1. Programación de videojuegos 2D.

Desarrollo de Aplicaciones Multiplataforma. Programación multimedia y dispositivos móviles.

1. Los videojuegos.

1.1 Historia del videojuego.

- En los años 70 Ralph Baer, junto a su equipo, había conseguido hacer funcionar un juego de ping pong para dos jugadores.
- Como de costumbre, el problema estaba a la hora de comercializar el proyecto. Preguntó a muchas empresas y siempre obtuvo respuestas negativas por su parte, hasta que encontró a Magnavox (donde trabajaba Bill Enders, el cual les habló del proyecto de Baer). Finalmente, esta empresa sí que acepta las condiciones y empieza a trabajar para comercializarla.
- En 1972 se lanza la Magnavox Odyssey al mercado, con 12 juegos distintos. Esta consola solo podía funcionar en los televisores de la misma marca.

1. Los videojuegos.

1.1 Historia del videojuego.

- Por otra parte, Nolan Bushnell y Ted Dabney juntan fondos para crear una empresa. El nombre de esta empresa sería Atari. Cuando Bushnell asiste a la Magnavox Profit Caravan de ese mismo año y ve la consola de esa compañía, tiene la genial idea de crear un juego similar. Contratan a Allan Alcorn y le encargan el proyecto. Tres meses más tarde ya tiene un prototipo operativo y a Bushnell y Dabney les encanta. Lo bautizan como Pong.
- Introducen este nuevo juego en bares y en seguida causa una gran sensación, llegándose hasta "estropear" las máquinas debido a que las cajas de las monedas estaban a rebosar. Pong sería el mayor éxito jamás visto en la industria de los videojuegos hasta ese momento. Sin embargo, Bushnell tendría problemas con Baer, ya que este último le acusaba de plagio de su juego. Finalmente consiguen un trato del que Bushnell sale completamente beneficiado.

1. Los videojuegos.

1.2 Época actual.

- Finalmente llegamos a la era moderna, la de las consolas en HD, con disco duro, con gráficos impresionantes y sobre todo con muchos juegos que siguen haciendo historia y que en un futuro serán recordados.
- Empieza el año 2000 con la aparición de la consola de sobremesa más vendida de toda la historia: PlayStation 2, la cual tuvo serios problemas para distribuirse en sus inicios.
- Al cabo de un año aparece la nueva consola de Nintendo: la GameCube. A ella se le suma la XBOX, el primer sistema de Microsoft. Estas dos consolas, pese a ser buenas máquinas, nunca pudieron con el éxito arrollador de PlayStation 2 y quedaron relegadas en un segundo lugar por la lucha en el mercado, compitiendo entre sí.

1. Los videojuegos.

1.2 Época actual.

- La penúltima generación de consolas se compone de la Playstation 4, la XBOX One y la Nintendo Switch, junto con el renacimiento del PC (se habla incluso de la PC Master Race, dada la dificultad que tienen las consolas en alcanzar el rendimiento gráfico de los PC) y sin olvidar la irrupción de los juegos para plataformas móviles, (smartphones, tablets,…).
- Los videojuegos han pasado a generar más dinero que las industrias del cine (solo entradas vendidas) y la música juntas, como en el caso de España; la industria de videojuegos generó 57.600 millones de euros durante 2009 en todo el mundo.
- Durante el año 2020 se lanzaron al mercado las consolas de nueva generación tanto de Sony como de Microsoft, llamadas PS5 y Xbox Series respectivamente. En el mercado de PC aparece también una nueva generación de las tarjetas gráficas de Nvidia (serie 3000) lo que implica un nuevo salto en la mejora gráfica de los juegos.

1. Los videojuegos.

1.3 Clasificación de los videojuegos.

- Arcade: los videojuegos tipo arcade se caracterizan por su jugabilidad simple, repetitiva y de acción rápida. Más que un género en sí se trata de un calificativo bajo el que se engloban todos aquellos juegos típicos de las máquinas recreativas (beat 'em up, matamarcianos, plataformas). Tuvieron su época dorada en la década de 1980.
- Deportes: los videojuegos de deportes son aquellos que simulan deportes del mundo real. Entre ellos encontramos golf, tenis, fútbol, hockey, juegos olímpicos, etc. La mecánica del juego es la misma que en el deporte original, aunque a veces incorpora algunos añadidos.
- Rol: emparentados con los de aventura, los videojuegos de rol, o RPG, se caracterizan por la interacción con el personaje, una historia profunda y una evolución del personaje a medida que la historia avanza. Para lograr la evolución generalmente se hace que el jugador se enfrasque en una aventura donde irá conociendo nuevos personajes, explorando el mundo para ir juntando armas, experiencia, aliados e incluso magia.

1. Los videojuegos.

1.3 Clasificación de los videojuegos.

- Estrategia: se caracterizan por la necesidad de manipular a un numeroso grupo de personajes, objetos o datos, haciendo uso de la inteligencia y la planificación, para lograr los objetivos. Aunque la mayoría de estos juegos son fundamentalmente de temática bélica, los hay también de estrategia económica, empresarial o social. Dos grandes subgéneros son los juegos de estrategia en tiempo real (también llamados RTS, siglas en inglés de real-time strategy), y los por turnos (TBS, siglas también en inglés de turn based strategy).
- Simulación: este género se caracteriza por recrear situaciones o actividades del mundo real, dejando al jugador tomar el control de lo que ocurre. En ocasiones la simulación pretende un alto grado de verosimilitud, lo que le otorga una componente didáctica. Los tipos de simulación más populares son los de manejo de vehículos (pilotar un coche, un avión, un tren...), los de construcción (construir una ciudad, un parque de atracciones o un imperio), o los de vida (dirigir la vida de una persona o un animal virtual).
- Aventura Gráfica: a comienzos de los 90, el uso creciente del ratón dio pie a los juegos de aventura de tipo «Point and click», también llamados aventura gráfica, en los que ya no se hacía necesaria la introducción de comandos. El jugador puede, por ejemplo, hacer clic con el puntero sobre una cuerda para recogerla.

