Programación Visual

¿Qué es la programación visual?

En la programación visual, los elementos del lenguaje de programación están disponibles en forma de bloques diseñados de manera gráfica, por lo que también se la llama programación gráfica. La apariencia y el etiquetado de los módulos permite identificar qué tarea en el flujo del programa pueden resolver. Los pictogramas sirven para orientar al usuario. Así, no se necesitan estructuras muy complejas ni un alto grado de abstracción.

¿En qué consiste la programación visual?

Es un tipo de lenguaje que utiliza componentes gráficos como iconos, botones y símbolos en forma de codificación. Este lenguaje de programación permite ilustrar visualmente el concepto de codificación generado por el ordenador. Utilizar un lenguaje de programación visual en el desarrollo de software tiene sus ventajas. El desarrollo de software moderno como herramienta de programación visual es una solución amigable para los usuarios que no son expertos en codificación. Debido a su relativa simplicidad, la programación visual es una forma perfecta de introducir a los usuarios en la codificación y la programación.
Sin embargo, a pesar de la simplicidad del lenguaje, puede resultar engorroso, ya que es rico en gráficos. Este lenguaje de programación es de mayor tamaño y, por tanto, ocupa más espacio en un ordenador, lo que puede provocar una ralentización de las funciones debido a la cantidad de memoria que requiere en una unidad.

Programación visual orientado a objetos

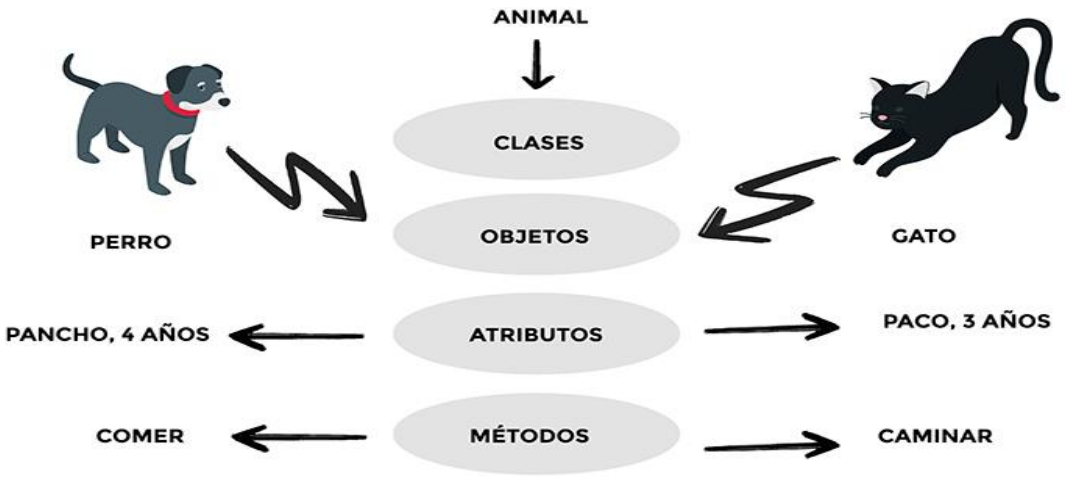
La Programación Orientada a Objetos es un paradigma de programación, es decir, un modelo o un estilo de programación que nos da unas guías sobre cómo trabajar con él. Se basa en el concepto de clases y objetos. Lenguajes secuenciales como COBOL o procedimentales como Basic o C, se centraban más en la lógica que en los datos. Otros más modernos como Java, C# y Python, utilizan paradigmas para definir los programas, siendo la Programación Orientada a Objetos la más popular.
Un programador diseña un programa de software organizando piezas de información y

comportamientos relacionados en una plantilla llamada clase. Luego, se crean objetos individuales a partir de la plantilla de clase.

¿Cómo se crean los programas orientados a objetos?

consistiría en hacer clases y crear objetos a partir de estas clases. Las clases forman el modelo a partir del que se estructuran los datos y los comportamientos.
El primer y más importante concepto de la POO es la distinción entre clase y objeto.
Una clase es una plantilla. Define de manera genérica cómo van a ser los objetos de un determinado tipo. Por ejemplo, una clase para representar a animales puede llamarse ‘animal’ y tener una serie de atributos, como ‘nombre’ o ‘edad’ (que normalmente son propiedades), y una serie con los comportamientos que estos pueden tener, como caminar o comer, y que a su vez se implementan como métodos de la clase (funciones).

Un ejemplo sencillo de un objeto, como decíamos antes, podría ser un animal. Un animal tiene una edad, por lo que creamos un nuevo atributo de ‘edad’ y, además, puede envejecer, por lo que definimos un nuevo método. Datos y lógica. Esto es lo que se define en muchos programas como la definición de una clase, que es la definición global y genérica de muchos objetos.



Con la clase se pueden crear instancias de un objeto, cada uno de ellos con sus atributos definidos de forma independiente.

Principios de la programación orientada a objetos

Encapsulación

La encapsulación contiene toda la información importante de un objeto dentro del mismo y solo expone la información seleccionada al mundo exterior. Esta propiedad permite asegurar que la información de un objeto esté oculta para el mundo exterior, agrupando en una Clase las características o atributos que cuentan con un acceso privado, y los comportamientos o métodos que presentan un acceso público. La encapsulación de cada objeto es responsable de su propia información y de su propio estado. La única forma en la que este se puede modificar es mediante los propios métodos del objeto.

Por lo tanto, los atributos internos de un objeto deberían ser inaccesibles desde fuera, pudiéndose modificar sólo llamando a las funciones correspondientes. Usamos de ejemplo un coche para explicar la encapsulación.

Abstracción

La abstracción es cuando el usuario interactúa solo con los atributos y métodos seleccionados de un objeto, utilizando herramientas simplificadas de alto nivel para acceder a un objeto complejo. En la programación orientada a objetos, los programas suelen ser muy grandes y los objetos se comunican mucho entre sí. Los objetos y las clases representan código subyacente, ocultando los detalles complejos al usuario.

