UNIVERSIDAD DE COSTA RICA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS DE LA COMPUTACIÓN E INFORMÁTICA

CI-2807- Creación de Videojuegos
Prof: Dr. Markus Eger

Resumen

Elaborado por:

Melissa Garro Morales B02604

11 de julio del 2019

Cadenas de Markov

Es un proceso en el que una variable aleatoria X_n va cambiando de estado (según las probabilidades de transición). En un videojuego los estados pueden representar:

- Tipos de enemigos.
- Tipos de acciones del NPC(agresivo,...,pasivo).
- Tipos de ataque del enemigo.
- _____.

Retos	
¿De dónde se obtienen las probabilidades de transición?	Las probabilidades de transición pueden ser asignadas manualmente para tener control sobre la salida.
¿Cómo se asegura de que la salida sea razonable?	Poner restricciones, probar que el nivel se pueda pasar, probar múltiples variantes.

¿Qué es un juego?

Hay muchas definiciones distintas de lo que un juego es, pero todos los los expertos coinciden en dos elementos: **Reglas y Meta(s)**.

Por otra parte, hay "juegos" que realmente **no** son juegos pero la forma en la que le damos uso los hace juegos, por ejemplo:

> Piezas de LEGO: al proponernos construir algo lo convertimos en un juego creando nuestras propias reglas y metas:
>
> **Meta:** Construir una casa.
>
> **Regla:** Utilizar legos rojos para las paredes.

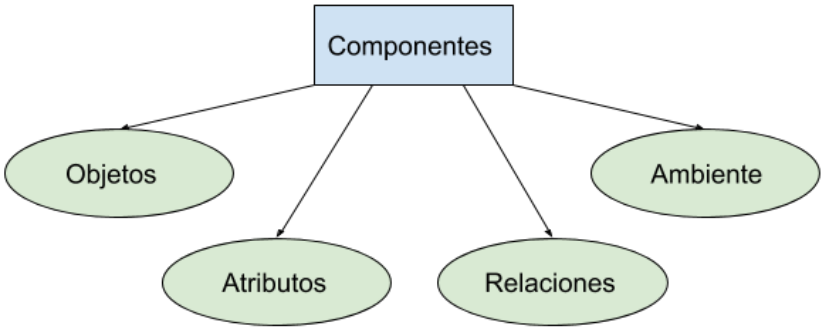
Géneros de juegos: Indican la mecánica del juego. Estamos más enfocados en la mecánica del juego por lo que es importante saber las características que los hacen diferentes. A continuación se mencionan algunos ejemplos de géneros:

Actividad: Cite características que los identifican

Género	Características
Shooter primera persona	

Shooter tercera persona	
Juego de aventura	
Juego de acción	
Plataforma	
Juego de estrategia en tiempo real	
Juego de rol	
Juego de simulación	
Juego de deportes	

Diseño de videojuegos: el juego es un sistema definido por un conjuntos de 4 componentes(el diseño de videojuegos define las interacciones entre estos):



Framing

El sistema de un juego puede integrar marcos(framing) que permiten que un conjunto de reglas, se experimenten como juego, dentro de un contexto cultural.

- ❏ Formal
  Se percibe el juego como un conjunto de reglas.
  Cerrado del entorno exterior.

- ❏ Experimental
  El juego es una experiencia para los jugadores.
  No prioriza la mecánica del juego, sino el efecto que tienen en el jugador.

- ❏ Cultural
  Impacto del juego en la sociedad/cultura.

Como ejemplo de lo anterior, podríamos mencionar que el juego de ajedrez está incrustado en un marco formal (por las reglas) y este sistema está integrado, a su vez, en un marco cultural, ya que las piezas blancas siempre inician la partida.

Reglas: Mecánica de juego

La mecánica del juego describe en detalle las interacciones entre los objetos, definiendo las reglas por las cuales los objetos interactúan y cambian sus atributos:

- Controles(derecha, izquierda, arriba, abajo,...) genera una acción específica.
- El comportamiento amistoso en el caso de aliados.
- El comportamiento agresivo en los enemigos.
- Comportamiento del nivel (Level behavior), cómo el nivel influencia el juego.
- Especifica las condiciones de ganar / perder.

Metas (Goals)

La meta de nuestro videojuego es lo que deseamos que el jugador haga:

-Rescatar a la princesa.
-Salvar al mundo.
-Alcanza el final del nivel.
-Terminar una misiones.
-_____.
-_____.

¿Cúal es el nombre que se le da a un juego sin meta particular?
R/_____.

❖ Diseño basado en Niveles (secciones, capítulos)

La progresión lineal hace que resulte más fácil de definir y se controlando mejor la experiencia del jugador. La meta sería completar el nivel.

❖ Diseño de juegos de combate

Se caracterizan por ser competitivos, además, la progresión exterior añade un meta-juego. Con frecuencia la meta está asociada a estrategias de monetización, con las que el jugador puede desbloquear: personajes, armas, técnicas, entre otros.

❖ Diseño de ventajas incrementales / puntuación

Eliminar monstruos hasta mientras se va incrementando la dificultad. Permite la meta de obtener armas mejoradas, que hacen que el juego sea sencillo de nuevo.

❖ Sandbox

Generalmente hay un conflicto central, se crea un mundo con muchas opciones para que los jugadores elijan qué hacer. Meta no está definida.