1. Los videojuegos.

1.3 Clasificación de los videojuegos.

- FPS: en los videojuegos de disparos en primera persona, conocidos también como FPS (first person shooter), se maneja al protagonista desde una perspectiva subjetiva, es decir, vemos en la pantalla lo que ve nuestro personaje, y por tanto no lo vemos a él. Sí vemos en cambio su arma, en primer plano, la cual deberemos usar para abatir a los diferentes enemigos que aparecerán frente a nosotros al avanzar. La perspectiva en primera persona, en un entorno 3D, tiene por meta dar al jugador la impresión de ser el personaje, buscando con ello una experiencia más realista de juego.
- Puzzles: los juegos de lógica o de agilidad mental, también llamados de puzzles o rompecabezas, ponen a prueba la inteligencia del jugador para la resolución de problemas, que pueden ser de índole matemático, espacial o lógico.
- Battle Royale: es un género en el cual se combinan mecánicas de los FPS con elementos de supervivencia, el objetivo es sobrevivir a las sucesivas rondas que reducen el espacio de juego. El nombre del género está tomado de la novela japonesa de año 1999 llamada Battle Royale, que presenta un tema similar al de un último jugador en pie. Uno de los primeros juegos en popularizar el género fue el PUBG (PlayerUnknown's Battlegrounds), al que le seguirían el Fortnite y muchos más.

2. Programación de videojuegos.

- La programación de videojuegos siempre ha implicado una gran complejidad, el acceso a los recursos hardware, audio, red y sobre todo la tarjeta gráfica, donde a menudo es necesario programar a muy bajo nivel utilizando el lenguaje C ó ensamblador, lo que suponía una gran barrera de entrada a programadores noveles.
- A finales de los 90 y principios de este milenio, solo grandes estudios podían permitirse las herramientas necesarias para desarrollar videojuegos de gran calidad, llamados triple A.
- La aparición de librerías específicas y posteriormente la popularización de motores de bajo coste o gratuitos han hecho posible que más programadores se animen a programar juegos. El crecimiento del número de estudios de desarrollo “indies” (independientes) ha sido tan rápido que ha provocado una saturación de títulos que dificulta la visibilidad de los mismos.

2. Programación de videojuegos.

2.1 Librerías.

- OpenGL: (Open Graphics Library) es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Fue desarrollada originalmente por Silicon Graphics Inc. (SGI) en 1992 y se usa ampliamente en CAD, realidad virtual, representación científica, visualización de información y simulación de vuelo. También se usa en desarrollo de videojuegos, donde compite con Direct3D en plataformas Microsoft Windows.
- DirectX: DirectX es una colección de API desarrolladas para facilitar las complejas tareas relacionadas con multimedia, especialmente programación de juegos y vídeo, en la plataforma Microsoft Windows.
- SDL: Simple DirectMedia Layer (SDL) es un conjunto de bibliotecas desarrolladas en el lenguaje de programación C que proporcionan funciones básicas para realizar operaciones de dibujo en dos dimensiones, gestión de efectos de sonido y música, además de carga y gestión de imágenes. Fueron desarrolladas inicialmente por Sam Lantinga, un desarrollador de videojuegos para la plataforma GNU/Linux.

2. Programación de videojuegos.

2.2 Motores/engines.

- Un motor de videojuego es un término que hace referencia a una serie de rutinas de programación que permiten el diseño, la creación y la representación de un videojuego. Del mismo modo existen motores de juegos que operan tanto en consolas de videojuegos como en sistemas operativos. La funcionalidad básica de un motor es proveer al videojuego de un motor de renderizado para los gráficos 2D y 3D, motor físico o detector de colisiones, sonidos, scripting, animación, inteligencia artificial, redes, streaming, administración de memoria y un escenario gráfico.
- Existen muchos motores de videojuegos con diferentes características, los más completos incluyen editores de niveles que integran todos los aspectos del desarrollo del juego.
- Recientemente motores de gran potencia, cuyas licencias costaban miles o decenas de miles de euros han lanzado versiones gratuitas o con costes muy reducidos (por ejemplo una parte de los beneficios).

2. Programación de videojuegos.

2.2 Motores/engines.

- Unreal Engine: creados por la compañía Epic Games. Implementado inicialmente en el shooter en primera persona Unreal en 1998, siendo la base de juegos como Unreal Tournament, Deus Ex, Turok, Tom Clancy's Rainbow Six: Vegas, America's Army, Red Steel, Gears of War, BioShock….Está escrito en Unreal Script (Lenguaje propio similar Java o C# modificado), siendo compatible con varias plataformas como PC (Microsoft Windows, GNU/Linux), Apple Macintosh (Mac OS, Mac OS X) y la mayoría de consolas (Dreamcast, Gamecube, Wii, Xbox, Xbox 360, PlayStation 2, PlayStation 3, Xbox One, PlayStation 4). Unreal Engine también ofrece varias herramientas adicionales de gran ayuda para diseñadores y artistas. La última versión de este motor es el Unreal Engine 5, está diseñado para las APIs OpenGL y DirectX.
- CryEngine: es un motor de juego creado por la empresa alemana desarrolladora de software Crytek, originalmente un motor de demostración para la empresa Nvidia, que al demostrar un gran potencial se implementa por primera vez en el videojuego Far Cry, desarrollado por la misma empresa creadora del motor. El 30 de marzo de 2006, la totalidad de los derechos de CryEngine son adquiridos por la distribuidora de videojuegos Ubisoft. Es compatible con la mayoría de las plataformas del mercado.

2. Programación de videojuegos.

2.2 Motores/engines.

- Recientemente Amazon ha lanzado su propio motor basado en CryEngine, llamado Lumberyard, de forma gratuita pero ligado fuertemente a los servicios en la nube de la plataforma de Amazon.
- Unity3D es un motor de videojuego multiplataforma creado por Unity Technologies. Unity está disponible como plataforma de desarrollo para Microsoft Windows, OS X y Linux, y permite crear juegos para Windows, OS X, Linux, Xbox 360, PlayStation 3, Playstation Vita, Wii, Wii U, iPad, iPhone, Android y Windows Phone. Gracias al plugin web de Unity, también se pueden desarrollar videojuegos de navegador para Windows y Mac. Se ha convertido en la referencia para los desarrolladores indies.
- Godot: un motor muy reciente, permite realizar juegos 2D y 3D, su licencia es MIT que es incluso más permisiva que la GPL.