Herencia

La herencia define relaciones jerárquicas entre clases, de forma que atributos y métodos comunes puedan ser reutilizados. Las clases principales extienden atributos y comportamientos a las clases secundarias. A través de la definición en una clase de los atributos y comportamientos básicos, se pueden crear clases secundarias, ampliando así la funcionalidad de la clase principal y agregando atributos y comportamientos adicionales. Los diferentes tipos de animales necesitarán diferentes métodos, por

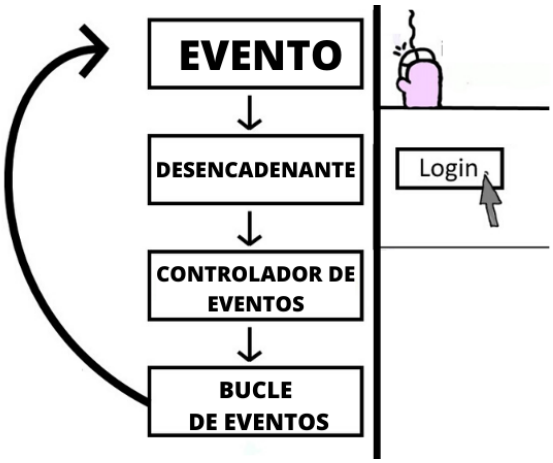
ejemplo, las aves deben poder poner huevos y los peces, nadan.

Polimorfismo

El polimorfismo consiste en diseñar objetos para compartir comportamientos, lo que nos permite procesar objetos de diferentes maneras. Al utilizar la herencia, los objetos pueden anular los comportamientos principales compartidos, con comportamientos secundarios específicos. Alrededor de estos principios de la programación orientada a objetos se construyen muchas cosas.

Características y aplicaciones de eventos

La programación orientada a eventos se refiere a un modelo de la programación de computadoras, donde se utilizan los eventos que suceden para la determinación del flujo de control de un programa.
Básicamente, separa la lógica de procesamiento de eventos del resto del código de un programa.
Teóricamente, el estilo de esta programación es compatible con todos los lenguajes de programación, aunque puede ser diferente en la forma de implementarse.



Métodos visuales y no visuales

Se pueden establecer muchas clasificaciones para los componentes. Una de ellas es la de visuales o controles, frente a no visuales. Un componente es visual cuando tiene una

representación gráfica en tiempo de diseño y ejecución (botones, barras de crol, cuadros de edición, etc.), y se dice no visual en caso contrario (temporizadores, cuadros de diálogo -no visibles en la fase de dieño-, etc). Por lo demás no existen más diferencias entre ellos, excepto, claro está, las derivadas de la visualización del componente.

Procesos de desarrollo visual en proyectos distribuidos y de escritorio

Un proceso de desarrollo de software es la descripción de una secuencia de actividades que deben ser seguida por un equipo de trabajadores para generar un conjunto coherente de productos, uno de los cuales en el programa del sistema deseado. El objetivo de un proceso de desarrollo de programas es la formalización de las actividades relacionadas con el desarrollo del software de un sistema informático.

Los elementos básicos de un proceso de desarrollo de software es definir los papeles que juegan los trabajadores, las actividades que desarrollan y los productos que deben generarse. En un plan de desarrollo cada trabajador debe tener su papel dentro de él, lo que define las actividades que debe realizar y los productos que debe generar. Los productos son los documentos o información que debe ser creada como consecuencia de la actividad que se desarrolla. Cada actividad debe tener siempre como principal objetivo generar ciertos productos bien definidos y especificados.

Requerimientos visuales de proyectos distribuidos y de escritorio

Un proceso de diseño de proyectos exitoso se desarrolla en 7 pasos. Estos incluyen desde la definición de metas y objetivos básicos hasta la consolidación de tu estrategia. Esto te permitirá mantenerte organizado al momento de gestionar un proyecto nuevo.

Herramientas y lenguajes de programación visual

C# / ASP.NET Core

ASP.NET Core es un marco multiplataforma gratuito de **código abierto** y de alto rendimiento que tiene como finalidad compilar modernas aplicaciones conectadas a Internet y basadas en la nube. Gracias a ASP.NET Core puede compilar servicios y aplicaciones web, aplicaciones de IoT y servicios de back-end móviles. Use sus herramientas de desarrollo favoritas en Windows, macOS y Linux. Implemente el contenido en la nube o de forma local a la vez que usa los lenguajes que ya conoce: HTML y JavaScript. Aproveche las ventajas que le ofrece la sintaxis Razor de ASP.NET para alinear C# directamente en sus vistas. Use TypeScript para crear código escrito, escalable y más manejable que se compile en JavaScript.

JavaScript

El editor de JavaScript en Visual Studio es compatible con EcmaScript 6 y tiene el motor IntelliSense más avanzado del mercado. JavaScript es un lenguaje de primera clase en Visual Studio. Puede usar la mayoría de las ayudas de edición estándar (fragmentos de código, IntelliSense, etc.) al escribir código JavaScript en el IDE de Visual Studio.

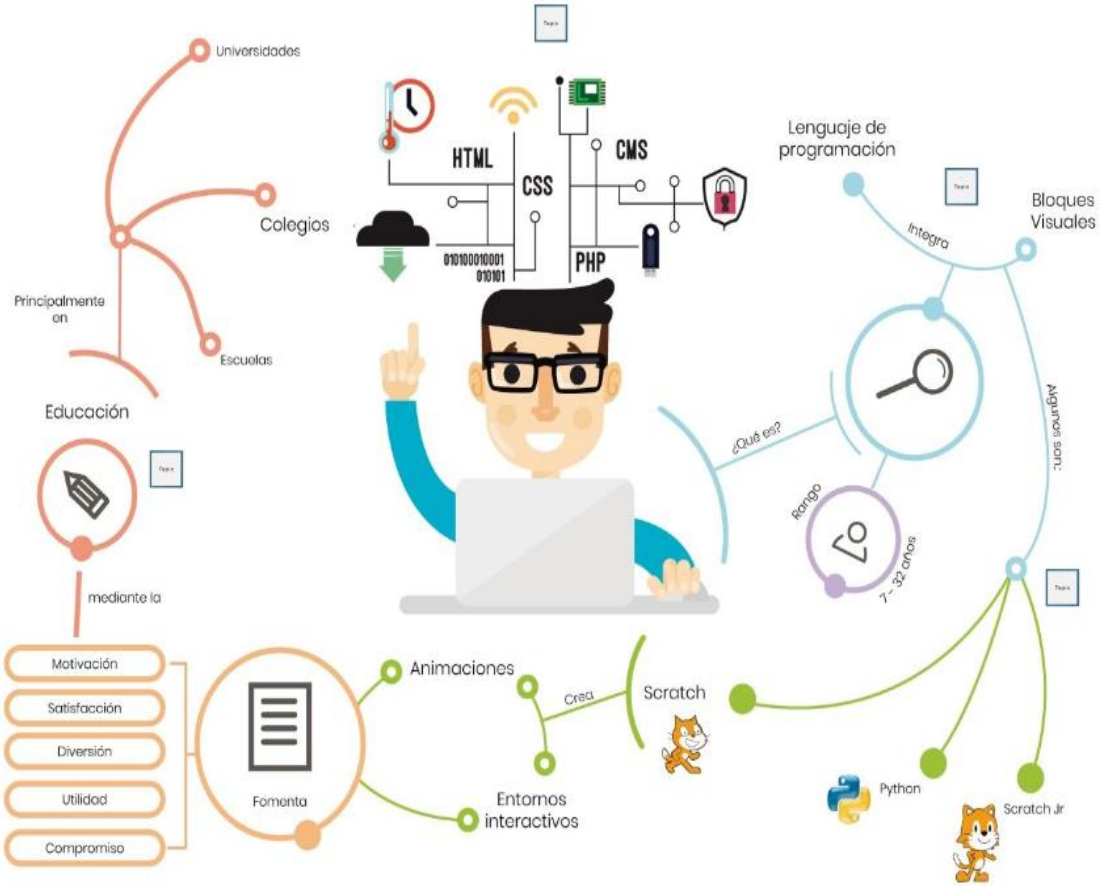
CSS, Less, Sass y Font Awesome en ASP.NET Core