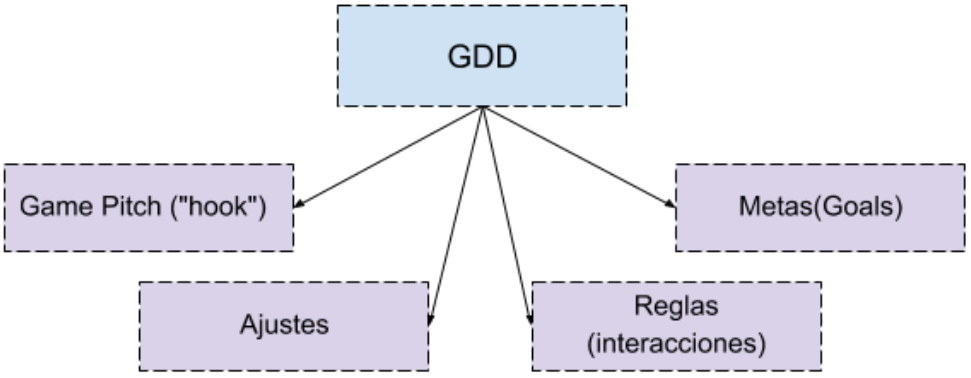
Logros(Achievements)

Además de las metas(goals), los juegos brindan a los jugadores "logros". A menudo se utilizan los logros como "bonificaciones" u objetivos opcionales. Algunos tipos:

- Inevitable. _____(Ejemplo)
- Opcional. _____(Ejemplo)
- Inspirador. _____(Ejemplo)

1.Game Design Document (GDD)

Documento que cambia constantemente, se han descrito como “Obsoletos”, GDD varía según la compañía. El GDD está compuesto de 4 partes:



1.1. Game Pitch:

Se pretende enganchar al lector, es la parte más importante del documento, se puede utilizar para dar un giro a la historia de un juego existente, mencionar la ambientación, la mecánica o alguna idea.

1.2. Ajustes(Setting):

Puede incluir los **Personajes, Mundo, Historia**. (no es aplicable a todos los juegos).

Personajes ¿Quién es el jugador?	Mundo ¿Dónde está el jugador?	Historia ¿Qué necesita hacer el jugador?
Personalidad	Tiempo / nivel tecnológico	Los personajes quieren alcanzar sus metas.
Metas	Ubicación, real o ficticia.	Las metas a alcanzar resultan en obstáculos.
Relaciones	Facciones / grupos relevantes	¿Cómo avanza el jugador a través de la historia?
Conflictos		Historias no lineales, juegos sandbox.

1.3. Reglas(interacciones)/Mecánica de juego: Se compara con el diseño en Ingeniería de software. En él se describe el juego formalmente y proporciona una base para la implementación. Responde a las preguntas: ¿Qué hace el jugador en el juego?. Las reglas describen en detalle las **relaciones**, **objetos** y **atributos** (En el GDD incluso se pueden definir en una tabla)**.**

1.4. Metas(Goals): ¿Qué debe lograr el jugador? Existen las condiciones para ganar, perder, cuáles son las misiones, ¿Por qué se debería jugar el videojuego?

Estética: Arte del videojuego, debe responder a preguntas tales cómo: ¿Fue hecha por los creadores del videojuego? sino,¿De dónde se obtuvo el arte del juego?, ¿Cómo debe verse/sentirse el juego? Si el juego va a tener sonido especificar si es propio o indicar su respectiva referencia. Arte Conceptual, prototipos de pantallas, etc.

Game Design Document (GDD) a un juego

Este documento sirve como una guía que nos dice como comenzar a programar e implementar lo que definimos en él. Los objetos se dividen en tres categorías:

- ➔ Objetos con comportamiento (código)
- ➔ Objetos pasivos de la escena
- ➔ Objetos abstractos (botones)

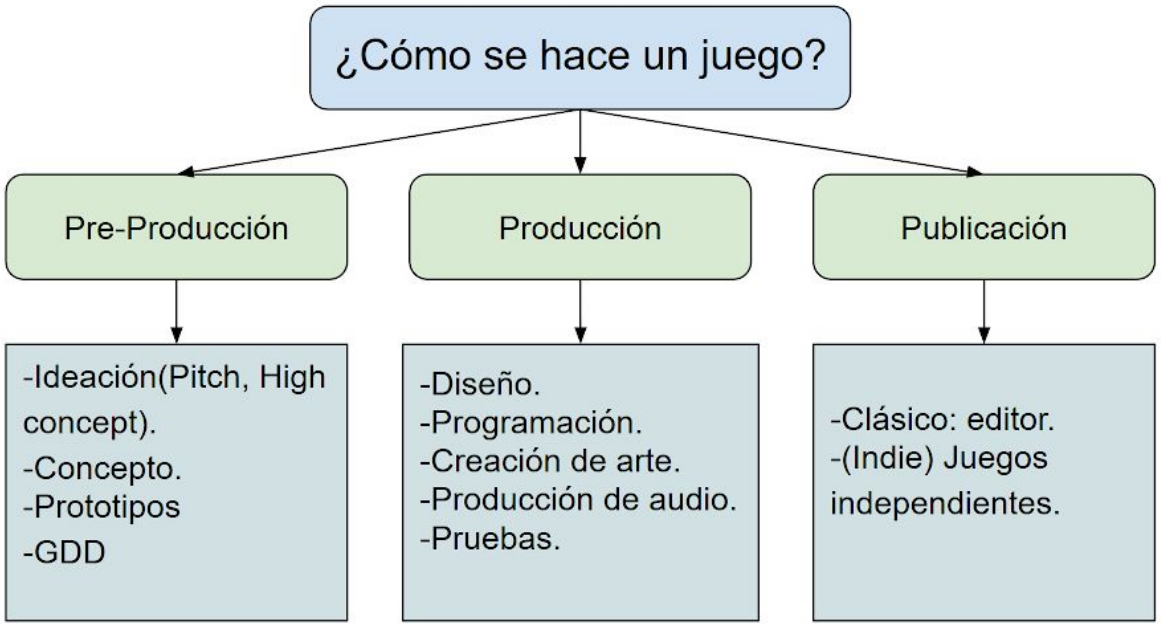
2. Proceso de desarrollo

2.1. Enfoque Clásico: Utilizado en el pasado, en este caso el *Game Design Document* (GDD) fue utilizado para presentar el proyecto a editores con el objetivo de obtener financiamiento.

