Software de entretenimiento y Videojuegos

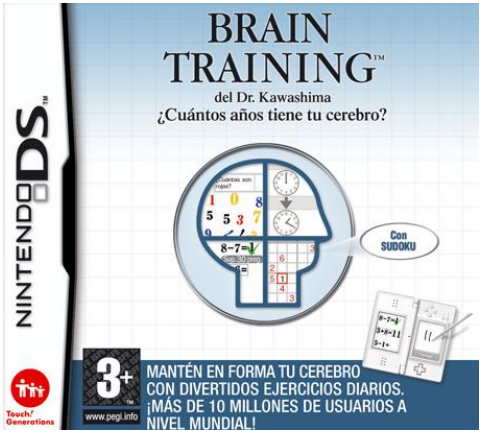
Jordán Pascual Espada
jordan@uniovi.es

Daniel Fernández Álvarez
UO212626@uniovi.es

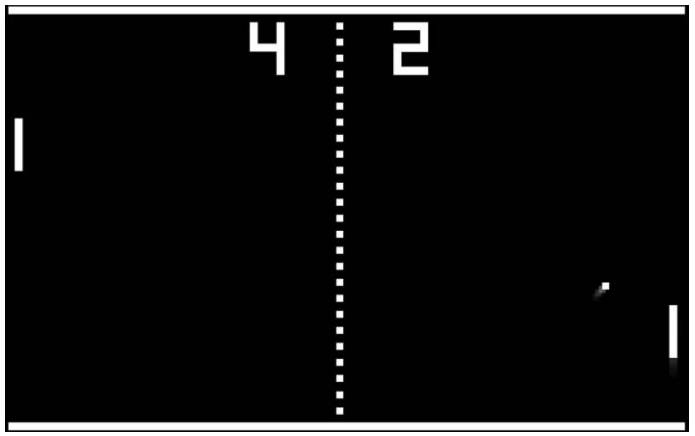
Introducción al desarrollo de videojuegos

- En el contenido desarrollado en las prácticas 1 y 2 se introducen algunos elementos comunes a la mayoría de videojuegos:
  - Enemigos.
  - Animaciones.
  - Estados.
  - Físicas.
- No todos los juegos incluyen todos estos elementos.

Introducción al desarrollo de videojuegos

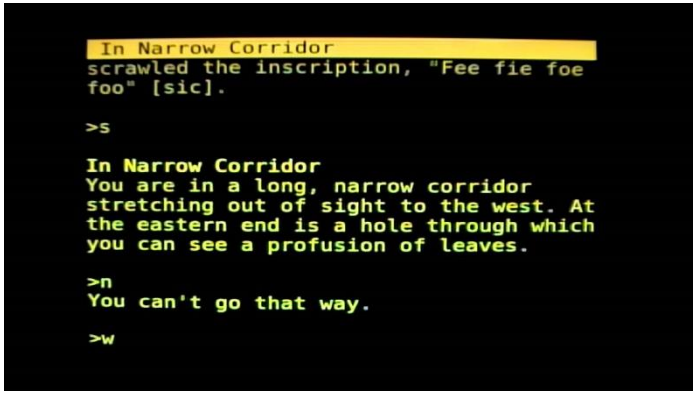


Brain training. No hay enemigos



Pong. No hay estados

Ajedrez online. No hay animaciones



Colossal Cave Adventure. No hay físicas

Enemigos

Enemigos

- En sentido estricto, otro jugador humano en un juego multijugador puede considerarse un enemigo…
- … pero con el término enemigo nos solemos referir a non-playable characters (**NPCs**).
- No todos los NPCs son enemigos.

Skyrim. NPCs en una taberna

Enemigos

- Hay diferentes tipos de enemigos (o NPCs en general) atendiendo a distintos criterios:
  - Pueden repetirse o ser únicos.
  - Estrategia de generación.
  - Su comportamiento se basa o no en decisiones del jugador.
  - Incluyen inteligencia compleja.
  - …

Enemigos

- De acuerdo a si son repetidos o únicos:
  - Mob (del inglés *mobile*):
    - Por sí solos no suelen representar un gran reto para el jugador.
    - Es común que se usen como medida para conseguir experiencia, recompensas, etc.
  - Boss (jefe):
    - Suelen tener un set de habilidades único.
    - En juegos con argumento suelen tener un carisma especial.
    - Suelen cerrar una etapa o capítulo en el juego. Presentan al jugador una oportunidad de probar sus habilidades frente a un enemigo más dificultoso de lo común.

Enemigos

KARL (smartphone).
Únicamente hay mobs

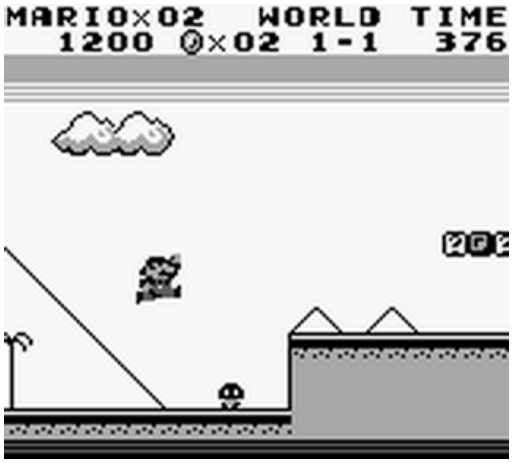
Prince of Persia: Las Arenas del Tiempo.
Mobs + bosses

Shadow of the Colossus.
Únicamente hay bosses

Enemigos

- Posicionamiento en los mapas:
  - Bosses: generalmente tienen una posición fija.
  - Mobs:
    - Cantidad y posición fijas.
    - Cantidad indeterminada y posición aleatoria.
      - Algoritmos de complejidad muy diversa para la generación automática de enemigos.