La mayoría de sitios comunes se benefician de la posibilidad de definir y mantener de manera eficaz los estilos y los archivos de hojas de estilo (CSS), además de tener un fácil acceso a los iconos sin imagen que ayudan a que la interfaz del sitio sea más intuitiva. Ahí es donde entran los lenguajes y herramientas que admiten **Less** y **Sass**, y bibliotecas como **Font Awesome**.

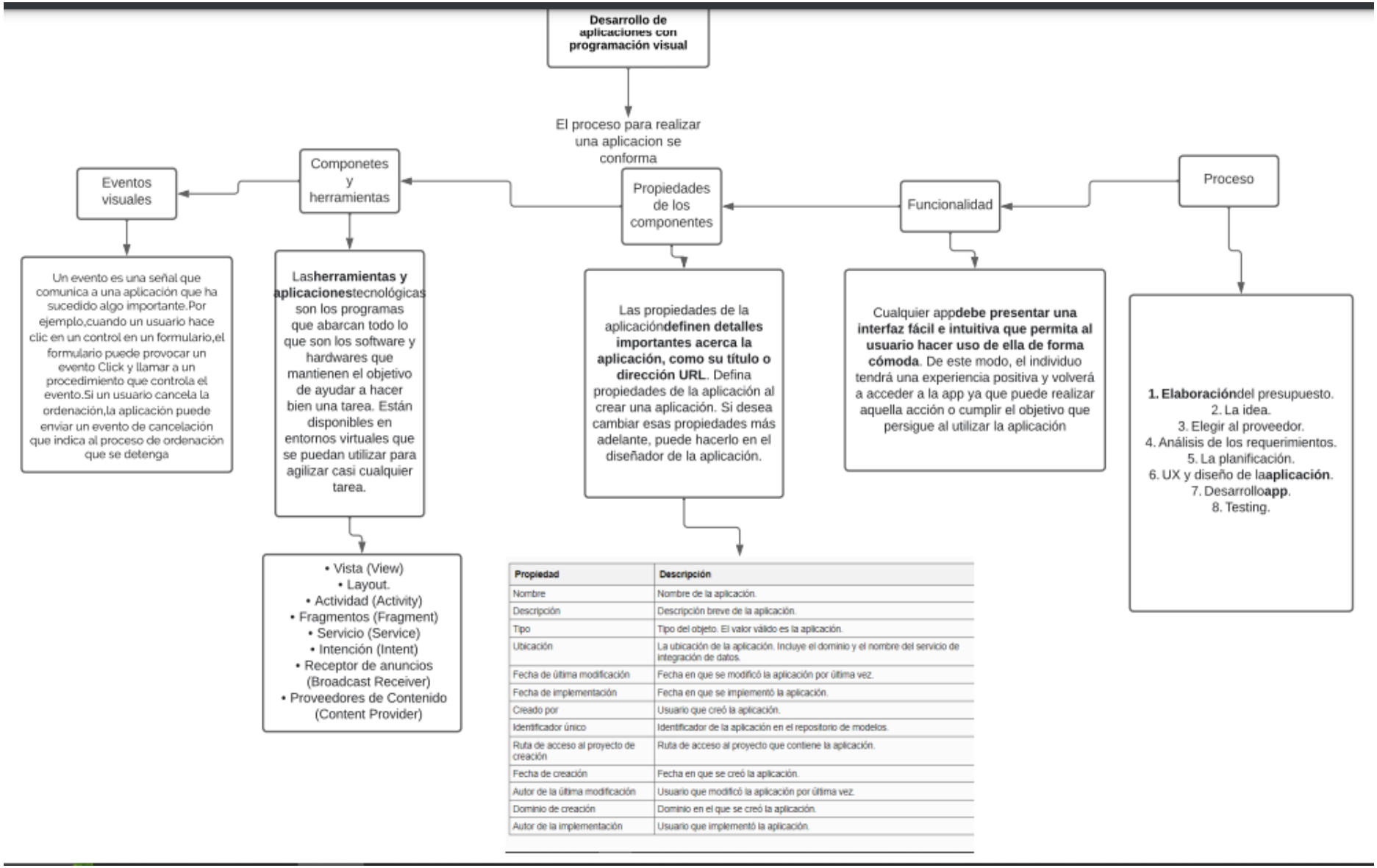
Referencias

- https://asana.com/es/resources/project-design
- https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/programacion-visual/
- https://elvex.ugr.es/decsai/builder/intro/3.html
- https://www.ctr.unican.es/asignaturas/MC_OO/Doc/OO_08_I2_Proceso.pdf
- https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/
- https://visualstudio.microsoft.com/es/vs/features/web/languages/

Mapa mental Diseño de programación visual



Mapa conceptual Desarrollo de aplicaciones con programación visual



Concepto de videojuegos

Tipos de game designer, storyboard

Game Designer

Como bien dice su nombre, un Game Designer o diseñador de videojuegos es el profesional encargado de diseñar todos los elementos que componen el juego, desde el concepto, las mecánicas o los niveles, entre otros. No es estrictamente necesario que sea programador, ni ser un diseñador gráfico o artista conceptual.