3. Conceptos previos de videojuegos.

3.1 Sprite.

- Un sprite (duendecillo o hada en inglés) es una imagen usada para representar un ente gráficamente (o parte de él) y poder posicionarlo en el lugar deseado de una escena mayor. Mediante este sistema además se pueden crear animaciones que representen dicho ente cambiando el sprite correspondiente. Un ente no tiene estar representado gráficamente por un único sprite sino que puede estar dividido en varios diferentes por:

  Limitaciones técnicas (tamaño máximo del sprite, número de colores, etc.)

  Optimización (especialmente si tiene formas irregulares o inconexas).

  Crear representaciones mediante composición (por ejemplo, cambiar el arma o vestimenta de un personaje).

- Los sprites pueden ser cualquier imagen o información que representa una imagen. Si bien teóricamente nada impide la usar imágenes vectoriales, las basta mayoría de las veces esta técnica es usada exclusivamente con imágenes bitmap.

3. Conceptos previos de videojuegos.

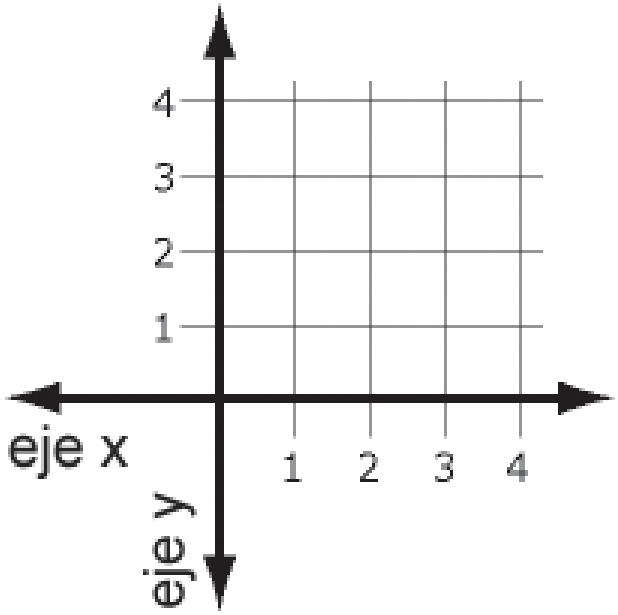
3.1 Sprite.

- Los sprites, en caso de ser un mapa de bits, para no tener que ser rectangulares o requerir un fondo con solo color, suelen contener información sobre sus partes transparentes como puede ser utilizar un color de la paleta, uso de otra imagen con la máscara de transparencia, o directamente la cantidad de componente alpha de cada pixel.
- Una forma muy común para almacenar todos los sprites es el uso de sprite sheets o plantillas de sprites, que consisten en imágenes que contienen todos los sprites, normalmente relacionados que pertenecen a un ente, y que también en vez de almacenar internamente todos los sprites por separado, lo que se hace es utilizar directamente tan solo la parte de la plantilla que corresponde a ese sprite.

3. Conceptos previos de videojuegos.

3.2 Coordenadas.

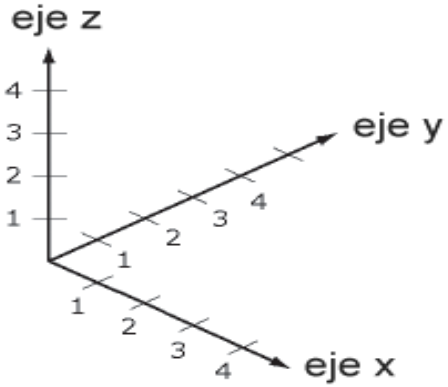
- Uno de los más importantes conceptos dentro del desarrollo de videojuegos es el eje de coordenadas. Es tan importante debido a que nos permite recrear un plano o un espacio y posicionar en su interior los diferentes elementos que forman el videojuego.
- Los ejes de coordenadas cartesianas son una parte fundamental de las matemáticas y la física, ya que nos sirven para representar funciones y también posiciones. En ellos, cada eje representa una dimensión, por lo que si hablamos de videojuegos encontraremos dos ejes básicos.
- Eje de coordenadas en el plano: representado por dos ejes X e Y que indican las dimensiones ancho y alto respectivamente. Se utiliza en juegos en dos dimensiones.

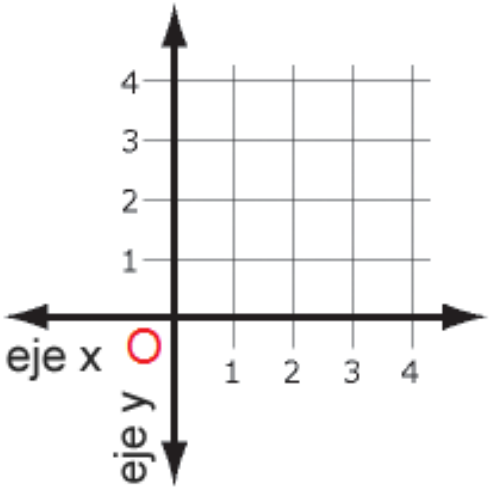


3. Conceptos previos de videojuegos.

3.2 Coordenadas.

- Eje de coordenadas en el espacio: representado por tres ejes, X, Y y Z que indican las dimensiones ancho, alto y profundidad respectivamente. Se utiliza en juegos en tres dimensiones, aunque también sirve para juegos en 2D cuando se utiliza el eje Z para indicar qué elementos se encuentran por encima de los otros en el plano, algo que normalmente se indica como una propiedad de los objetos que determina el orden de renderizado de abajo hacia arriba.
- Origen de coordenadas: el término origen es muy utilizado cuando se habla del eje de coordenadas. Corresponde al lugar donde se cortan los ejes y tiene el valor cero. Es muy importante ya que gracias a él podemos tomar una referencia inicial a partir de la cual posicionar elementos sobre el eje de coordenadas, utilizando coordenadas, valga la redundancia