Enemigos



Super Mario Land
Posición y cantidad fijas

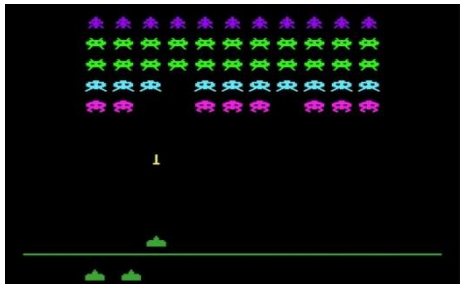
Minecraft. Posición y cantidad indeterminadas

Left 4 dead 2. Algoritmo de generación complejo.

- Un mismo nivel tiene una dificultad similar cada vez que es jugado, pero hay margen para generación aleatoria de enemigos.
- No hay final bosses, pero sí mobs más potentes. A veces aparecen en distintos lugares en distintas ejecuciones del mismo nivel.
- Intento de mantener un juego entre calma y tensión propio de una película de terror.

Enemigos

- De acuerdo al jugador:
  - No tienen en cuenta acciones del jugador.
  - Tienen en cuenta acciones del jugador:
    - Diferentes grados de inteligencia.
    - A veces se busca emular comportamiento humano o sub-humano.



Space Invaders

Pac-man

Wii Sports Resort

Enemigos

- Inteligencia:
  - Cualquier acción tomada por un enemigo de entre un set de opciones puede ser entendida como “inteligencia”.
  - Un desarrollador debe decidir qué tipo de inteligencia implementar:
    - La mejor decisión posible. Aumenta la dificultad del juego, crea una experiencia más artificial.
    - La decisión más “humana” posible. No se busca tomar la mejor decisión, si no la más realista imitando psique humana para aumentar el realismo del juego.
    - En ambos casos es necesario gestionar el nivel de dificultad: niveles no imposibles.

Enemigos

- Inteligencia artificial (IA) en los videojuegos:
  - Tema actual en distintos contextos:
    - Industria: mejor experiencia de juego, mejores ventas.
    - Academia: el videojuego es un escenario excelente para la prueba y desarrollo de técnicas de IA.

Enemigos

- Juegos asimétricos:
  - Muchos NPCs tienen motivaciones distintas (u opuestas en caso de enemigos) al jugador.
  - Cada tipo de NPC puede tener distintos algoritmos de decisión.
  - Cuanto más asimétricos sean los agentes de un juego, más complejo puede resultar desarrollar y coordinar sus algoritmos de inteligencia.

Enemigos

- Técnicas de IA empleadas:
  - Redes neuronales:
    - Inspiradas en la mente humana.
    - Hay una serie de parámetros con una serie de pesos a la hora de tomar decisiones.
    - Una red neuronal se puede entrenar con resultados para modificar la importancia de cada factor en la toma de una decisión → Pueden mejorar, aprender de errores y desarrollar comportamientos sofisticados.
    - … si se les entrena correctamente. Si no puede derivar en situaciones como la vivida por Microsoft con Tay

Enemigos

- Técnicas de IA empleadas:
  - Árboles de decisión:
    - Diagramas lógicos compuestos por estados de partida, posibilidades y resultados.
    - Unreal facilita el uso de árboles de decisión para programar NPCs.
    - Generalmente se emplean para decidir entre un set reducido de opciones de alto nivel.
    - En videojuegos complejos con muchas variables un árbol de decisión puede resultar en un problema NP-Complejo → no hay algoritmo que lo resuelva en tiempo polinómico, la complejidad crece exponencialmente con el tamaño del problema.
    - Complejidades tan altas son inviables en escenarios que requieren respuesta rápida.

Enemigos

- Árboles de decisión dentro del propio videojuego:

Final Fantasy XII. Sistema de gambits

Enemigos

- Técnicas de IA empleadas:
  - Hierarchical Concurrent State Machines HCSM:
    - Se basan en diagramas de estado
    - Cada diagrama puede estar a su vez compuesto de otros diagramas, de forma que la salida del estado de uno tiene influencia en el otro.
    - Los diagramas pueden estar o no relacionados, ser concurrentes o computarse de forma secuencial.
    - Cada HCMS produce una salida. Esta salida puede actualizarse en cada iteración del juego.

Enemigos

- En el contexto de las prácticas del bloque 1:
  - Posicionamiento en el mapa:
    - Comenzamos con posiciones fijas (coordenadas en pixels).
    - Cambiamos a generación aleatoria en el juego de naves.
    - Volvemos a posicionamiento fijo cargando enemigos desde el mapa en el juego plataformas.