Storyboard

El Storyboard no es otra cosa que la realización de una secuencia de dibujos realizados en una plantilla, acompañados de textos breves que definen la estructura de la historia. Un proceso imprescindible en la fase de preproducción para el desarrollo de Videojuegos que permite previsualizar el resultado final de la producción.
¿Por qué es importante el Storyboard en el Diseño de Videojuegos?
El uso del Storyboard tiene su origen en el cine de Animación, cuando la compañía Walt Disney comenzó a usarlo en sus películas en el año 1930. Su uso se popularizo 10 años más tarde y, se extendió entre otras compañías del sector del entretenimiento, entre ellas el sector de los videojuegos.

La creación de un Storyboard para videojuegos es muy similar al que se emplea en otros sectores como publicidad, cine o televisión. Sin embargo, difiere en un elemento principal: A diferencia de otros sectores donde solo hay una línea de acción, normalmente establecida por el director del guion, el diseño de un videojuego requiere que el Storyboard cuente con diferentes líneas de acción, en función de los movimientos del jugador.
Su objetivo principal es hacer el videojuego inmersivo, estableciendo la interacción que el jugador tendrá con los diferentes niveles, objetivos y fases del juego.

Tipos de Storyboards y técnicas

Veamos cómo son los principales Storyboards para este fin:

- En el Diseño del juego: Aquí se establece una plantilla principal donde se refleja en las diferentes cuadriculas la estructura base del juego, la vista previa de todos los niveles que conformarán el juego.
- En el Diseño de niveles: A partir del storyboard creado anteriormente, creamos un Storyboard para cada uno de niveles del juego por separado detallando las diferentes acciones, escenas y objetivos del juego. Su función es separar y distinguir cada nivel.
- En el Diseño de escenas, objetos y personajes: Realizamos el Storyboard de los personajes, objetos y entornos del juego para describir como interactúan entre sí. El Storyboard de un personaje es muy importante, ya que los jugadores serán los que interactuarán con estos personajes durante el juego.

Se trata de una forma muy útil de previsualizar y estructurar el contenido de un juego y, se puede realizar con técnicas de dibujo tradicionales o digitales:

Tradicional: Pueden ser en blanco y negro, a color, bocetos e incluso ilustraciones más elaboradas, todo en función del artista y las necesidades de la producción.
Digital: Mediante softwares que disponen de herramientas para la creación de Storyboards como es el caso de Toon Boon Storyboard Pro ,software estándar en la industria, que permite planificar la animación 2D y la creación de recursos para juegos.

Motores de videojuegos y lenguajes de videojuegos

Un motor de videojuego es un término que hace referencia a una serie de librerías de programación que permiten el diseño, la creación y la representación de un videojuego.
El aspecto más destacado a la hora de elegir un motor de videojuegos entre todos los disponibles que hay en el mercado son las capacidades gráficas, ya que son las encargadas de mostrar las imágenes 2D y 3D en pantalla, así como calcular algunos aspectos como los polígonos, la iluminación, las texturas … Otras características para tener en cuenta a la hora de la elección son la facilidad de aprender a usar el motor de videojuegos y la facilidad para exportar el juego a diferentes plataformas.
Algunas de las funcionalidades más importantes son:

El motor de físicas

El motor de físicas es el que hace posible aplicar aproximaciones físicas a los videojuegos para que tengan una sensación más realista en la interacción de los objetos con el entorno. En otras palabras, es el encargado de realizar los cálculos necesarios para que un objeto simule tener atributos físicos como peso, volumen, aceleración, gravedad …

El motor de sonido

Los sonidos y la banda sonora de un videojuego es también una parte muy importante. El motor de sonidos es el encargado de cargar pistas, modificar su tasa de bits, quitarlas de reproducción, sincronizarlas entre otras cosas.

El scripting

Todos los motores de videojuegos tienen un lenguaje de programación que permite implementar el funcionamiento de los personajes y objetos que forman parte del videojuego.

Dentro de las diferentes opciones de motores de videojuegos podemos distinguirlos en populares y motores propietarios o privados que son los creados por empresas importante de videojuegos para diseñar sus títulos más populares.

Los motores populares más utilizados y que más posibilidades dan al desarrollador son:

- **Unreal Engine:** Fue creado por Epic Games en 1998. En 2012 se presentó Unreal Engine 4, una nueva versión del motor. Entre las empresas que lo utilizan se encuentran Electronic Arts y Ubisoft. Utiliza el lenguaje de programación C++.

- **Unity 3D:** Se trata de una de las innovaciones más importantes creadas por la comunidad científica y de videojuegos y permite jugar a complejos videojuegos en 3D sin necesidad de instalarlos en el ordenador. Los videojuegos creados con el motor Unity 3D se pueden jugar en un navegador con el reproductor Unity Web Player, eliminando la necesidad de instalar el videojuego.

Los motores propietarios o profesionales más conocidos por los juegos que han sido diseñados con ellos:

- **Frostbite Engine:** Este motor para videojuegos creado por Digital Illusions CE se utiliza para crear videojuegos de acción en primera persona. Se presentó principalmente para la serie de videojuegos Battlefield. Ha jugado un papel fundamental en prácticamente todos los videojuegos de EA. L a nueva versión del motor Frostbite Engine es Frostbite 3.

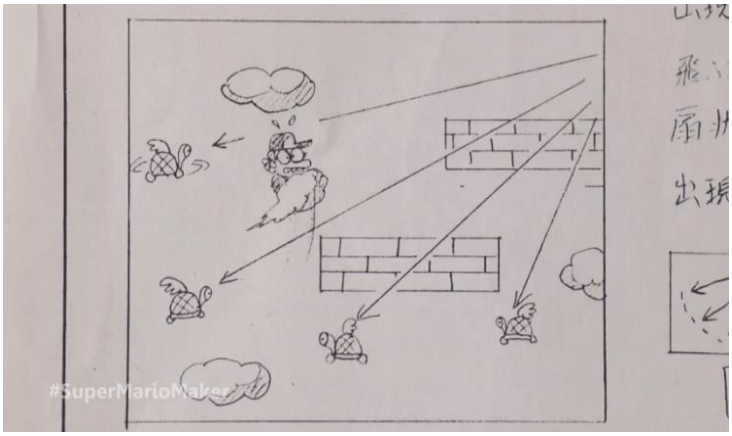
- **Decima Engine:** Alberga herramientas y características para crear inteligencia artificial, física, lógica y mundos en el desarrollo, así como compatibilidad con 4K y HDR.