Game Development Studio: Responsable de la idea e implementación del juego.

Game Publisher: Responsable de financiar el desarrollo del juego, comercializarlo y distribuirlo.

2.2. Enfoque de desarrollo independiente (Indie): Con la llegada de Internet el desarrollo de videojuegos se hizo más accesible, existen más enfoques distintos para el desarrollo de videojuegos y variedad de herramientas como Unity, Unreal Engine, entre otras.



Fase 1: Preproducción

En esta etapa se espera responder:

- ¿Qué hace que el juego sea diferente?
- ¿Es una idea viable?
- ¿Cómo pasas de la idea a un juego real?
- *Game Design Document*

Fase 2: Producción

Se divide en tres partes:

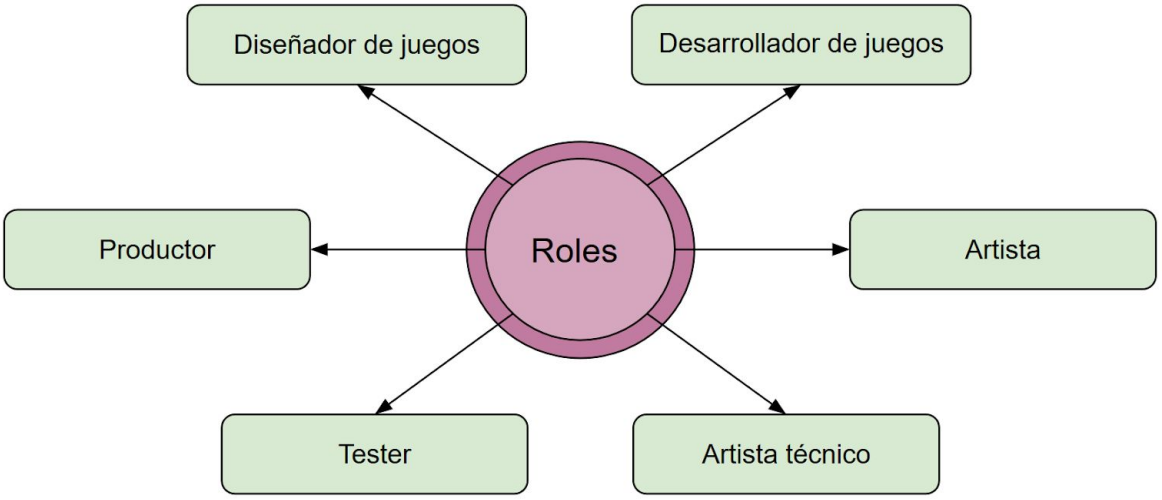
Diseño	Programación	Arte
Del mundo en el que está orientado el juego. Del sistema (mecánica, reglas, interacciones). De contenido (niveles, misiones). Del audio del juego. De la interfaz de usuario (menú).	Gráficos, efectos. Implementación de la mecánica. AI (*Artificial Intelligence*) Física. Interrelaciones. Herramientas para diseñadores.	Activos gráficos como: 2D 3D Interfaz de usuario Audio: Música del juego Efectos de sonido Efectos especiales: Efectos de partículas VR Virtual Reality

Fase 3: Publicación

Existen dos enfoques:

Tradicional	Independiente(Indie)
1) Desarrollador construye el juego. 2) El editor vende el juego. 3) El editor a menudo financia al desarrollador durante el desarrollo.	1) Plataformas móviles (Google Play, AppStore). 2) Steam, Epic Games Store. 3) Auto-publicación (sitio web).

3. Roles en el desarrollo de juegos



3.1. Diseñador de Juegos

- ❏ Visionario principal (puede consistir en un equipo).
- ❏ Desarrolla(n) la idea, la mecánica, los objetivos(*goals*) del juego.
- ❏ Esto no significa que nadie más tenga influencia en la mecánica.

3.2. Desarrollador de Juegos

- ❏ Escribe el código.
- ❏ Especializaciones:
  - ➔ Motor / Gráficos / Audio.
  - ➔ Jugabilidad.
  - ➔ Inteligencia Artificial.
  - ➔ Interfaz gráfica.
  - ➔ Herramientas.

3.3. Artista

- ❏ Los artistas crean los activos para el juego.
  - ➔ Modelos 2D / 3D
  - ➔ Elementos de la interfaz de usuario.
  - ➔ Sonido.

- ❏ La mayoría de los artistas no son programadores y viceversa.

3.4. Artista Técnico

- ❏ Los artistas técnicos están capacitados en arte y programación, y sirven como conexión entre los dos "mundos".
- ❏ También puede ser responsable de efectos tales como animaciones, líquidos, efectos de partículas.

3.5. Tester

Al igual que las pruebas de software regulares, las pruebas de juegos revelan errores de software

Algunos tipos de "errores"(*bugs*)

- Problemas de balance.
- Problemas de jugabilidad.
- Inconsistencias narrativas.
- Exploits.

3.6. Productor

- ❏ Project Manager.
- ❏ Responsable de mantener el proyecto en marcha.
- ❏ De las ofertas de contratación, presupuesto e infraestructura.
- ❏ Elimina obstáculos para el equipo.
- ❏ Gestiona las expectativas de los interesados.

Game Engines(Motor de juegos)

- Los juegos normalmente se dividen entre un *engine* y contenido
- La *engine* generalmente puede soportar múltiples juegos diferentes (teóricamente)
- Las extensiones del juego no requieren cambios la *engine* (idealmente)
- El motor proporciona componentes para cargar contenido, renderización, física, inteligencia artificial, sonido, etc.
- ¿Por qué?
  - ➔ Rendering, física, etc. son componentes reutilizables.
  - ➔ El juego se vuelve más pequeño y sencillo de cambiar.