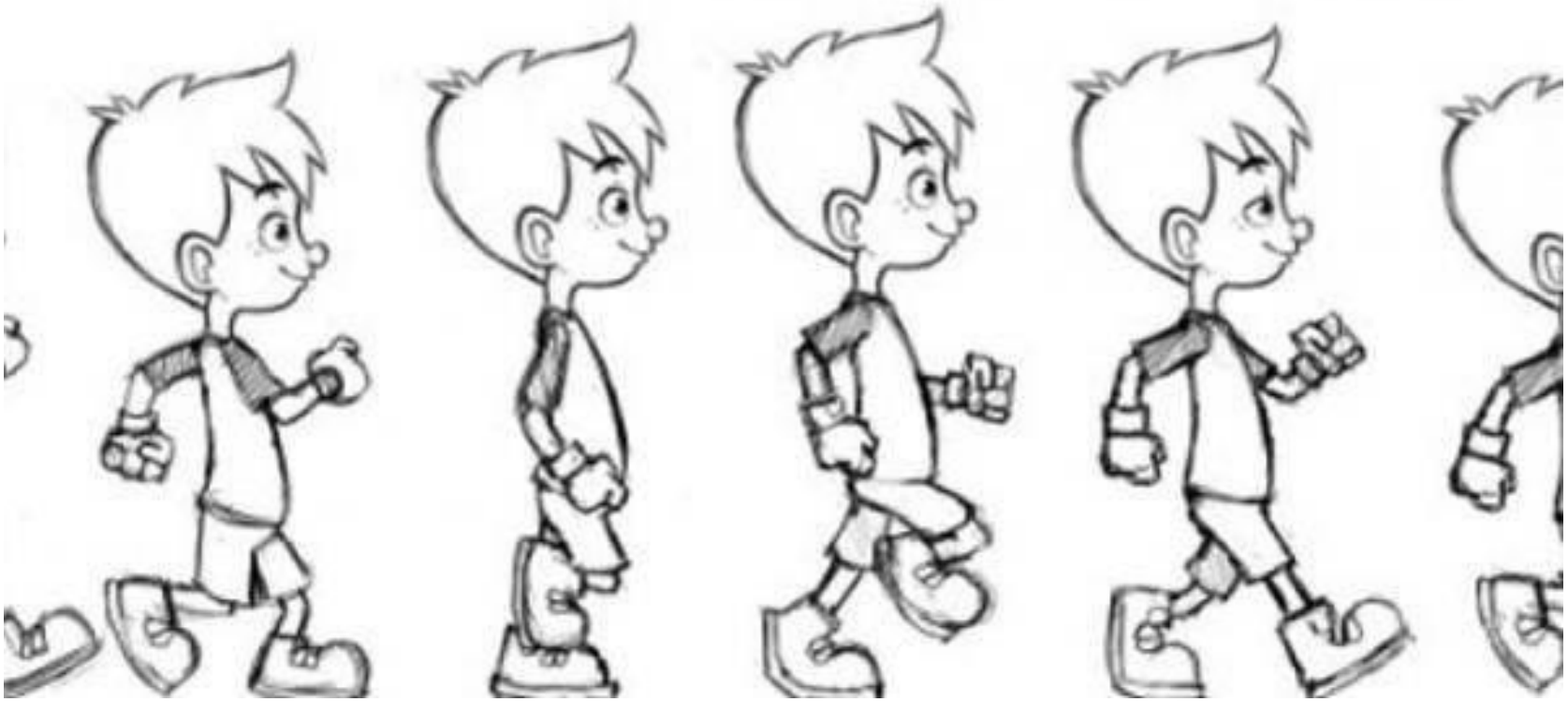
Enemigos

- En el contexto de las prácticas del bloque 1:
  - Tipo:
    - Sólo un tipo de NPC.
    - Todos los enemigos tienen las mismas características iniciales y set de habilidades.

Enemigos

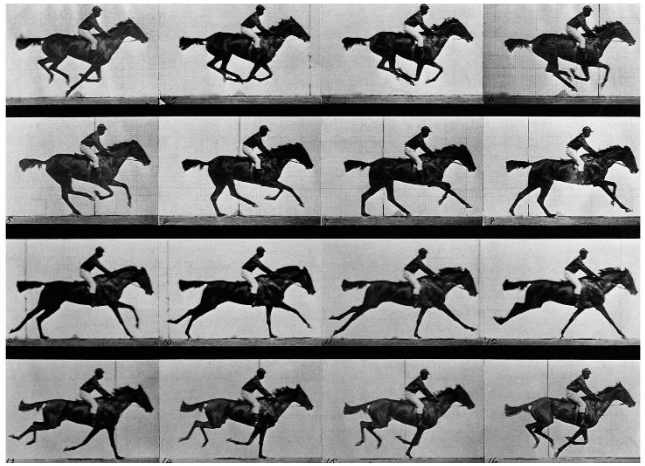
- En el contexto de las prácticas del bloque 1:
  - Inteligencia y comportamiento:
    - No afectada por las acciones del jugador.
    - Diferentes estrategias: seguir una línea, rebotes, detección de colisiones y cambio de dirección.
    - Está planteado como ampliación modificar el algoritmo de comportamiento.
    - Ejemplo: perseguir al usuario.

Animaciones



Animaciones

- Inicios:
  - Folioscopio (flip book): también conocido como cine de pulgar.
  - Libros en los que cada página representa un fotograma y el paso sucesivo de páginas representa una animación.
  - Patentado por John Barnes Linnett.
  - Usado por primera vez con fotografías por Eadweard Muybridge.



Animaciones

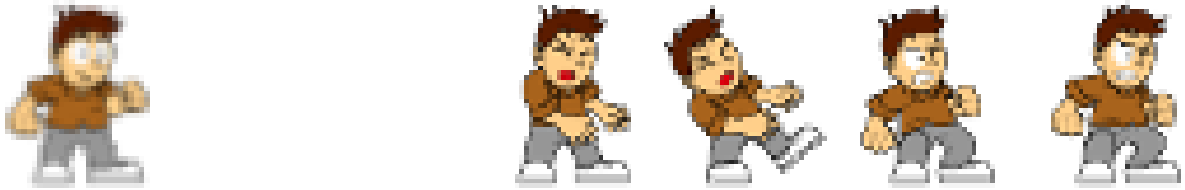
- Primeras películas
  - Thomas Edison inventa el kinetoscopio (1889-1895). Capaz de proyectar una tira de imágenes para crear la ilusión de movimiento.
  - Max Skladanowsky fue el primero en proyectar películas para el público (1895). Duraban unos 6 segundos, con música de fondo, sobre temas variados: niños bailando, un combate, un canguro…
  - Los hermanos Lumière inventan el cinematógrafo, aparato que sirve como cámara y como proyector (1985). Su primera película proyectada duraba 46 segundos.