- **Luminous Studio:** Es un motor de videojuegos multiplataforma desarrollado y usado internamente por Square Enix. Con este motor se desarrolla el juego Final Fantasy.

Metodologias de desarrollo de videojuegos

Fase de Concepción

Todo comienza con una idea a partir de la cual se conformarán los aspectos fundamentales. Se determina el género o géneros del videojuego, cómo será el proceso de juego (game play), y también se constituye un guion gráfico (story board) en el que se tratan todo tipo de ideas preconcebidas que pueden ir adaptándose, como por ejemplo el estilo de los personajes, el ambiente, la música, etc. Una vez se sabe qué hacer entonces es el momento de diseñar.



Viñeta de un Storyboard de los primeros Super Mario

Fase de Diseño

Se empieza definiendo los elementos que componen el juego. Se desarrolla la historia, se crean bocetos de guiones para determinar los objetivos, se deciden los personajes principales, el contexto, etc.

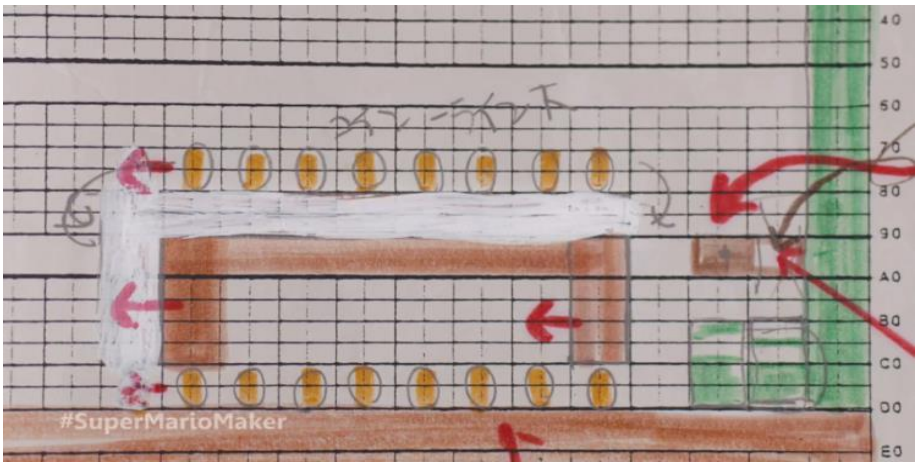
Utilizando estos esbozos de guiones los artistas se ponen manos a la obra para crear conceptos del aspecto del juego, la forma en que se visualizarán los personajes, los escenarios, objetos, etc. Su trabajo es presentar propuestas visuales para ir dando forma a la idea original.

También se describen los elementos sonoros de los que consta el juego: efectos de sonidos, ambientación, música, voces, etc. Aunque todavía no se compone ni se graba nada.

Paralelamente se especifica el funcionamiento general del videojuego, algo que depende del género, ya que señalan la forma en que las entidades virtuales interactúan dentro del juego.

Finalmente, con una idea algo más clara del rumbo que tomará el juego, se hace el diseño de la programación, que describe la manera en la que se implementará el videojuego, el lenguaje o lenguajes de programación que se utilizarán, las metodologías que se seguirán, etc.

Todo lo anterior tendrá como objetivo generar el Documento de Diseño que especificará el desarrollo del arte, las mecánicas y la programación del videojuego.



Diseño original de un escenario de Super Mario

Fase de Planificación

Esta etapa tiene como objetivo identificar las diferentes tareas para desarrollar el videojuego. Se reparte el trabajo entre los distintos componentes del equipo de desarrollo, se fijan plazos de entregas, se planifican reuniones de seguimiento, etc.

NO.	TASK NAME	WEEK4	WEEK5	WEEK6	WEEK7	WEEK8	WEEK9	WEEK10	WEEK11	WEEK12	WEEK13
1	**System Design**										
2	Graphics and User Interface -										
3	Graphics and User Interface -										
4	Graphics and User Interface -										
5	Graphics and User Interface -										
6	Game Dynamics - Generation										
7	Game Dynamics - Motion of										
8	Game Dynamics - Interaction										
9	**Integrate Graphics and UI**										
10	Radar (Sensory) - Enemy										
11	Radar (Sensory) - Trajectory										
12	Radar (Sensory) - Speed										
13	Real-Time Defense Planner (RTDP) - Target Identifier										
14	Real-Time Defense Planner (RTDP) - Strategy Planner										
15	**Integrate Radar and RTDP**										
16	Communication Protocol -										
17	Communication Protocol -										
18	**Integrate Communication**										
19	**Full System Integration**										
20	**Full System Testing**										
21	**Documentation**										
22	**Presentation**										
	Legend										
	Name	Si Shuo	Varsha	Viet	Yinbei	Yin	Zhen Yu	All	Partial		
	Color										

Ejemplo de diagrama de Gantt para planificar un proyecto

Fase de Producción

Una vez se tiene claro lo que hay que hacer, cómo hacerlo, y se ha planificado el tiempo para llevarlo a cabo, entonces se empieza la producción con el objetivo de crear el juego, como mínimo en una versión inicial o prototipo a mejorar gradualmente.

Se llevan por tanto a cabo todas las tareas de la fase de planificación teniendo como guía el documento de diseño: programación, ilustración, desarrollo de interfaces, animación, modelado, desarrollo del sonido, etc.

Si finalmente se logra ensamblar correctamente todas las piezas entonces esta fase culmina (por ahora). Sin embargo, al igual que en el desarrollo de software tradicional, es muy difícil que todo salga bien a la primera, por lo que se entra en una fase para probar a fondo el videojuego.

Oficinas de Ubisoft en Toronto (Canadá)

Fase de Pruebas

En esta etapa se corrigen los errores del proceso de programación y se mejora la jugabilidad a medida que se prueba el juego.

