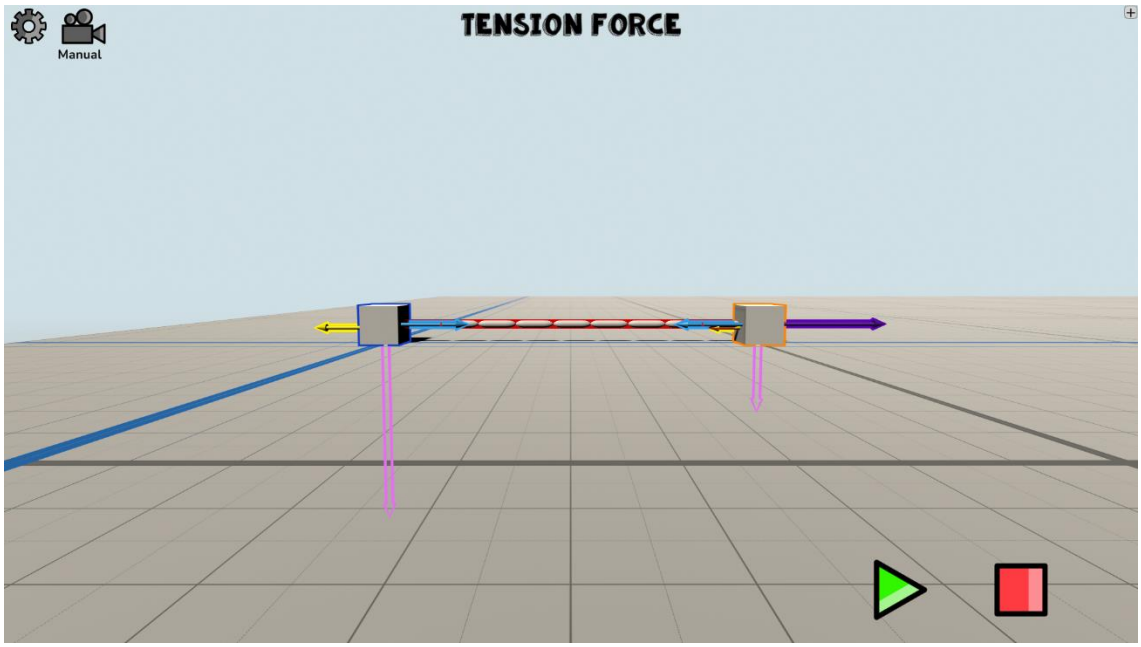

Figura 3.23: Visualización de la simulación "Tension Force"

3.3.5. Colisiones elásticas e inelásticas

Una colisión se define como un encuentro o interacción de partículas que provoca un intercambio de energía y/o cantidad de movimiento.

Cuando la energía cinética total de un sistema se conserva tras la colisión, se conoce como colisión elástica, mientras que, cuando la energía cinética del sistema tras la colisión es menor que la anterior al choque, se conoce como colisión inelástica.

A su vez, si el sistema está aislado, la cantidad de movimiento se conserva, indistintamente si es colisión elástica o inelástica. Sin embargo, si hay rozamiento, la cantidad de movimiento del sistema va disminuyendo progresivamente.

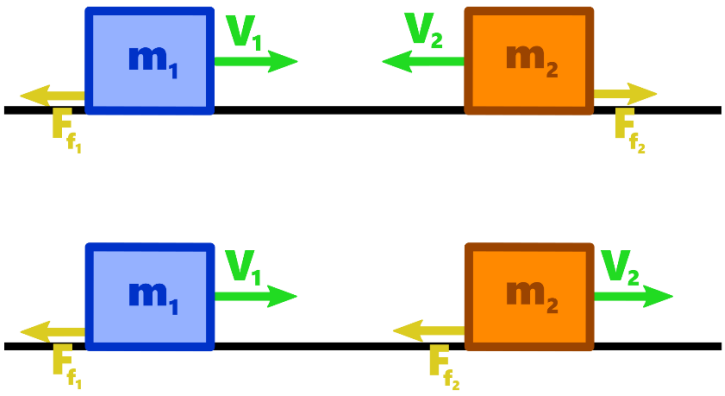

Figura 3.24: Colisiones elásticas e inelásticas

En la aplicación, pueden ser configuradas las masas, velocidad iniciales y coeficientes de rozamiento de los dos cubos. Además, se muestran los vectores de velocidad, aceleración, peso y rozamiento de cada cubo.

En la casilla que aparece en la parte inferior izquierda del panel principal se puede configurar el tipo de colisión que se va a simular. Si la casilla está marcada la colisión es elástica y, si no, la colisión es inelástica.