Animaciones

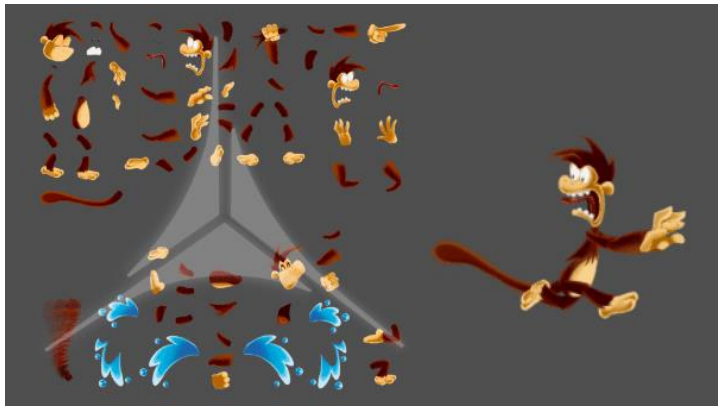
- El paradigma no ha cambiado tanto:
  - En la primeras sesiones de prácticas hay un esquema constante de ejecución. Se trata de un loop, que en cada tick (frame) :
    1. Detecta los eventos que genera el usuario (controles).
    2. Actualiza el estado lógico de cada elemento del juego.
    3. **Repinta el juego → crea la ilusión de movimiento.**
- **Este esquema siempre se repite, aunque estemos desarrollando con frameworks o engines**

Animaciones

- Animación de personajes en el primer bloque:
  - Sprites (animados o no animados).



  - Alternativa común en 2d: atlas.

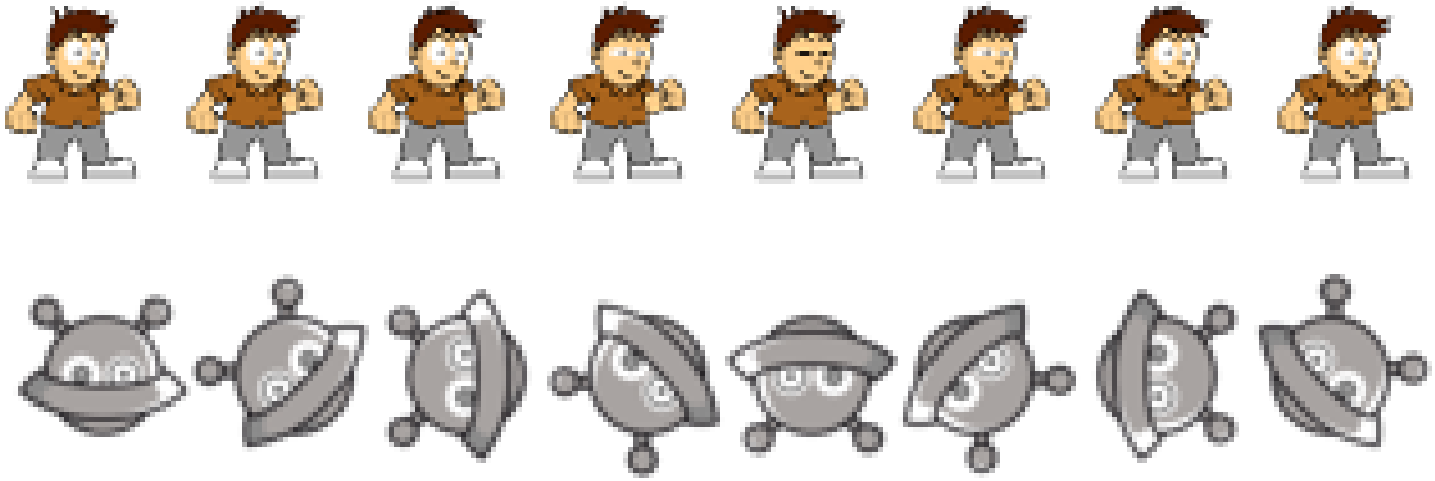


Animaciones

- Hojas de sprites:
  - Se utilizan para animar un elemento concreto (personaje, objeto…).
  - Contienen una secuencia de imágenes que crean la ilusión del movimiento de una entidad tras ser sustituidas de forma rápida (similar al flip book).
  - Se utilizan hojas con muchos sprites en lugar de un archivo por cada sprite por motivos de eficiencia: evitamos cargar un gran número de archivos.

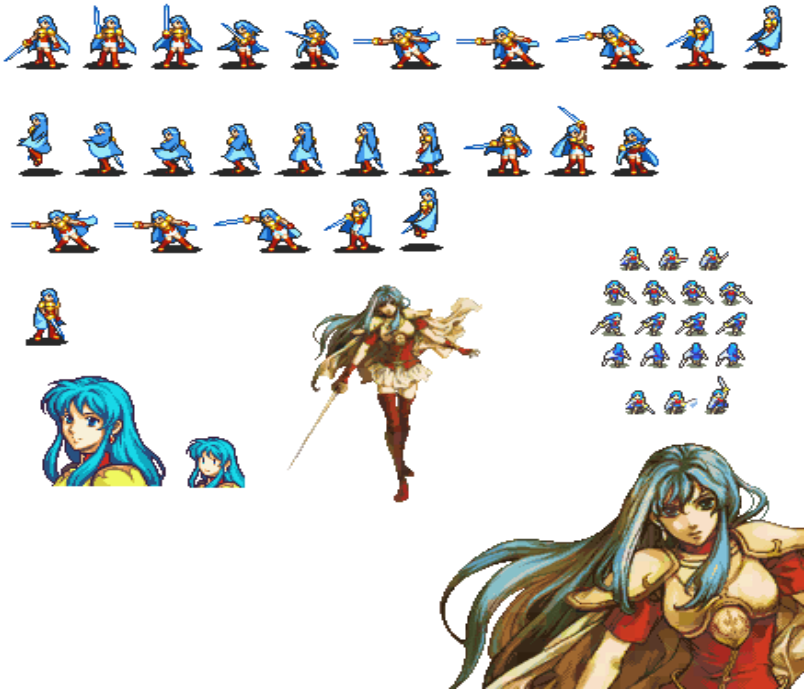
Animaciones

- En el bloque 1 de prácticas utilizamos hojas de sprites del mismo tamaño y posicionados longitudinalmente.