Generalmente encontraremos dos tipos: las pruebas alpha, realizadas por un pequeño grupo de personas generalmente involucradas en el desarrollo, y las pruebas beta, realizadas por un equipo externo de jugadores. Las primeras tienen el objetivo de corregir defectos graves y mejorar características fundamentales no contempladas en el documento de diseño, mientras que las segundas se enfocan en detectar fallos menores y perfilar la experiencia de usuario.

Comparación entre la Alpha y la Beta de Battlefield 3

Fase de Distribución/Márketing

En cuanto a la distribución es el proceso de crear las copias del juego ya finalizado y llevarlo a las tiendas (ya sean físicas o digitales) para que los jugadores puedan comprarlo o hacerse con él.

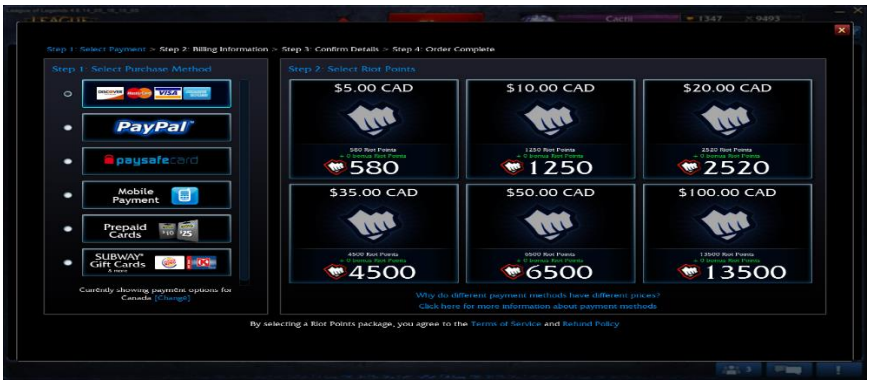
Por otro lado, el márketing es también fundamental para dar a conocer el videojuego y conseguir el mayor número de jugadores posibles. No tiene un orden concreto dentro del desarrollo, pues algunas empresas empiezan a hacer campaña de sus videojuegos meses e incluso años antes de publicarlos. La verdad es que depende de los recursos que los desarrolladores quieran destinar a promocionar la obra y no tiene por qué ser un departamento dentro de la propia empresa, sino que tanto la distribución como el márketing se pueden delegar a empresas externas especialistas en estas áreas.

Vale la pena comentar el fenómeno “hype”, que ocurre cuando una empresa hace uso de una excesiva publicidad para dar a conocer su producto, creando incluso una necesidad inexistente en los potenciales consumidores. Lo malo ocurre cuando el producto no está a la altura de lo prometido y entonces se convierte en el blanco de multitud de críticas en muy poco tiempo, algo que puede perjudicar gravemente la imagen de los creadores.

Fase de Mantenimiento

Pese a que el juego esté finalizado y en las manos de los jugadores, su ciclo de vida aún está lejos de terminar. La fase de mantenimiento es el momento de arreglar nuevos errores, mejorarlo, etc. Esto se hace sacando parches o actualizaciones al mercado.

Sin embargo, es también una oportunidad para seguir sacándole partido. Ya sea en forma de micro transacciones, suscripciones de pago o incluso con expansiones completas que añaden nuevas características al videojuego sin modificar en profundidad el motor del mismo, digamos que sería más o menos como aprovechar al máximo la base inicial.



Proceso de diseño de interfaces de videojuegos en 2d y 3d

¿Qué es el diseño de interfaz para usuario en un videojuego?

Como ya te hemos comentado, la interfaz de usuario en un videojuego es el punto de interacción entre el jugador y el juego. Su objetivo fundamental es el de brindar la información necesaria para que el usuario pueda hacer todo lo que el juego le propone de manera totalmente fluida. Un buen diseño de UI guía de manera directa o intuitiva para que el jugador pueda recorrer el mundo de tu videojuego de forma correcta.

A continuación, te contamos algunas cosas que debes tener en cuenta a la hora de diseñar la interfaz de un videojuego:

- **Entorno/Plataforma.** Lo primero a tener en cuenta es dónde se va a jugar el juego que estás diseñando. Debes tener en cuenta las posibilidades y limitaciones que te ofrece la plataforma. No es lo mismo hacer juegos para smartphones que para una consola o que para un PC.
- **Contenido.** Un buen diseño de UI proporciona al jugador toda la información necesaria para que pueda interactuar con el juego y que todo sea fluido.
- **Diseño Visual.** Los videojuegos, casi siempre, entran por los ojos. Un apartado visual feo o denso en la interfaz del juego puede resultar contraproducente y sacar al jugador de la experiencia inmersiva que quieres proporcionarle. Debes definir el estilo de arte.
- **Arquitectura de la información.** Definir qué elementos son de mayor o menor importancia para el usuario y organizarlos de tal forma que todo resulte en un diseño de interfaz coherente y relevante.

Desarrollo de prototipos de videojuegos

Hay que contemplar muchos factores y posibilidades inciertas para dar con un buen **prototipo**. El **prototipado** de videogames es la clave para poner buen rumbo a su desarrollo.

El desarrollo de videojuegos es una profesión **increíblemente apasionante**, **enriquecedora** y a la vez, **compleja**. Con su nacimiento en la década de los ’70, algunos, hemos tenido el privilegio de vivir su evolución.

En sus inicios, cuando con la **Commodore 64** o la **Texas TI-99/4A**, con una pantalla vectorial parpadeante que nos hacían divertir con naves destruyendo asteroides o alienígenas, que apenas se distinguían, o unos rectángulos blancos que representaban jugadores de ping-pong; hasta hoy donde nos podemos sumergir en distintos mundos y vivir historias increíbles, gracias a la ‘Virtual Reality’, con unas gráficas que nada tienen que ver con las de los inicios de esta industria y lo que aún nos queda por disfrutar,

gracias a personas como tú que le apasiona este intricado mundo del desarrollo y creación de videojuegos.

Con una evolución continua de casi 40 años, los videojuegos han pasado de pequeñas maravillas de la ingeniería informática a grandes proyectos construidos por centenares de profesionales durante más de dos años. La complejidad, extensión de contenidos y riqueza de detalles de un videojuego ambicioso requieren hoy grandes equipos de trabajo especializado, con personal dedicado íntegramente a gestionar las tareas de producción y desarrollo.