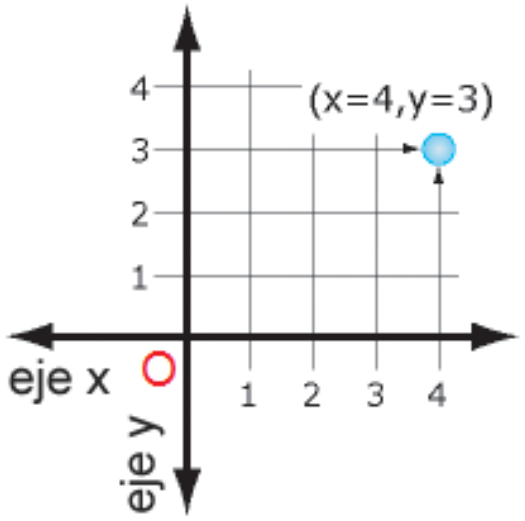


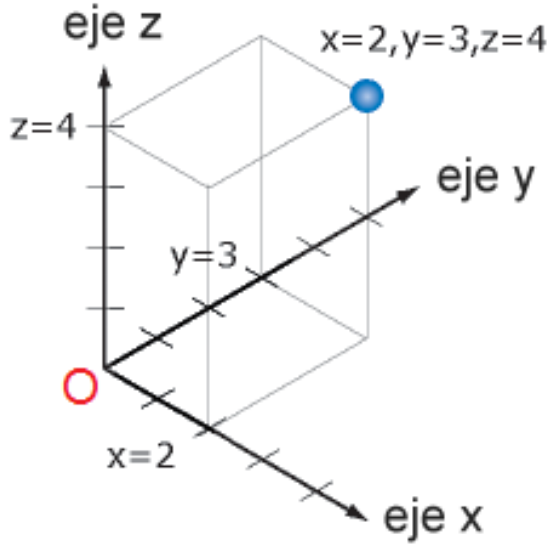


3. Conceptos previos de videojuegos.

3.2 Coordenadas.

- Representación de elementos: tomando un punto sobre el eje de coordenadas (que es la mínima unidad que se puede representar), cada coordenada nos indica la distancia de cada dimensión partiendo del origen donde se encuentra el punto. Por ejemplo un punto A en el plano 2D estará formado por dos coordenadas A = (X,Y).
- En el espacio 3D estará formado por tres A = (X,Y,Z).

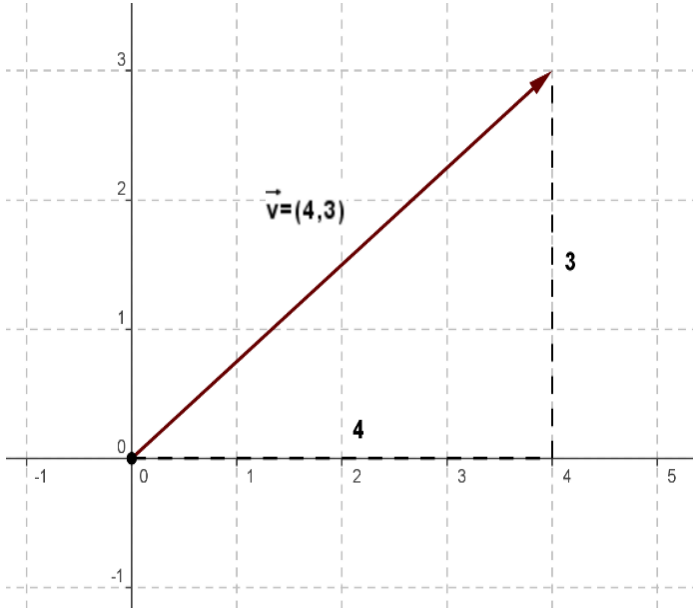


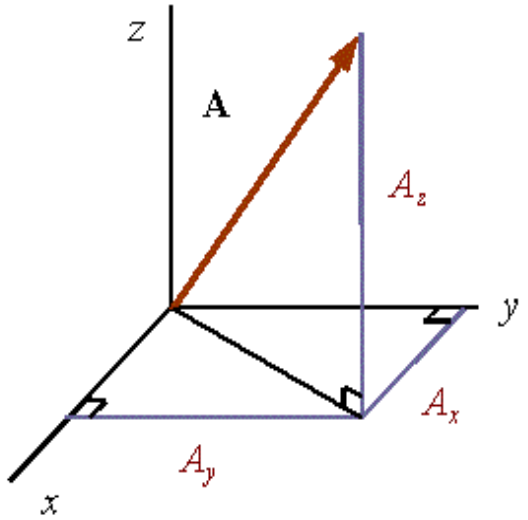


3. Conceptos previos de videojuegos.

3.2 Coordenadas.

• Otro elemento que se puede representar sobre el eje de coordenadas es un vector, que no es más que la distancia entre el origen y un punto, con dirección al punto.





4. Unity.

4.1 Introducción.

• Unity es un motor gráfico 3D para PC y Mac que viene empaquetado como una herramienta para crear juegos, aplicaciones interactivas, visualizaciones y animaciones en 3D y tiempo real. Unity puede publicar contenido para múltiples plataformas como PC, Mac, Nintendo Swift y iPhone. El motor también puede publicar juegos basados en web usando el plugin Unity web player.

• El editor de Unity es el centro de la línea de producción, ofreciendo un completo editor visual para crear juegos. **El contenido del juego es construido desde el editor** y el gameplay se programa usando un lenguaje de scripts. Esto significa que los desarrolladores no necesitan ser unos expertos en C++ para crear juegos con Unity, ya que las mecánicas de juego son compiladas usando una versión de JavaScript, **C#** o Boo, un dialecto de Python (el soporte a Boo se eliminó a partir de la versión 5).