Animaciones

- No es la única alternativa:
  - Sprites en matriz
  - Sprites de diferente tamaño

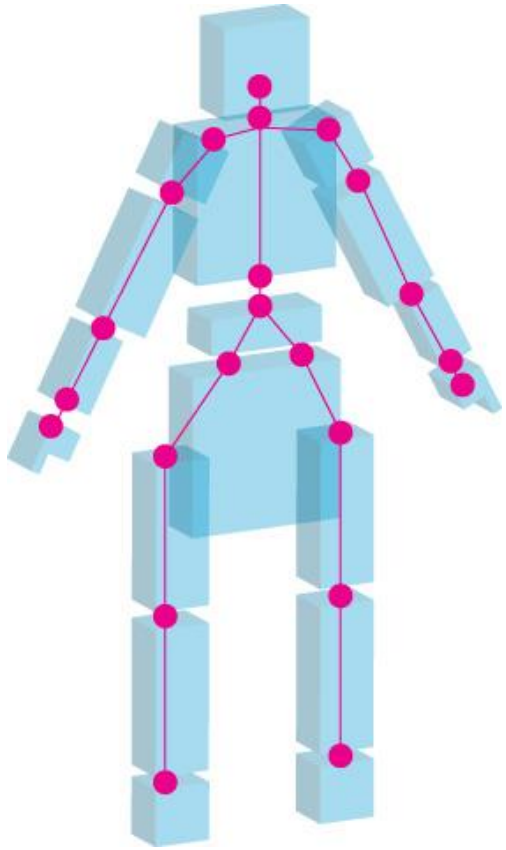


Animaciones

- Para trabajar con hojas de sprites de diferente tamaño:
  - Es necesario contar con los metadatos de cuántas imágenes hay, donde están, etc.
  - Los frameworks suelen incorporar herramientas de edición de sprites:
    - Autodetección de formas.
    - Especificación de rejillas.
    - Edición manual del tamaño de cada sprite.
    - Especificación de punto de referencia.

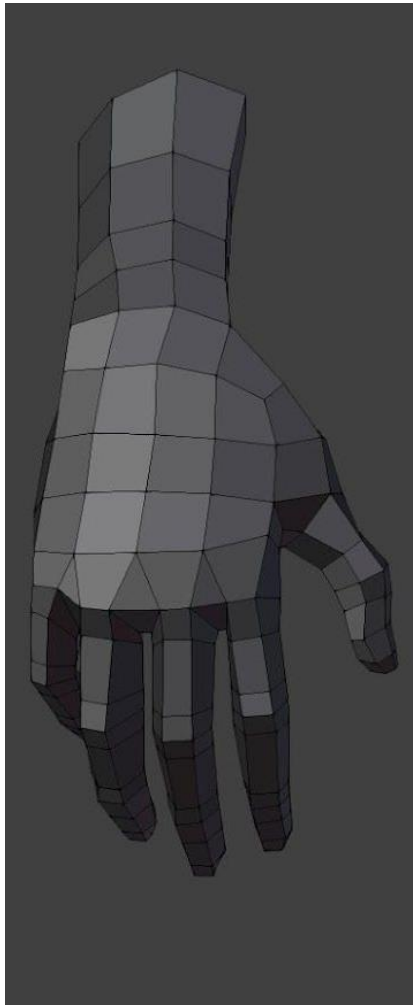
Animaciones

- Técnicas en 3D:
  - Estructuras jerárquicas de objetos:
    - Se definen puntos clave y se les asocian formas geométricas.
    - Para la animación se definen posiciones claves de los puntos.
    - Las formas son independientes. Se pueden generar huecos.



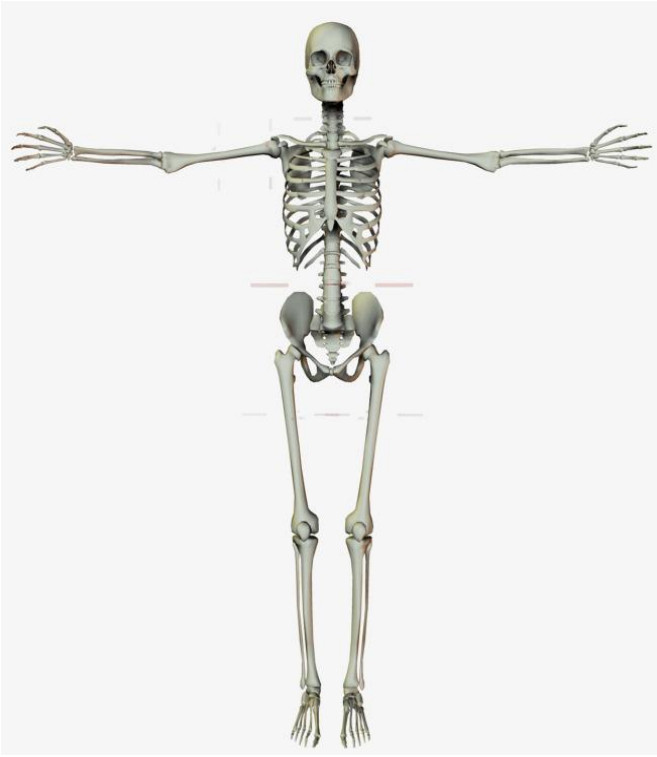
Animaciones

- Algunas técnicas 3D:
  - Mezcla entre mallas:
    - Las superficies se definen con mallas compuestas por un número finitos de vértices.
    - Más vértices → más detalle, más complejidad de cálculos.
    - Se definen posiciones iniciales y finales.
    - Los movimientos intermedios se calculan interpolando.
    - Buenas animaciones implican muchas muestras para no crear interpolaciones poco naturales.