Ahora bien, ya hemos dado la introducción y es este el momento de comenzar con el punto que nos ocupa hoy que es:

El Prototipado.

Como ya dijimos anteriormente, éste es una **fórmula incierta**, ya que el resultado de tantos esfuerzos hercúleos, pueden dar como resultado juegos maravillosos como **Red Dead Redemption**, **Borderlands 2**, **Horizon Zero Dawn** o **The Last of Us**. Por desgracia, muchas veces nos encontramos con juegos como **Aliens Colonial Marines** que, a pesar de contar con grandes presupuestos y equipos experimentados, fracasan estrepitosamente, con severas críticas de parte de la audiencia y de la crítica. También tenemos el caso de excelentes juegos como **BioShock Infinite** que, a pesar de su calidad, han necesitado mucho tiempo para su desarrollo que los ingresos por ventas no lo han podido compensar y se han llevado por delante al estudio entero.

Esto no sólo ocurre para grandes juegos de consola, también se aplica a los microgames creados para las **App Stores** de nuestros dispositivos digitales que no sólo tienen videogames de cuestionable calidad, que con frecuencia descartamos apenas unos minutos después de descargarlos, a juegos que, por su coste de desarrollo, no puede cubrirlos y les ocasionan cuantiosas pérdidas a sus creadores.

Además de trabajar en cada una de las áreas del desarrollo, el conjunto de la creación del videogame, ha de mantener coherencia, esta es **la razón** de ser del **prototipado** en el **diseño del código**, la **jugabilidad** y el **aspecto audiovisual**.

Aunque en muchos casos los pequeños desarrolladores noveles crean juegos deprisa y corriendo, también nos encontramos habitualmente con juegos que pretendían ser muy buenos, pero no lo consiguieron. **¿Por qué ocurre esto?** y **¿Qué pueden hacer los desarrolladores para evitarlo?**

Para Avanzar a veces hay que Descartar.

Cualquier videojuego está compuesto de muchos elementos: **modelos en 3D**, **texturas**, **animaciones**, **sonidos**, **música**, **escenarios**, **personajes**, **líneas de guion**, **diálogos**, **puzles**, etc. Por debajo de esos elementos, claramente visibles, hay más partes indispensables para el funcionamiento y atractivo del videogame: **millones de líneas de código de engines 3D**, **motores de física**, **sistemas de carga de datos**, **VR**, **AI**… Uniéndolo todo está el diseño, que da razón y sentido a cada pieza que lo forma: **gameplay**, **mecánicas**, **ritmo**, **dificultad**, **controles**, **repuestas visuales y sonoras**, **recompensas**, **penalizaciones**, **probabilidades**, etc.

Si cuando estamos desarrollando el videojuego nos damos cuenta de que algunas ideas que teníamos al inicio no eran todo lo **divertidas** o **interesantes** que pensamos, necesitamos **adaptarlas**, **modificarlas** o, simplemente **sustituirlas** o **descartarlas**; llagado este punto, aunque resulte difícil, hay que hacerlo. Claro está, que estos cambios o modificaciones seguramente nos obligarán a su vez a retocar otras áreas del videogame, y esto **afecta a todo el equipo** y produciendo la **pérdida de semanas** o **meses de trabajo**.

Los Problemas hay que Solucionarlos.

El **prototipado** nos permite poner a prueba la mecánica del juego en su estado primigenio, lo que permite aumentar la eficiencia en el esfuerzo empleado para el desarrollo.

Si **cometemos errores** y vemos que el juego no es lo amigable que pensábamos o que no produce tanta diversión como creíamos, ya sea porque los combates no tienen una apariencia real o hay que pulsar muchas veces una tecla para que el avatar logre hacer la acción requerida, generalmente es producido por errores que hemos cometido y este es el momento de **modificarlos**, cambiando cámaras, si el error es producto de ellas, retocando los movimientos, etc.

Aunque un juego tenga problemas o errores evidentes, ya sean de **ritmo**, **dificultad**, **claridad**, **rendimiento**, o **audiovisuales**, generalmente puede ser muy difícil detectar la raíz de los mismos, o sus posibles soluciones. Todas las piezas del juego están interconectadas y afectan a la experiencia del jugador. Ni aun teniendo todo el tiempo del mundo para desarrollar nuestro videogame, deberemos probar y a veces a ciegas hasta que se nos presente la solución. Si no queremos quedarnos bloqueados necesitamos poder encontrar, **rápida** y **eficazmente**, los problemas de nuestro juego y solucionarlos con prontitud.

Necesitaremos probar los fundamentos del videojuego lo antes posible, y así, poder experimentar ágilmente con ellos hasta dar con la fórmula definitiva que queremos que haga de nuestro juego una experiencia gloriosa. Para hacer esto, lo ideal, es al inicio, antes de que el juego esté muy desarrollado y los problemas anteriormente descritos nos dificulten mucho seguir avanzando.

La Iteración como Llave Maestra.

El principio o idea fundamental del prototipado es, entonces, hacer el mínimo trabajo necesario para responder al mayor número de preguntas sobre el videojuego. Se trata, en definitiva, de crear una versión de nuestro proyecto que demuestre que las bases

fundamentales del juego están claras y que el resto del tiempo va a consistir "**simplemente**" en dotarlo de contenidos. A partir de esto, el desarrollo puede prolongarse meses o años, pero sabremos que todo el trabajo será útil y encajará de forma eficiente. El prototipado toma diferentes aspectos en distintas disciplinas del desarrollo, y por desgracia todavía hay muchos equipos que no tienen la suficiente experiencia o método de trabajo para realizar un prototipado efectivo. Los patrones principales para tener en mente cuando prototipamos un juego son los siguientes:

Las preguntas concretas que debemos responder. Pueden ser tan generales como: **"¿Es la idea de juego divertida?**", o precisas como: "**¿Puede mi tecnología gráfica representar personajes tan reales?**"

Conseguir los máximos resultados con el mínimo esfuerzo. Reutilizar gráficos y código de proyectos anteriores, y no perder tiempo en nada que no contribuya directamente a responder a las preguntas que tenemos.

Sacar conclusiones de las respuestas y estar dispuestos a abandonar conceptos. Los requisitos de gráficos y programación de un juego comercial son mucho más estrictos que los que exigimos a nuestros prototipos.