4. Unity.

4.1 Introducción.

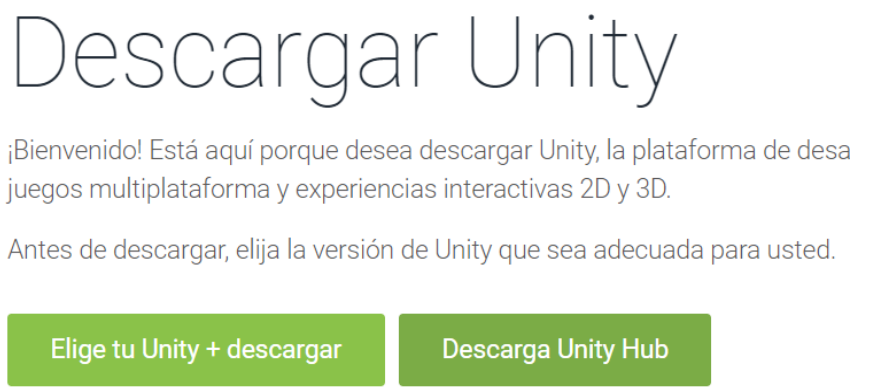
• Los juegos creados en Unity son estructurados en escenas como el motor Gamebryo. En Unity una escena puede ser cualquier parte del juego, desde el menú de inicio como un nivel o área de tu juego; la elección es tuya ya que una escena es un lienzo en blanco sobre el que dibujar cada parte del juego usando las herramientas de Unity.

• El motor también incluye un editor de terrenos, desde donde puedes crear un terreno (como una hoja en blanco), sobre la que los artistas podrán esculpir la geometría del terreno usando herramientas visuales, pintar o texturizar, cubrir de hierba o colocar árboles y otros elementos de terreno importados desde aplicaciones 3D como 3DS Max o Maya.

4. Unity.

4.2 Instalación de Unity.

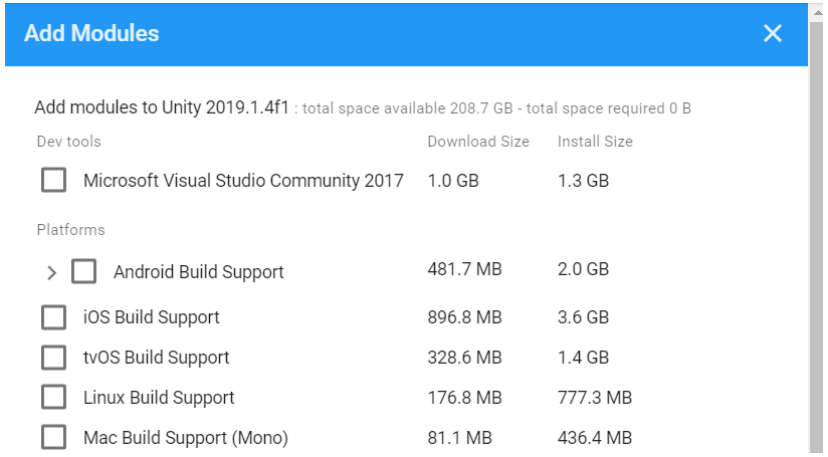
- En versiones reciente de Unity es posible descargar el editor de forma independiente, o bien, a través de un instalador independiente llamado Unity Hub que permite instalar varias versiones diferentes del editor en el mismo equipo, de forma que abriremos cada proyecto con la versión deseada.



4. Unity.

4.2 Instalación de Unity.

• A través de Unity Hub también se pueden instalar los módulos de soporte para distintas plataformas (Android, Mac, etc…) y el editor de código para scripts (normalmente Visual Studio).