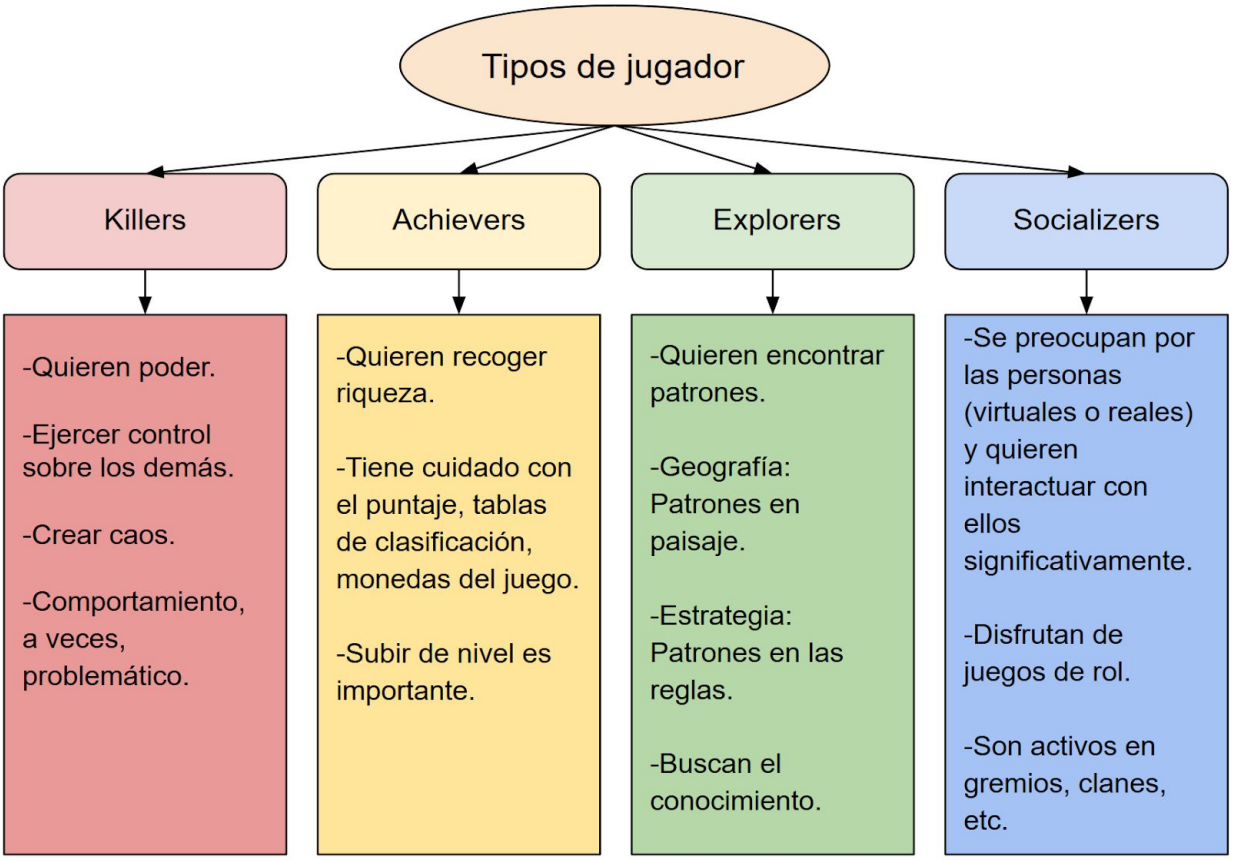
➔ Se hace más fácil hacer más juegos.
➔ La buena práctica de ingeniería de software dice que deben escribirse por separado.
➔ *Game Engine* puede ser licenciada.

Usando Game Engine:

1) La programación es muy diferente de la programación regular, cuando te acostumbras, esta forma de codificación es muy buena.
2) En la mayoría de los programas tienes un hilo principal de ejecución.
3) Ahora tienes docenas de objetos, cada uno con su propia rutina, considere cada objeto del juego por separado e implemente su comportamiento individualmente.
4) Las colisiones físicas hacen que los eventos ocurran, se debe pensar en términos de "si x entonces y" (como en tu GDD), para cada objeto del juego.
5) Si necesita un estado global, cree un objeto GameManager o similar que lo almacene.

Ejemplos de Engines:

2D: PyGame, GameMaker, RPGMaker, etc.
3D: Ogre3D, CryEngine, Lumberyard, Unreal, etc.