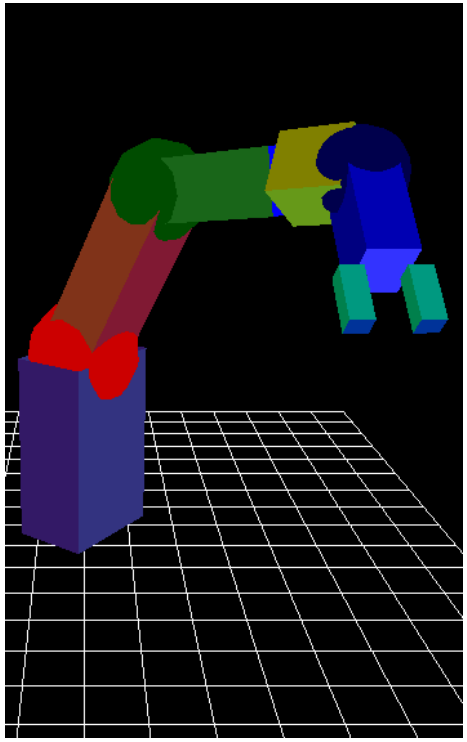
Animaciones

- Algunas técnicas 3D:
  - Endoesqueleto + mallado de piel:
    - Combina las fortalezas de las dos anteriores.
    - Se define una estructura jerárquica de objetos.
    - Forma una entidad única a través de establecer un mallado común que los envuelve a todos.



Animaciones

- Algunas técnicas 3D:
  - Cinemática inversa:
    - Consiste en calcular el movimiento de una cadena de articulaciones para conseguir que un cuerpo pase de una posición inicial a una final.
    - Las ecuaciones para resolver estos problemas pueden llegar a ser muy complejas y no tener solución única.
    - Comienza con la robótica → Escenario relativamente sencillo, robots con pocas articulaciones.



Animaciones

- Cinemática inversa:
  - Alta complejidad en humanos por el número de articulaciones → tecnologías de captura de movimiento.

Estados

Estado

- Noción fundamental en muchos juegos:
  - Pokemon a la orden de atacar:
  - Normal: ataca.
  - Dormido: puede que despierte y ataque, puede que siga dormido.
  - Confuso: quizá está tan confuso que se hiere a sí mismo.

Estados

- Los estados afectan a multitud de elementos del juego:
  - Jugador: los diferentes estados del jugador pueden generar diferentes acciones ante un mismo evento.
  - NPCs: se comportarán de distinta forma respecto al entorno en función de su estado.
  - Entorno: también los objetos inanimados pueden tener estados que afecten tanto a la estética como a la jugabilidad. Un muro podría pasar a tener un estado “derruido”, que no provocaría su desaparición total del mapa pero quizá sí afecte a las colisiones.

Estados

- Desde el punto de vista del diseño:
  - Necesario definir los posibles estados de cada elemento del juego y los comportamientos asociados a cada estado.
- Desde el punto de vista del desarrollo:
  - ¿Cómo crear un código mantenible para gestionar la lógica de estados?

Patrón de diseño State

Estados

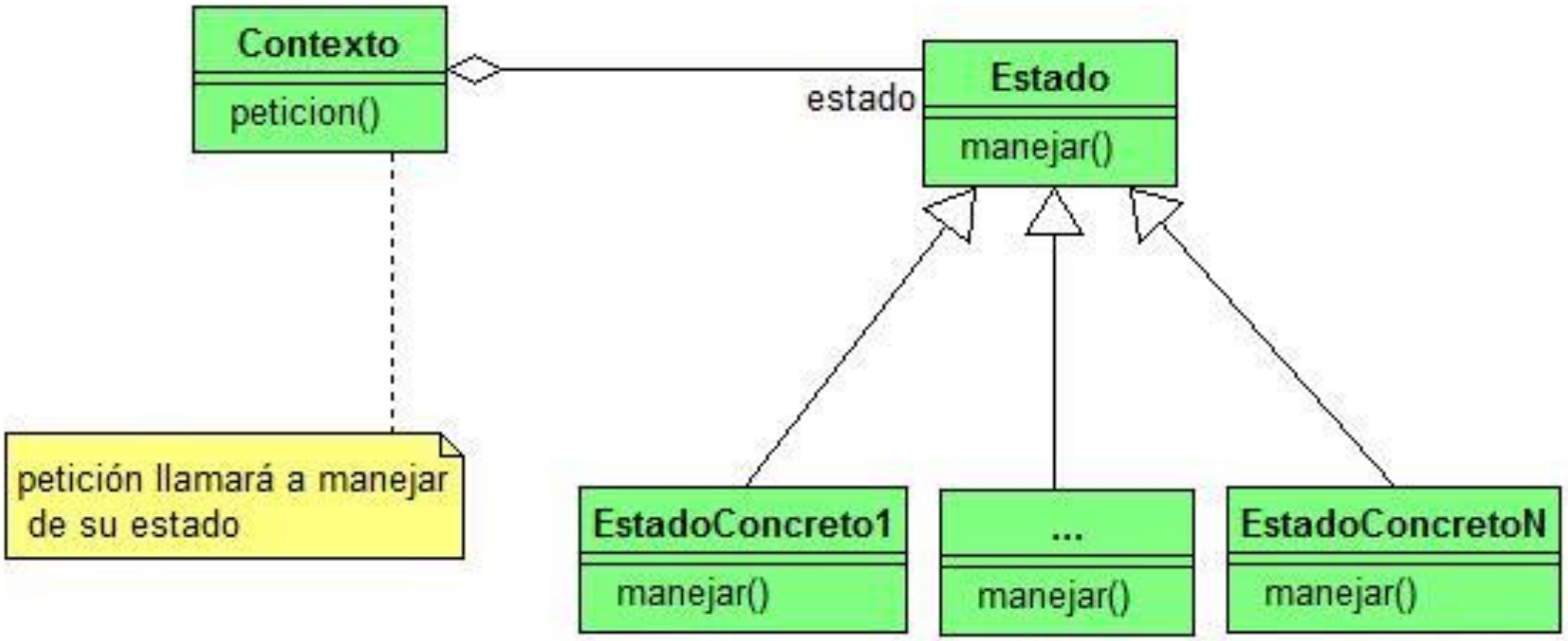
- Patrones de diseño:
  - Por muy específico que te parezca el problema al que te estás enfrentando, es muy probable que desde un punto de vista estructural alguien antes que tú ya se halla enfrentado (y halla resuelto) un problema muy similar.
  - En esencia, un patrón de diseño es una forma reutilizable de resolver un problema.
  - Utilidad:
    - Ahorran tiempo de diseño.
    - “Aseguran” un código mantenible.
    - Establecen un lenguaje común.