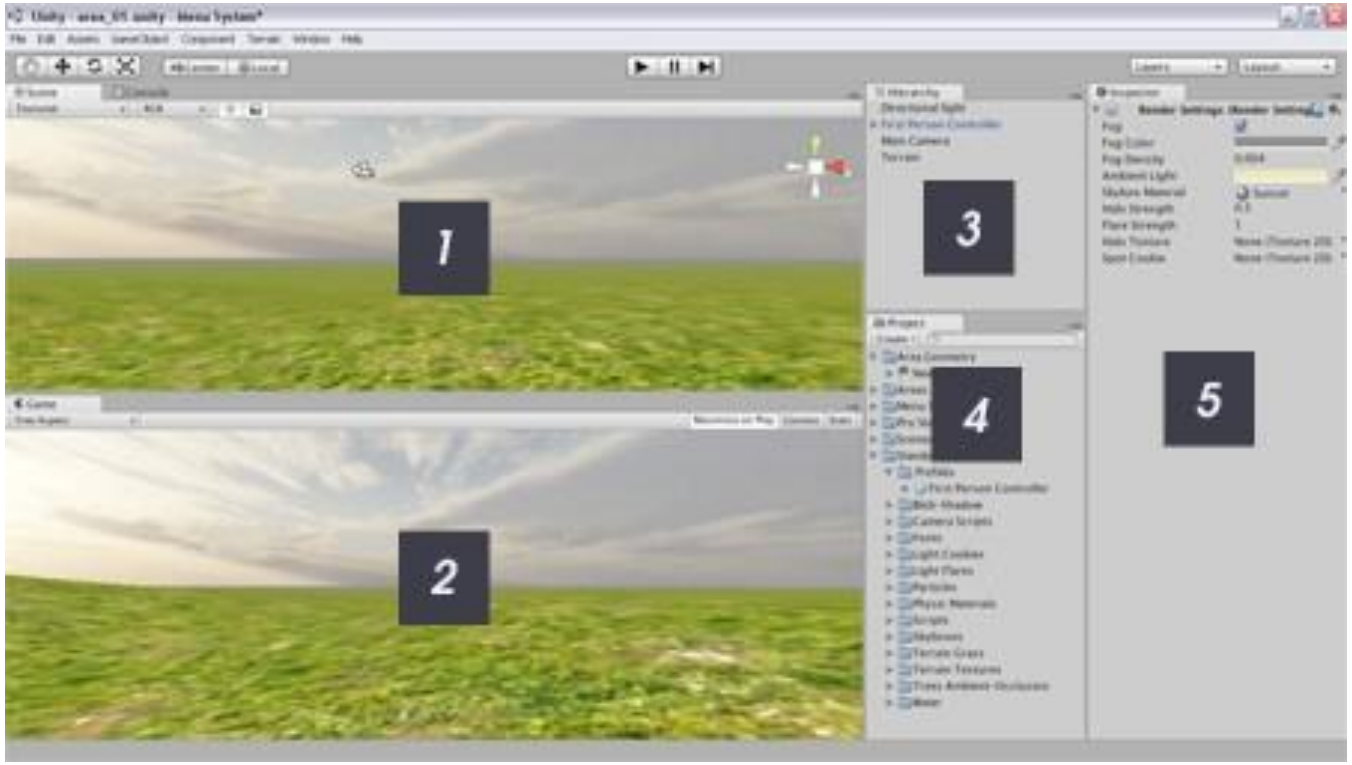


- Si no tenéis instalado un editor de C# os recomiendo marcar la opción de Visual Studio dado que se integra bien con Unity. También deberéis marcar la opción de Android Build Support para poder compilar para Android.

4. Unity.

4.3 La interfaz de usuario de Unity.

- Existen principalmente 5 áreas de la interfaz de Unity, numeradas en la imagen de abajo.



4. Unity.

4.3 La interfaz de usuario de Unity.

1. Vista de Escena.

- La escena es el área de construcción de Unity donde construimos visualmente cada escena de nuestro juego.

- En la vista de juego obtendremos una previsualización de nuestro juego. En cualquier momento podemos reproducir nuestro juego y jugarlo en esta vista.

2. Vista de Juego.

- En esta vista vemos el juego en acción y podemos interactuar con el mismo.

3. Vista de Jerarquía.

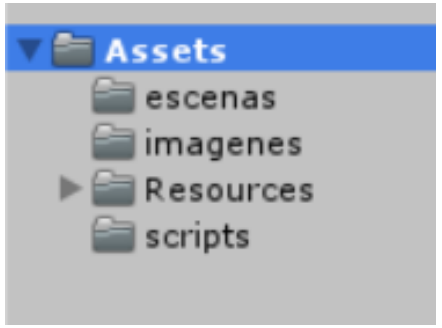
- La vista de jerarquía contiene todos los objetos en la escena actual.

4. Unity.

4.3 La interfaz de usuario de Unity.

4. Vista de Proyecto.

• Esta es la librería de assets para nuestro juego. Esto incluye modelos 2D/3D, código, sónidos, etc. Puedes importar objetos 3D de distintas aplicaciones a la librería, puedes importar texturas y crear otros objetos como Scripts o Prefabs que se almacenarán aquí. Todos los assets que importes en tu juego se almacenarán aquí para que puedas usarlos en tu juego.



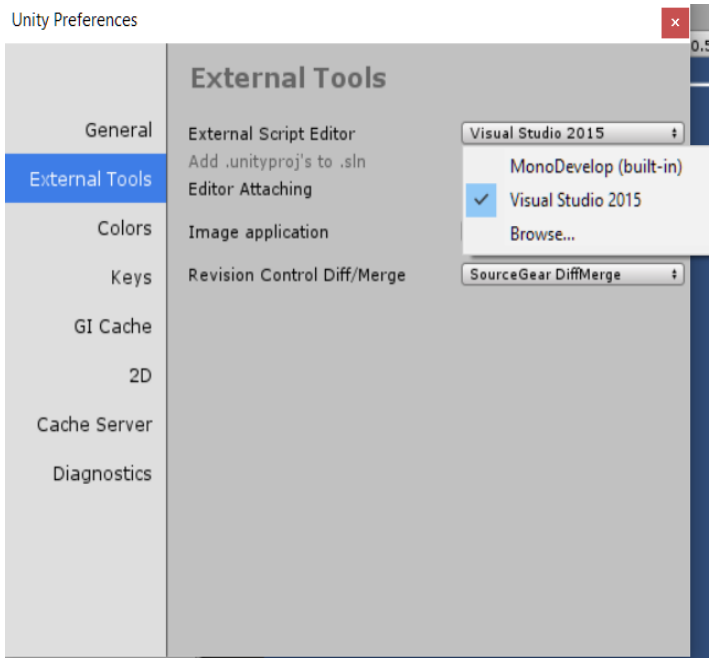
• Ya que un juego normal contendrá varias escenas y una gran cantidad de assets es una buena idea estructurar la librería en diferentes carpetas que harán que nuestros assets se encuentren organizados y sea más fácil trabajar con ellos.

4. Unity.

4.3 La interfaz de usuario de Unity.

5. Vista de Inspector.

• La vista de inspector sirve para varias cosas. Si seleccionas objetos entonces mostrará las propiedades de ese objeto donde puedes personalizar varias características del objeto. También contiene la configuración para ciertas herramientas como la herramienta de terrenos si tenemos el terreno seleccionado. Los scripts son escritos usando editores como MonoDevelop (incluido en la instalación) o Visual Studio. Puedes elegir que editor utilizar desde el menú Edit -> Preferences.



• Unity detectará qué editores compatibles tienes instalados. Cuando creamos un script, podemos saltar al editor de scripts desde Unity, guardar el script y usarlo en el juego.

4. Unity.

4.3 La interfaz de usuario de Unity.

• Al igual que muchas aplicaciones podemos modificar la interfaz por defecto de Unity. Prueba a hacer clic sobre la pestaña de la vista de juego y arrastrarlo al lado de la pestaña de la vista de escena, verás que aparece un indicador de posición. Suelta el botón del ratón y la vista de juego no estará en la misma posición que la vista de escena, estará a un lado. Ahora podremos cambiar entre la vista de escena y la de juego al hacer click sobre la pestaña apropiada. Esto te dará mucho más espacio para la vista de escena, lo que resulta muy útil.