Ecuaciones aplicables en la simulación:

$$\boldsymbol{W_i = m_i \cdot g}$$

$$\boldsymbol{F_{f_i} = \mu_i \cdot W_i}$$

$$\boldsymbol{\sum F_{x_i} = - F_{f_i} = m_i \cdot a_i}$$

Si no hay rozamiento:

$$\boldsymbol{m_1 \cdot v_{1_0} + m_2 \cdot v_{2_0} = m_1 \cdot v_{1_f} + m_2 \cdot v_{2_f}}$$

Si la colisión es elástica:

$$\boldsymbol{\frac{1}{2} \cdot m_1 \cdot v_{1_0}^2 + \frac{1}{2} \cdot m_2 \cdot v_{2_0}^2 = \frac{1}{2} \cdot m_1 \cdot v_{1_f}^2 + \frac{1}{2} \cdot m_2 \cdot v_{2_f}^2}$$

Si la colisión es inelástica:

$$\boldsymbol{\frac{1}{2} \cdot m_1 \cdot v_{1_0}^2 + \frac{1}{2} \cdot m_2 \cdot v_{2_0}^2 > \frac{1}{2} \cdot m_1 \cdot v_{1_f}^2 + \frac{1}{2} \cdot m_2 \cdot v_{2_f}^2}$$

donde,

$i = 1\ o\ 2$

$W \equiv peso\ del\ cubo$

$m \equiv masa\ del\ cubo$

$g \equiv aceleración\ de\ la\ gravedad$

$F_f \equiv fuerza\ de\ rozamiento$

$\mu \equiv coeficiente\ de\ rozamiento$

$a \equiv aceleración\ del\ cubo$

$v \equiv velocidad\ del\ cubo$

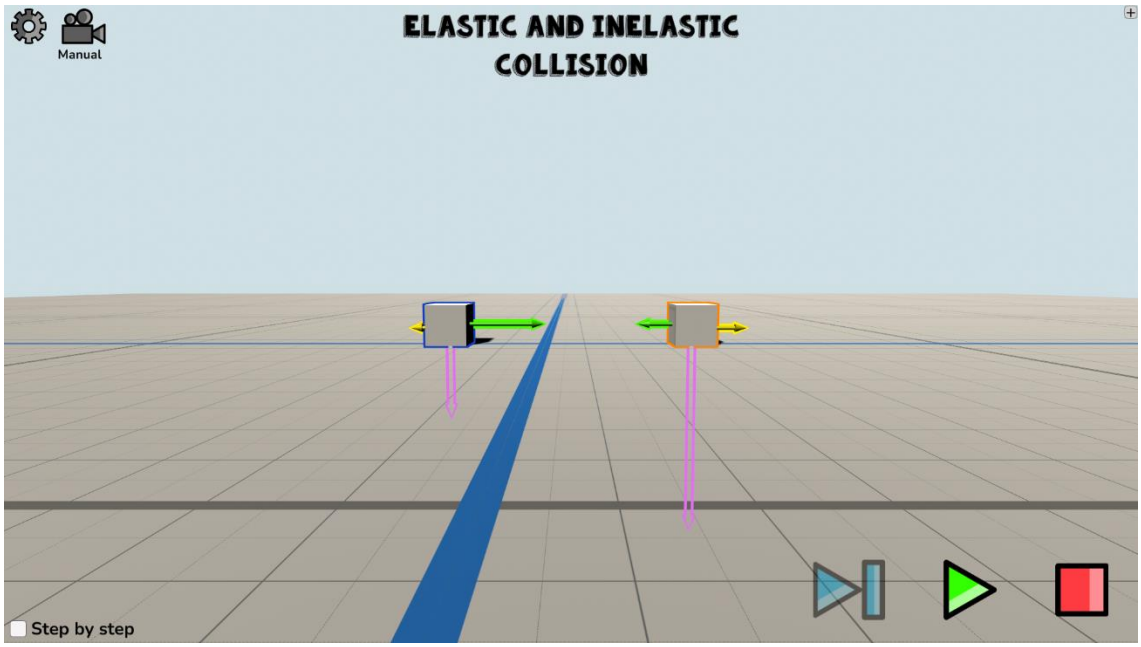

Figura 3.25: *Visualización de la simulación "Elastic and Inelastic Collisions"*

4. CONCLUSIONES Y LÍNEAS FUTURAS

Como se ha podido ver, la plataforma de desarrollo de Unity ha permitido crear, de manera simple, escenarios en 3D visualmente atractivos acompañados de una interfaz de usuario intuitiva y fácil de utilizar. Además, el motor de físicas que trae integrado ha facilitado el desarrollo de la aplicación para simular experimentos físicos de forma virtual.

Esta herramienta puede ayudar a que estudiantes de instituto y universidad les sea más sencillo comprender la cinemática y la dinámica visualizando la variación de las magnitudes de los diferentes parámetros, así como de los vectores, y ver los cambios en el comportamiento de la simulación en función de la configuración inicial que se establezca.

Para este proyecto se han desarrollado cinco escenarios de simulación, pero con las capacidades y herramientas que ofrece Unity podría extenderse la aplicación con nuevos escenarios de simulación e incluso nuevas funcionalidades como podría ser, por ejemplo, una opción para que los usuarios puedan crear su propio experimento a partir de un escenario vacío en el que puedan ir añadiendo diferentes objetos y configurarlos a su gusto con la posición, velocidad inicial, rozamiento u otros parámetros. De esta manera, los usuarios podrían adaptar una simulación a sus necesidades y experimentar nuevas situaciones, y no limitarse a las que el programa ofrece por defecto.

En cuanto a mi experiencia con Unity, es el primer proyecto que desarrollo con este software y me ha resultado intuitivo y fácil de utilizar, y me parece que es una herramienta muy accesible para que cualquier persona o grupo de trabajo con pocos recursos sea capaz de desarrollar una aplicación o videojuego.