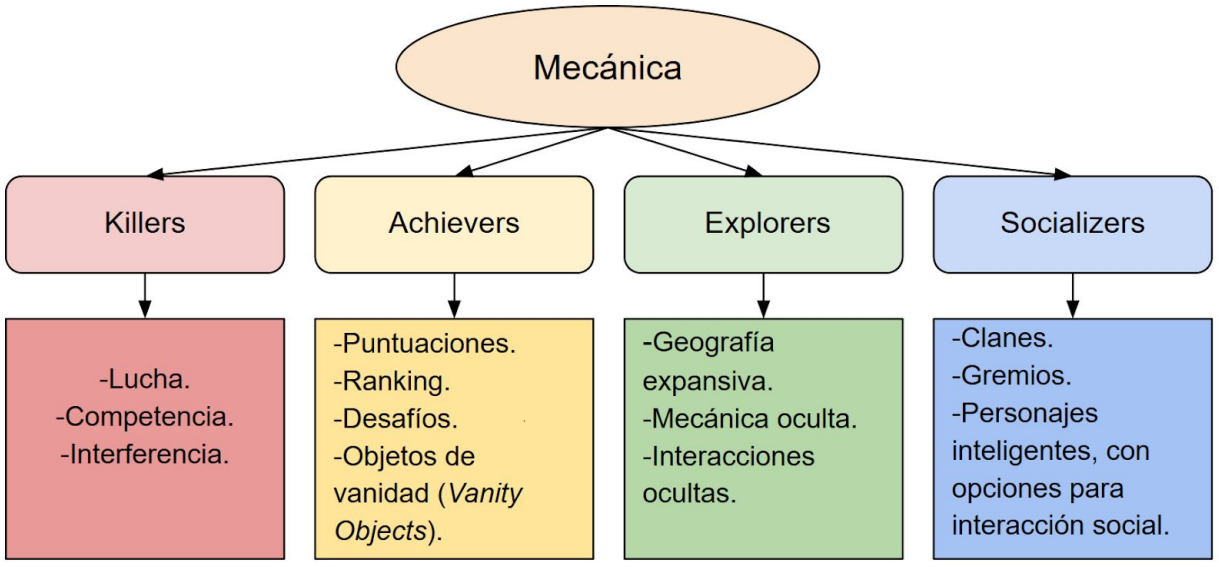


¿Qué tipo de jugador eres?_____.

¿Por qué es importante?

En definitiva, los juegos son para jugadores y diferentes jugadores quieren cosas distintas, entender esto nos permite ofrecer contenido a tipos específicos.

Entender lo que quieren los jugadores nos ayuda a diseñar mejores juegos para ellos.



Flujo(Flow)

En el "flujo" de juego, las personas se centran exclusivamente en la tarea:

Si es demasiado difícil: la gente se pone ansiosa.

Si es demasiado fácil: la gente se aburre.

Lo que es considerado fácil o difícil cambia con el tiempo conforme el jugador va desarrollando habilidades para completar la tarea.

Para mantenerse en el "flujo", los jugadores necesitan:

- ❏ Objetivos(*goals*) concretos.
- ❏ Reglas constantes.
- ❏ Acciones que les permitan alcanzar los objetivos(*goals*).
- ❏ Retroalimentación clara y oportuna.
- ❏ Sin distracciones.

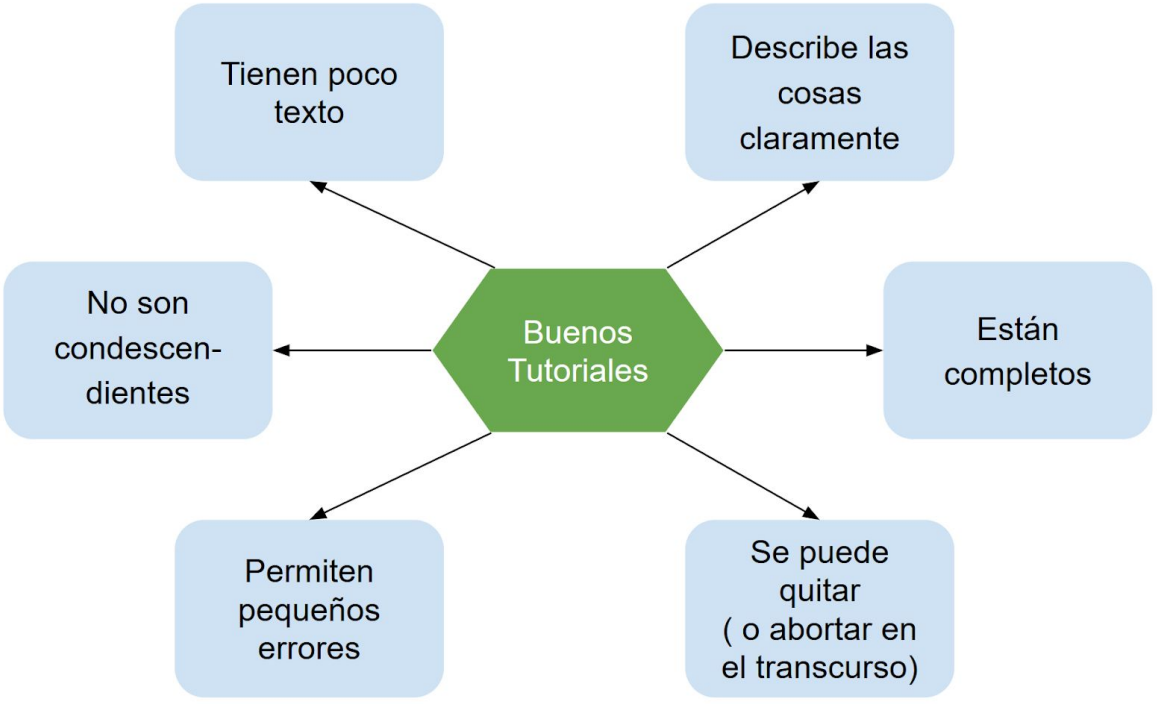
En los videojuegos esto significa:

- ❏ Proporcionar al jugador algo que hacer(tareas, objetivos).
- ❏ No cambiar las reglas de repente.
- ❏ Aumenta la dificultad de forma gradual, a medida que los jugadores mejoran(desarrollan habilidades).
- ❏ Dígale al jugador cómo lo están haciendo en sus metas para que sepa lo que debe mejorar.
- ❏ No abruman al jugador con información.

¿Cómo los juegos te mantienen en el flujo?

- ❖ Te da una mejor arma cuando matar monstruos es más difícil.
- ❖ Introduce nuevas mecánicas gradualmente, luego mezcla y combina.
- ❖ Dividen las misiones en subtareas más pequeñas de complejidad creciente.
- ❖ Retroalimentación del progreso.

Tutoriales



4. Narrativa

¿Qué pasa con el jugador?

- La narrativa lo puede guiar a lo largo de los niveles.
    - Tiene que resolver los obstáculos.
- El jugador podría tomar decisiones que influyan en la narrativa.
    - Historias ramificadas.
    - Explosión combinatoria.
- No hay historia, la narrativa surge de las interacciones.

La narrativa consta de dos partes:

1) Historia → Lo que está pasando
2) Discurso → ¿Cómo se narra?