• Cuando trabajemos con la vista de escena es útil también saber que pulsar el espacio agrandará la vista de escena hasta completar el editor.

4. Unity.

4.4 Vista de escena.

• La vista de escena es un entorno 3D para crear cada escena. Trabajar con la vista de escena, en la forma más sencilla, sería arrastrar un objeto desde la vista de proyecto a la vista de escena que colocará el objeto en la escena; entonces podrás posicionarlo, escalarlo y rotarlo sin salir de la vista de escena.

• La vista de escena es también el lugar donde editas los terrenos (esculpiéndolos, pintando texturas y colocando elementos), colocas luces y cámaras y otros objetos.

4. Unity.

4.4 Vista de escena.

Modos de visualización.

• Por defecto la vista de escena tiene una perspectiva 3D de la escena. Podemos cambiar esto por un número de vistas ortográficas: top down, side y front. En la parte derecha de la vista de escena veréis un "Gizmo" que parece una caja con conos que salen de ella.

Podemos usar este Gizmo para cambiar la perspectiva:

- Hacer clic sobre la caja nos llevará al modo perspectiva.
- Hacer clic en el cono verde (y) nos llevará al modo Top-Down.
- Hacer clic sobre el cono rojo (x) nos llevará al modo Side (derecha).
- Hacer clic sobre el icono azul (z) nos llevará al modo Front (frontal).
- También tenemos 3 conos grises que nos llevarán a los siguientes modos: "Back, Left y Bottom", en castellano, Atrás, Izquierda y Abajo.

4. Unity.

4.4.1. Configuración de la visualización.

En la esquina izquierda de la vista de escena encontraremos un conjunto de botones para cambiar la configuración general de la visualización. Vamos a ver cada uno de ellos, de izquierda a derecha.

- Render Mode: la primera opción es Render mode o modo de renderización en castellano. Por defecto aparecerá en “Shaded”. Si hacemos clic aparecerá una lista desplegable con un número de diferentes opciones de renderizado.
    - Shaded: las texturas se renderizan en la vista.
    - Wireframe: las superficies no se renderizan, solo vemos la malla.
    - Shaded Wireframe: las texturas se renderizan, pero también vemos la malla.
- 2D: la segunda opción es el modo de visualización, aparecerá por defecto en 2D si nuestro proyecto lo hemos creado para 2D. Podemos seleccionar el tipo de vista entre 2D y 3D.

4. Unity.

4.4.1. Configuración de la visualización.

- Interruptor de luces: el siguiente botón enciende o apaga la iluminación del escenario. Apagar la iluminación resultará en una escena mostrada sin luces; lo que puede ser útil para el rendimiento y también si no hay luces en la escena.
- Interruptor de sonido: permite encender y apagar el sonido.
- Interruptor de skybox, lense flare y niebla: el último botón activa y desactiva estos tres efectos. Esta opción es útil para desactivar los efectos por razones de rendimiento o visibilidad al trabajar sobre una escena.
- Gizmos: permite visualizar en pantalla elementos que no van a ser mostrados durante la ejecución en la plataforma de destino, por ejemplo, un símbolo donde se reproduzca un sonido, el icono de la cámara, los colliders, etc…son elementos que nos sirven para depurar durante la ejecución.
- Buscador: nos permite buscar un elemento dentro de la escena.

4. Unity.

4.4.2. Botones de Control.

- Debajo de las opciones de visualización verás una fila con 4 botones. 

- Puedes usar Q, W, E, R para alternar entre cada uno de los controles, que detallamos debajo:

- Hand Tool (Q): Este control nos permite movernos alrededor en la vista de escena. Mantener ALT nos permitirá rotar, COMMAND/CTRL nos permitirá hacer zoom y SHIFT incrementa la velocidad de movimiento mientras usas la herramienta.

    - Translate Tool (W): nos permite mover cualquier objeto seleccionado en la escena en los ejes X, Y y Z.

    - Rotate Tool (E): nos permite rotar cualquier objeto seleccionado en la escena.

    - Scale Tool (R): nos permite escalar cualquier objeto seleccionado en la escena.

- Editor de vértices, nos permite deformar una malla a través del editor. 

4. Unity.

4.5. Vista de Proyecto.

• La vista de proyecto es esencialmente una librería de assets para el proyecto de nuestro juego. Todos los componentes del juego que crees desde el editor y todos los objetos que importes como modelos 3D, texturas, efectos de sonido, música etc. se guardaran ahí.

• Como este panel contiene todos los assets de tu juego, y no solo los que están en la escena actual, es importante mantener una buena estructura. Podemos crear carpetas y colocar los objetos dentro de esas carpetas para crear una jerarquía de carpetas.

4. Unity.

4.5. Vista de Proyecto.

• No crees la estructura de tu librería o muevas assets usando Windows Explorer o Finder ya que Unity puede perder enlaces y dependencias importantes. Como práctica general debemos usar las herramientas de organización de la vista de proyecto para crear la estructura y organizar los assets puesto que Unity seguirá su localización.

• En la parte superior de la vista de proyecto veras un botón “Create”, que mostrará una lista desplegable con varias opciones de creación. Podremos crear carpetas, scripts, shaders, animaciones y otros tipos de objetos usando este panel.

• Hacer dos clics sobre un ítem en la librería nos permitirá renombrarlo.

• Puedes hacer clic y arrastrar carpetas e ítems para organizar la estructura.

• Puedes importar y exportar paquetes (colecciones de assets) simplemente haciendo clic derecho sobre la vista de proyecto y seleccionando la opción apropiada.

• Puedes importar assets como archivos de audio, texturas modelos etc. con hacer clic derecho y seleccionar “Import Asset”.

4. Unity.

4.6. Vista de Jerarquía.

• La vista de jerarquía contiene una lista de todos los objetos usados en la escena actual. Cualquier objeto que coloques en la escena aparecerá como una entrada en la jerarquía.