Estados

- Patrones de diseño:
  - Todo patrón se compone de unos elementos fundamentales:
    - Nombre: imprescindible para establecer un lenguaje común.
    - Tipo: creacionales, estructurales o de comportamiento.
    - Propósito y descripción.
- Patrón State:
  - Nombre alternativo: Objects for State.
  - Tipo: patrón de comportamiento.
  - Propósito: permitir que un objeto cambie su comportamiento en tiempo de ejecución en función de su estado interno.

Estados

- Estructura del patrón State



Estados

- Escenario de ejemplo

```
public class Persona {
    Edad estado;

    void preguntarQueTal(){
        estado.preguntarQueTal();
    }
    void crecer(){
        estado.crecer(this);
    }
    void setEstado(Edad estado){
        this.estado = estado;
    }
}
```

```
public abstract class Edad {
    abstract void preguntarQueTal();
    abstract void crecer(Persona persona);
}
```

Estados

```
public class Adolescente extends Edad{

	void preguntarQueTal() {
		gruñir();
		irALaHabitacion();
	}
	void crecer(Persona persona){
		persona.setEstado(new Veterano());
	}
}
```

```
public class Veterano extends Edad{

	void preguntarQueTal() {
		hablarDeLaMili();
	}
	void crecer(Persona persona) {
		persona.setEstado(new Bebe());
		//Aplicación reencarnativa
	}
}
```

```
public class Bebe extends Edad{
	void preguntarQueTal() {
		; //No hacer nada
	}
	void crecer(Persona persona){
		persona.setEstado(new Adolescente());
	}
}
```

Estados

- Ventajas del patrón State:
  - Se consigue encapsular en una clase la funcionalidad completa de un estado.
  - Los estados pueden cambiarse en tiempo de ejecución.
  - Resulta sencillo incorporar nuevos estados.
  - Transiciones entre estados más explicitas.
  - Posibilidad de compartir estados si sólo definen un comportamiento.

Estados

- Desventajas del patrón state:
  - Si el comportamiento de un objeto depende a la vez de varios estados el código puede ser problemático de mantener.
  - Escenarios que requieren muchos estados pueden desarrollar estados incongruentes o incompatibles entre elementos.

Estados

- Cuestiones de implementación a tener en cuenta:
  - Gestionar la creación y destrucción de objetos de estado → peligro de caer en una explosión de objetos. ¿Es posible aplicar el patrón Singleton o clases estáticas?
  - El patrón no define qué agente dentro del código es el encargado de hacer la transición entre estados. Alternativas:
    - ¿Los propios objetos?
    - ¿Delegar en tablas de transición de estados?

Estados

- Contexto de prácticas:
  - En el código proporcionado para el bloque I no se aplica el patrón State.
  - En vuestras propias practicas, ¿merece la pena aplicarlo?
    - Si la lógica de estados tiene una cierta complejidad, es probable que sí.
    - Cómo detectar si mi código tiene esa complejidad en referencia al estado: ¿tus métodos tienen con frecuencia una serie de bloques “if” anidados para decidir una respuesta en función de varios parámetros → ¡¡Aplica State antes de que sea tarde!!

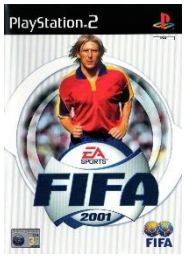
Físicas

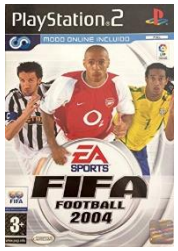
Físicas

- El manejo de físicas abarca todo aquello relacionado con el movimiento de objetos físicos y su interacción con otros objetos o condiciones del entorno:
  - Gravedad
  - Colisiones
  - Suma de fuerzas
  - Fricción
  - Rebotes
  - …

Físicas

- Diferentes formas de aplicar físicas:





Cada nueva edición del videojuego **FIFA** promete unas físicas del balón mejoradas respecto a la anterior.

GeometryDash es un Speed Run en 2D con un único control. Su dificultad reside en un cambio constante de las físicas: inversión de gravedad, cambio entre plataformas y aéreo…

Físicas

- Existen también frameworks o librerías para físicas de videojuegos en JavaScript:
  - Playground.js
  - Phaser
  - Panda.js
  - Quintus
- No vamos a usar ninguna en el bloque 1, no es el objetivo → **mantengámoslo simple**.

Físicas

- Cocos2D: motores para entornos en 2D como:
  - Box2D.
  - Chipmunk.
- Unreal:
  - PhysX.

UNREAL
ENGINE

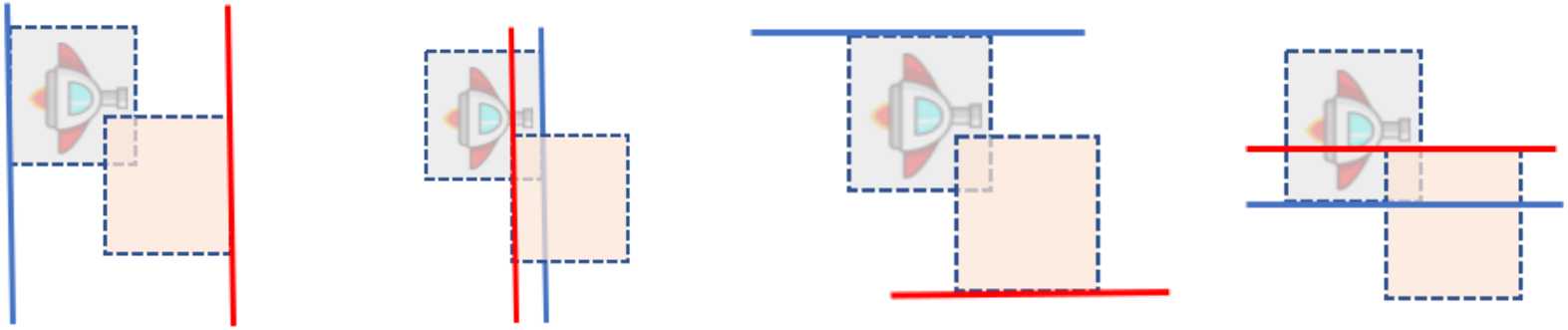
PhysX™
by NVIDIA

Físicas

- En el contexto de nuestra práctica en el bloque I, dos tipos de físicas :
  - Detección de colisiones.
  - Fuerzas en el entorno:
    - Ausencia de fuerzas → libertad de movimiento total.
    - Gravedad → aceleración continua hacia abajo.