Por ello, un **prototipo** con frecuencia resulta una **base poco sólida** para construir el **juego completo**. Lo importante es que tirar y reconstruir los contenidos de un prototipo es más efectivo que hacer piezas completas del juego.

Hay ejemplos de prototipos que se centran en las necesidades de una disciplina concreta, sea **diseño**, **arte** o **programación**, mientras que otros se dedican a validar y precisar la manera en que las necesidades de dos o tres de esas disciplinas encajan entre ellas. Por ejemplo, la programación puede construir prototipos que ayuden a determinar el equilibrio correcto entre la cantidad de personajes simultáneos en pantalla y la cantidad de polígonos que deben tener estos personajes.

El **prototipado** no siempre conduce a buenos resultados. El mayor ejemplo en la historia de los juegos probablemente sea **Spore**, que contó con una extensa fase de prototipado y aun así su desarrollo se alargó hasta 5 años para no conseguir el éxito buscado. Los desarrolladores de Maxis pusieron a disposición del público algunos de los prototipos del juego, que estudiaban conceptos como la formación de una galaxia o la erosión del agua sobre un terreno. Esos prototipos están disponibles para su descarga. Como veis, la creación de un prototipo determina en gran medida el éxito de un videojuego.

Hay tres tipos de prototipado que deberemos hacer para testear en un principio nuestro videogame, y estos son:

- **Prototipo de Código.**
- **Prototipo Artístico.**
- **Prototipo de diseño.**

A continuación, haremos una síntesis de cada uno de ellos, para que los noveles desarrolladores los tengan como guía a la hora de comenzar con esta importante fase en el desarrollo de los videojuegos.

Prototipado del Código.

El equipo de programación de un juego realizará prototipos con varios objetivos diferentes:

Construir experimentos con las nuevas tecnologías que se vayan a utilizar en el proyecto: **engines 3D**, **herramientas**, o incluso **consolas** o **hardware** que todavía no se han comercializado. El **objetivo** es **averiguar** su **potencial** y **posibilidades**. Implementar los conceptos básicos de diseño una vez aclarados, y ayudar a los diseñadores a iterar sobre ellos para afinarlos.

Frecuentemente los programadores realizan prototipos con tecnologías o lenguajes de programación diferentes a los que van a usar en el juego definitivo. Por ejemplo, es posible que el juego se desarrolle usando **Unreal Engine,** pero los prototipos de diseño

y gameplay se desarrollen usando otros sistemas como **Unity** o incluso **HTML5**, que pueden ser más ágiles para trabajar, aunque no sean los más idóneos para el producto final. Un ejemplo interesante fue el de **Demigod**, desarrollado por **Gas Powered Games**. Describen que desarrollaron y afinaron el juego durante meses sin gráficos, con "**cajas blancas**" representando a las unidades y combatientes, y solamente cuando estuvieron contentos con el diseño definitivo empezaron a decorarlo y embellecerlo con el desarrollo de los gráficos.

Prototipado del Arte.

El equipo de **arte gráfico** de un juego necesita al principio afinar con precisión el **estilo artístico**, **estético** y **técnico** de los recursos de '**art'** del **game**. Esto implica tener claro no solamente el aspecto de los personajes escenarios y objetos por separado, sino también la forma en que se integran unos con otros, y los requisitos y limitaciones técnicas con los que se construyen.

Los **prototipos de arte** frecuentemente se construyen con herramientas propias de creación gráfica: **Photoshop**, **DAZ3D Studio**, **Blender**, **Bryce**, **Carrara**, **Maya**, **3ds MAX**, etc. Por ejemplo, un grafista puede construir una secuencia que muestra un efecto visual de lluvia sobre el escenario del juego usando las potentes herramientas de sistemas de partículas de su programa de modelado 3D; y una vez alcanzado el aspecto visual deseado, retocar y reconstruir ese objetivo usando técnicas que estén soportadas por el engine 3D del juego.

De igual manera se realizan prototipos animados para personajes, escenarios, e incluso interfaz gráfico, **animaciones** de logotipos, y cualquier otro elemento que resulte más fácil construir y renderizar en una herramienta estándar que directamente en el juego. Esto facilita las pruebas, proponer y presentar alternativas e ideas diferentes, para elegir la mejor antes de realizar el trabajo completo de integrarla en el juego final.

Prototipo de Diseño.

Aunque la disciplina de diseño puede ser la más abstracta de las tres, centrada inicialmente en la concepción y exploración de ideas, pero no en su implementación, es de hecho la que más herramientas tiene para prototipar esos conceptos, y la que más puede beneficiarse al hacerlo, ya que el trabajo de todo el equipo va a estar basado en las decisiones del diseño.

Muchos diseñadores con orientación técnica utilizan sistemas sencillos de programar como **Flash** o **GameMaker** para crear prototipos de sus ideas. Excel es una herramienta muy útil para cualquier diseñador, por sus facilidades para organizar ideas, tabular números y fórmulas, y visualizar resultados en forma de gráficas. En muchos casos también se pueden usar las macros programables de Excel para hacer prototipos que incorporen funcionalidades y para simular interacciones entre los elementos de juego, por ejemplo, los combates entre las unidades de un juego de estrategia que el jugador no controla directamente.

Pero un diseñador puede también **prototipar idea**s lejos del ordenador: los conceptos básicos de muchos juegos se pueden reducir a interacciones entre cartas, o fichas de un tablero, y construir los prototipos de esa manera. También es posible utilizar piezas de cartón, o incluso conjuntos de construcción, como **LEGO**, para diseñar la disposición general de los niveles de un juego, y sacar conclusiones sobre navegabilidad, facilidad de interpretarlos (o de perderse en sus laberintos) y tamaño general.

Referencias

- **https://es.linkedin.com/pulse/las-fases-del-desarrollo-de-videojuegos-el-garc%C3%ADa-ramis-7k-**
- **https://www.tokioschool.com/noticias/diseno-interfaz-videojuego/**
- **https://docs.hektorprofe.net/escueladevideojuegos/articulos/fases-del-desarrollo-de-videojuegos/**
- **https://www.superprof.es/blog/lenguaje-desarrollo-juego/**
- **https://blogs.upm.es/observatoriogate/2018/07/04/que-es-un-motor-de-videojuegos/**
- **https://www.arteneo.com/blog/storyboard-videojuegos-escuela-madrid/**