4.1. Historia: La historia no tiene por qué ser explícitamente definida en el juego, puede ser implícita.

¿Qué impulsa la historia del juego?	
Metas	Obstáculos
• Metas(*goals*) del personaje. • Metas(*goals*) del autor *(Authorial goals).*	• Algo no funciona como se planeó en la historia. • Alguien más está tratando de interferir debido a un objetivo(*goals*) conflictivo.

4.2. Discurso:

¿Cómo contamos la historia?			
Cortes de Escena	Diálogo	Artefactos de ese mundo	Interacciones

Mundos

Cuando se crean mundos en los videojuegos estos deberían ser:

- Intuitivos
- Que permitan contar muchas historias.
- Historias fantásticas o fuera de lo común.
- Permite descubrir nuevas cosas.
- Son diferentes del mundo real.

AI del Juego

La primera regla del AI del juego: *No importa cuán inteligente o tonta sea tu IA, siempre y cuando juegue bien.*

Pathfinding

En muchos juegos tienes oponentes en movimiento controlados por computadora, se debe determinar hacia dónde van.

Hay dos opciones:

1) El enfoque simple, asignando un movimiento determinado que camine de un lado a otro o que se ubiquen en puntos fijos, etc.
2) El enfoque complejo, ¿Qué hacemos si queremos que encuentre un objetivo? R/_____.

Pathfinding

Consiste en el cálculo de una ruta desde la posición actual hasta un objetivo. Una opción es representar *Pathfinding:* con grafos, los cuales pueden ser dirigidos(con pesos en sus aristas) o no dirigidos.

Para encontrar una ruta en un grafo desde un nodo de inicio a un nodo de destino significa encontrar los nodos para recorrer ese camino conectado por aristas de inicio hasta llegar al objetivo, por lo que necesitamos Algoritmos de búsqueda. El peso o costo final se calcularía como la suma de los costos de las aristas que se deben atravesar para llegar al destino.

5. Algoritmos de búsqueda

Todos los algoritmos de búsqueda (que se están viendo en clase) funcionan igual:

1) Almacena una lista de nodos a visitar, llamada "frontera".
2) Toma de la frontera el siguiente "mejor" nodo y agrega todos sus vecinos a la frontera.
3) Si el objetivo(destino) está en la frontera, hemos terminado.

La diferencia radica en que utilizan diferentes estructuras de datos para almacenar la frontera.

Nota: Este esquema también funciona para árboles y para grafos en los que se necesita manejar la posibilidad de que un nodo sea agregado a la frontera por segunda vez.

5.1. Búsqueda en profundidad primero

Funcionamiento general del algoritmo:

1) Elija un orden para comenzar a recorrer el grafo.
2) Almacene la frontera como una pila (*LIFO → Last In, First Out*).
3) Esencialmente, hace esto:

    Visita al primer vecino.
    Visita al primer vecino del primer vecino.
    Visita al primer vecino del primer vecino del primer vecino.

Y así sucesivamente, hasta que encontrar un bucle (continuando con el segundo vecino), o llegar el objetivo.

Limitaciones

- La búsqueda en profundidad primero puede llevar a caminos muy largos.
- Si tuviéramos un gráfico infinito, la búsqueda en profundidad primero probablemente fallará.
- Ignora los costos(pesos) de las aristas.

5.2. Búsqueda en amplitud primero

Guarde la "frontera" como una lista, inicializando el nodo inicio.

Si el objetivo está en la frontera, hemos terminado.

Sino, para cada nodo en la frontera, agregue todos sus vecinos a la frontera.

Repita.

Básicamente, primeramente se agregan todos los vecinos, luego todos los vecinos del vecino, luego todos los vecinos del vecino del vecino, etc.

Limitaciones

- La búsqueda en amplitud primero puede necesitar mucha memoria para recordar la frontera y los caminos para llegar allí.
- Ignora los costos(pesos) de las aristas.

5.3. A* búsqueda

Para encontrar una solución óptima, siga expandiendo los nodos hasta que el nodo objetivo(destino) sea el mejor nodo en la frontera.

A* garantiza encontrar la solución óptima si la heurística es:

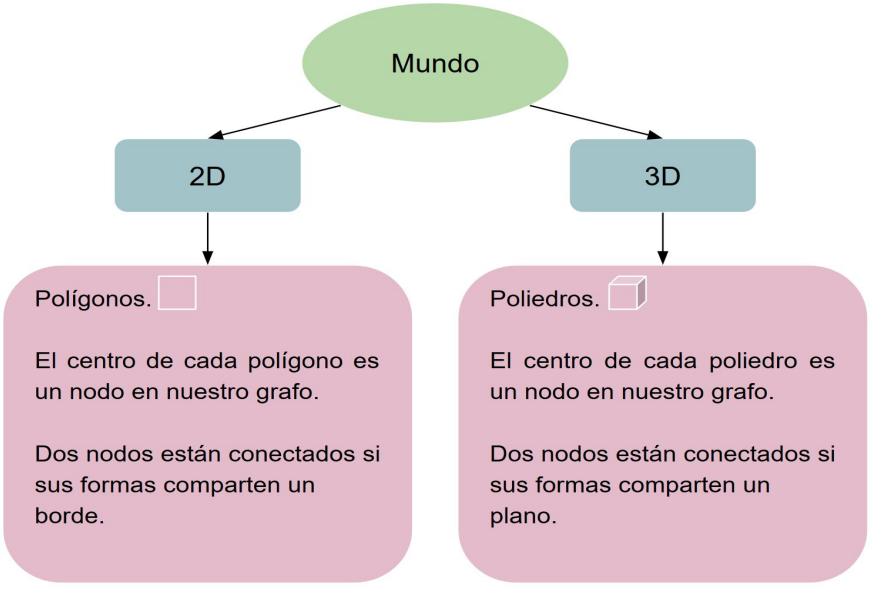
- Admisible: Nunca subestime el costo.
- Consistente: para un nodo “x” y su vecino “y”, el valor heurístico de “x” debe ser menor o igual al de “y” más el costo de ir desde “x” a “y”.
- Toda heurística consistente también es admisible.
- Para utilizar A * se debe definir una heurística.