• La jerarquía también sirve como método rápido y fácil para seleccionar objetos en la escena. Si quieres por ejemplo, seleccionar un objeto de la escena, puedes seleccionarlo desde la jerarquía en lugar de moverte por la escena, encontrarlo y seleccionarlo.

• Cuando un objeto es seleccionado en la jerarquía también lo es en la vista de escena, donde puedes moverlo, escalarlo, rotarlo, borrarlo o editarlo. El inspector también mostrará las propiedades del objeto seleccionado; de esta forma la jerarquía sirve como una herramienta útil para seleccionar rápidamente objetos y editar sus propiedades.

4. Unity.

4.8. Vista de Juego.

• En Unity puedes ejecutar tu juego sin salir del editor, lo que es una bendición para los diseñadores que están construyendo niveles y los desarrolladores que están añadiendo nuevas mecánicas de juego.

• La imagen sobre estas líneas muestra los controles de reproducción, que están localizados en la parte superior del editor. Puedes entrar en la previsualización del juego en cualquier momento pulsando el botón de reproducción (el primero por la izquierda), pausar usando el botón de pausa (central) o saltar adelante usando el botón derecho.

• Puedes jugar desde la vista de juego o extenderla a pantalla completa.

• El panel también tiene un menú contextual en la esquina izquierda que aparece como “Free Aspect” por defecto, de esta lista podremos seleccionar un número de proporciones para el juego, lo que es ideal para probar nuestro juego en distintas pantallas y plataformas.

• Ten cuidado al realizar cambios estando en modo juego, por ejemplo añadir objetos a la escena, cambiar valores en el inspector, etc…porque al parar el modo juego estos cambios desaparecerán.

5. Scripting.

En este apartado veremos algunos de las operaciones más habituales a realizar desde los scripts C#.

- C# es un lenguaje muy similar a java, no te costará nada acostumbrarte.
  - Tipos básicos: int, double, bool, string…
  - Arrays: int[] enteros = new int[7] {0,0,0,0,0,0,0};
  - Cada Script genera una clase con el nombre del fichero, que hereda de MonoBehaviour :

```
public class nombreClase : MonoBehaviour {
```

  - Las funciones son muy similares a java :

```
void Start () {
```

  - En vez de import utilizamos using : using System;

5. Scripting.

- Obtener la referencia a un objeto de la escena :

```
GameObject miObjeto = GameObject.Find("objetoEnEscena”);
```

- Mover un objeto: utilizamos el componente transform del objeto y se utiliza un vector de 3 coordenadas (x,y,z). Por ejemplo lo podemos situar en la punto central de la escena.

```
miobjeto.gameObject.transform.position = new Vector3(0, 0, 0);
```

- Acceso a características de un objeto: a través de GetComponent podemos acceder a los componentes del objeto (se pueden consultar en el inspector) por ejemplo el RigidBody.

```
miObjeto.GetComponent<Rigidbody>().collisionDetectionMode =

CollisionDetectionMode.ContinuousDynamic;
```

5. Scripting.

- Instanciar un objeto en tiempo de ejecución: para ello necesitamos la referencia al objeto, la posición donde queremos que aparezca y la rotación.

```
//primero cargamos el objeto desde la carpeta Resource

GameObject miObjeto = (GameObject)Resources.Load("objeto");

Instantiate(miObjeto, new Vector3(0,0,0), Quaternion.identity);
```

- Destruir un objeto.

```
DestroyObject(miObjeto.gameObject);
```

- Control del tiempo: podemos realizar distintas operaciones con el objeto Time, por ejemplo pausar la ejecución.

```
Time.timeScale = 0;
```

5. Scripting.

- Carga de una escena.

```
SceneManager.LoadScene(“nombreEscena”);
```

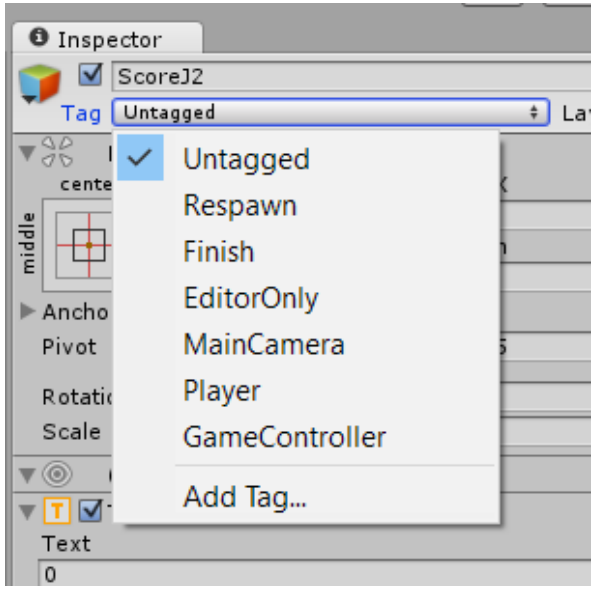
- Acceso a todos los objetos con un tag determinado.

```
GameObject contenedor_objetos = GameObject.FindGameObjectWithTag("miAsset");

foreach (Transform objeto in contenedor_objetos.transform){

 …

 }
```



- Los tags pueden asignarse a los objetos dentro de la escena de forma que un grupo de objetos compartan el mismo tag.

5. Scripting.

- Preferencias de usuario PlayerPrefs: podemos guardar valores entre ejecuciones del juego, por ejemplo volumen del sonido, puntuaciones, etc… para ello se utiliza la clase PlayerPrefs.

```
PlayerPrefs.SetString("nombreJugador", “pepe”);

….

String miJugador = PlayerPrefs.GetString("nombreJugador");
```

5. Scripting.

- Acceso al script de otro objeto de la escena: puede ser necesario acceder al script asociado a otro objeto en cuyo caso.

```
private Script_OtroObjeto script_otroObjeto;

//obtenemos la referencia al objeto dentro de la escena

GameObject otroObjeto = GameObject.Find("otroObjeto");

//obtenemos la referencia al script dentro del objeto

script_otroObjeto = otroObjeto.GetComponent<Script_OtroObjeto>();
```