Físicas

- Detección de colisiones:
  - Estrategia escogida en prácticas:
    - Cada elemento es un rectángulo.
    - Se produce una colisión cada vez que dos rectángulos tienen algún área solapada.
    - Comprobamos esta situación comparando la posición de los lados de los rectángulos.

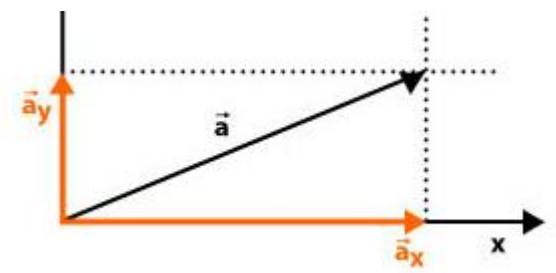


Físicas

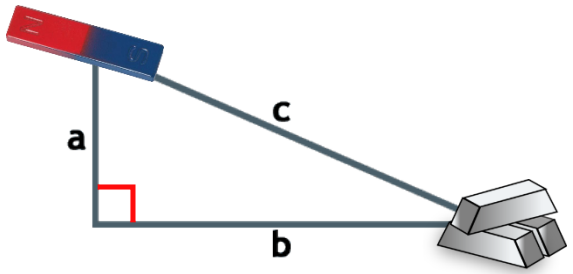
- Gestión de fuerzas del entorno:
  - Ausencia de fuerzas: algoritmo simple, solo es necesario procesar comandos del usuario y colisiones.
  - Gravedad: desde el punto de vista físico, la gravedad consiste en una aceleración continua.
    - Implementación simple a través de un aumento continuo de la velocidad y en ausencia de colisión en cada tick.
    - Fuerza de la gravedad configurable modificando un único parámetro.
    - Estrategia compatible con otras fuerzas (gravedad invertida, magnetismo…)

Físicas

- Aceleraciones en direcciones que no son verticales u horizontales:
  - Descomposición vectorial.



- Fuerzas en función a la distancia a ciertos objetos (ej: magnetismo):
  - Teorema de Pitágoras



$a^2 + b^2 = c^2$

Mapas de tiles

- Mapas de tiles:
  - El mapa dividido en cuadrados (casi siempre) del mismo tamaño.
  - Cada elemento del mapa se posiciona en un tile (o un número entero de tiles).
  - Cada elemento móvil se puede desplazar de un tile a otro, pero no a posiciones intermedias.
  - Se usan casi siempre en entornos 2D (a veces con perspectiva 2.5D).
  - Simplifica mucho los cálculos:
    - Las colisiones consisten únicamente en comprobar si dos elementos están en el mismo tile de la matriz.
    - Fácil manejo de las coordenadas x e y, que representan un tile de la matriz → nos olvidamos de hacer cálculos con el ancho/alto de las imágenes respecto a un punto de referencia.

Mapas de tiles

The Legend of Zelda: A Link to the Past

Pokemon Yellow

Final Fantasy VI

The Battle for Wesnoth

Mapas de tiles

- Perspectiva 2.5D:
  - Consiste en crear cierta sensación de 3D en entornos 2D.
  - Se consigue priorizando el orden en que se pintan los objetos, dando lugar a volumen y profundidad.

Mapas de tiles

- Perspectiva 2.5D en mapas de tiles:
  - Los objetos pueden ocupar físicamente menos tiles de los que ocupan estéticamente.
  - Sólo los tiles físicos se utilizan para el cálculo de colisiones.
  - Puede ocurrir que tiles estéticos de diferentes objetos ocupen la misma posición en la matriz.
  - La sensación de profundidad se crea pintando antes los que tenga su tile inferior más arriba en la pantalla.

Mapas de tiles

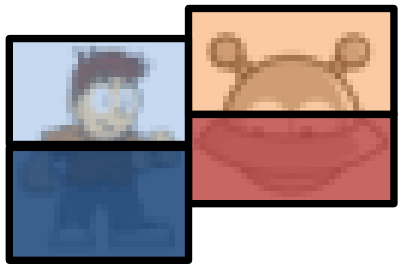
- Perspectiva 2.5D en mapas de tiles:

Mapas de tiles

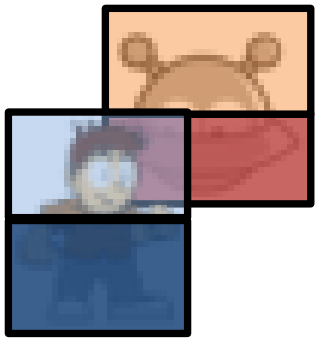
- Perspectiva 2.5D en mapas de tiles:

Mapas de tiles

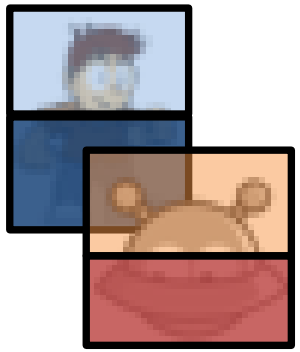
- ¿Perspectiva 2.5D fuera de mapas de tiles?
  - Estrategia muy similar, implementación más compleja.
  - Cada objeto debería almacenar cuál es su parte física para el cálculo de colisiones y cuál su parte estética.
  - Los objetos han de pintarse empezando por aquellos cuya coordenada inferior está más arriba en la pantalla.



Colisión



Jugador superpuesto



Enemigo superpuesto