También puede reducir los requisitos de memoria de A* utilizando la “Profundización iterativa”(***Iterative Deepening***), la cual consiste en comenzar con un límite de profundidad 1, ejecutar Búsqueda en profundidad primero, y si no puede encontrar una ruta, aumentar el límite de profundidad.

Desgraciadamente, los mundos en nuestros juegos no suelen ser grafos.
¿Cómo construir un grafo para nuestro juego? Podríamos tomar cada posición que un personaje puede alcanzar como nodo en un grafo.
La solución a este problema es ***Nav Meshes.***

Nav Meshes

Poliedros Convexos: Dentro de una figura convexa se puede alcanzar desde cualquier otro en línea recta. Así que vamos a cortar nuestro mundo del juego en formas convexas. Unity tiene soporte incorporado para Nav Meshes.



Conductas de dirección(*Steering behavior*)

En nuestro juego podemos asignar diferentes conductas como: buscar*(seek)* un destino, huir*(flee)* de un enemigo, que un personaje llegue a un destino determinado*(arrive)* o que simplemente vague por la zona*(wander)*. En el último caso mencionado, se recomienda limitar con una esfera el espacio por el que puede rondar.

Path following

Si queremos que el agente siga una ruta predeterminada(*Path following*): haga del objetivo(*target*) el primer punto de la ruta y al acercarse se cambia el objetivo(*target*) al siguiente punto de destino y así sucesivamente.

Alternativa, se puede interpolar entre los puntos al punto más cercano al agente. Además, se puede añadir desplazamiento en dirección al camino para que el agente siempre intente volver a él.

Composición

Los comportamientos de dirección*(Steering behavior)* se pueden combinar fácilmente, para ello utilice *Unit vectors.*

Evitar obstáculos

Si se desea evadir un obstáculo defina un comportamiento de dirección*(Steering behavior)* para evitar obstáculos y combínelo con la el comportamiento de la ruta a seguir *(Path following).*

Rebaño(Flocking)

Si tenemos un grupo de personajes que se comportan como un rebaño se implementan tres tipos de comportamientos(Steering behavior): Evasión, Cohesión, Alineamiento.

Interpolación

¿Dónde utilizamos la interpolación? Cuando contamos con dos o más valores y queremos obtener valores intermedios. Los valores pueden ser números, posiciones, rotaciones, colores, entre otros.

¿Cuál es el caso más sencillo de interpolación?

R/_____.

AI Behavior

AI igual que con "game", nadie se a puesto de acuerdo con lo que realmente es "AI"(Artificial intelligence).

Comúnmente se le define como: *"la ciencia de los agentes inteligentes".*

Y ¿Qué es un agente? Cada personaje "inteligente" en el juego es un "agente"
Los agentes determinar qué acción realizar con base a su conocimiento en el juego.

6. Técnicas para utilizar AI en juegos

6.1. Árboles de decisión

Los estados puede ser representado en árboles, donde cada rama es una declaración “si” basada en alguna condición. Las hojas de los árboles representan acciones que el personaje puede tomar dependiendo de la situación en la que se encuentre.

Limitaciones

- ❏ No hemos cambiado nada de las declaraciones condicionales.
- ❏ Diseñar un árbol de decisiones implica mucho trabajo manual.
- ❏ Tampoco hay persistencia, el agente puede decidir un nuevo comportamiento cada vez que se evalúe el árbol.

Ventaja

- ★ Los árboles de decisión se pueden aprender (a veces) con técnicas de *Machine Learning*.

6.2. Máquinas de estados finitos *(Finite State Machines)*

Podemos hacer que nuestro código sea más agradable si separamos las decisiones y el comportamiento, en donde:

- Cada comportamiento es un "estado".
- El personaje decide cuándo cambiar de estado.

Limitaciones

- ❏ No hay un concepto real de "tiempo", tiene que ser "agregado".
- ❏ Si se desea agregar un estado, debe determinar cómo se relaciona con cada otro estado.
- ❏ Es difícil tener más de una máquina de estado finito.
- ❏ Se dificulta reutilizar las subpartes.

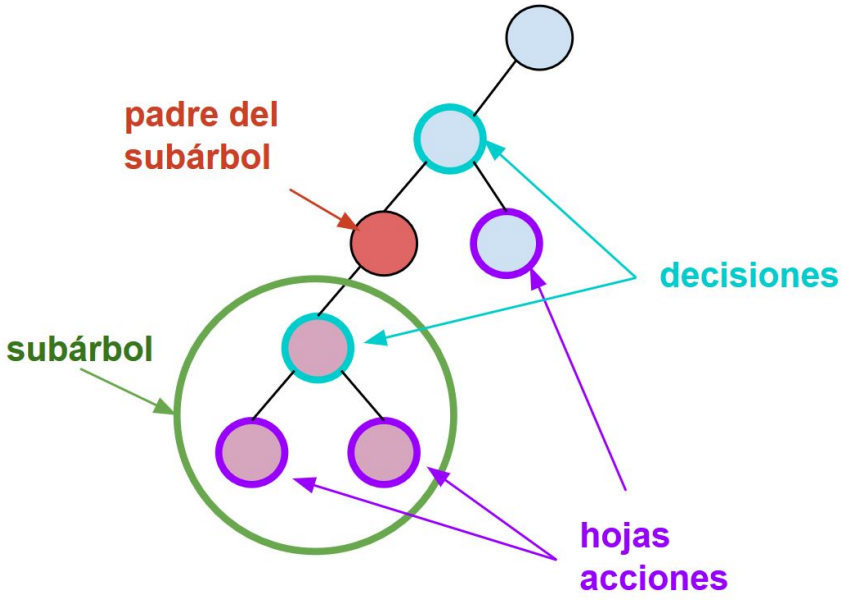
6.3. Máquinas de estado finito jerárquico

Las máquinas de estados finitos definen el comportamiento del agente.
Los nodos son comportamientos → cada nodo es otra sub-máquina.
Esto conduce a cierta reutilización y facilita la creación.

6.4. Árboles de comportamiento *(Behavior Trees)*

Son grafos hechos árboles, si en estos tenemos subárbol, solo debemos preocuparnos por la conexión de el subárbol con su nodo padre.
Las **hojas** del árbol serán las **acciones reales**.
Los **nodos interiores** definen las **decisiones**.

Nota: Esto es **extensible** (nuevos tipos de nodos), fácilmente **configurable** (solo une diferentes nodos para hacer un árbol) y **reutilizable** (los subárboles pueden usarse varias veces)



Tipos de nodos:

- Elección*(Choice)*: Ejecutar el primer hijo que tenga éxito.
- Secuencia(Sequence): Ejecutar todos los hijos hasta que uno falle.
- Bucle(Loop): Sigue ejecutando hijo(s) hasta que uno falle.
- Elección aleatoria(Random choice): Ejecutar uno de los hijos al azar.
- Tiempo(Timing): Ejecuta el primer hijo durante x segundos, luego el segundo hijo, etc.

Planificación

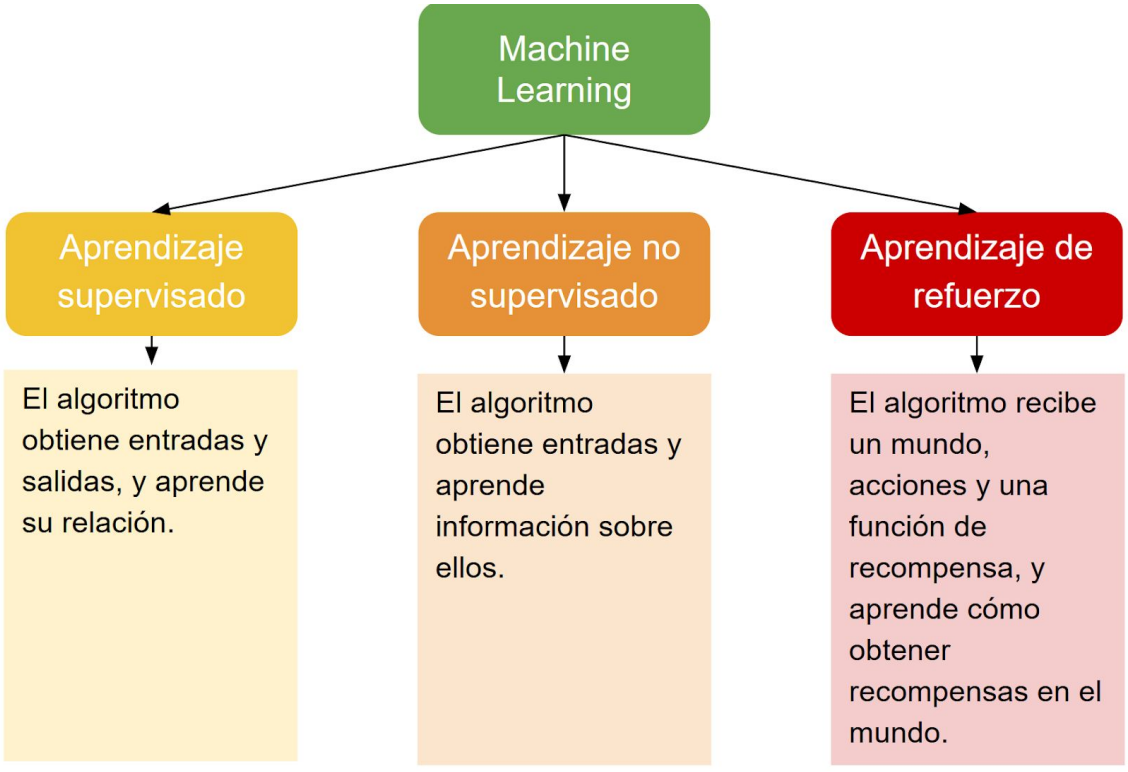
Hasta ahora solo hemos visto cómo podemos ensamblar comportamientos de forma **estática:**¿Qué pasaría si queremos que el agente decida una estrategia?
En ese caso se le concede un objetivo al agente y deje que descubra cómo alcanzar ese objetivo.

> *Dado el estado actual del juego, una lista de acciones y un objetivo, encuentra una secuencia de acciones que conduzcan al objetivo.*

Nota: Cualquier vértice en el que se cumpla el objetivo, es nuestro objetivo*(goals)*.
La planificación realmente no ayuda si otro personaje, o el jugador interfiere.
Los adversarios al igual que el jugador, eligen opciones para que el agente no logre su objetivo.
*Ver la parte práctica en las diapositivas para **MiniMax** y **Alpha-beta Pruning.***

Aprendizaje automático*(Machine Learning)*

Se pretende proporcionar algunos datos/experiencias a un algoritmo, y deriva la forma de resolver un problema. Generalmente estos algoritmos necesitan optimización.



Procedural Content Generation(PCG)

Hacer juegos requiere mucho esfuerzo: codificación, arte, música, historia, etc. Algunos de estos procesos de creación pueden automatizados o asistidos: Procedural Content Generation (PCG).

Sin embargo, hay 2 problemas:

1. Generar contenido no siempre es fácil.
2. Generar buen contenido suele ser incluso más difícil.

¿Qué podemos generar?

- ➔ Niveles
- ➔ Enemigos
- ➔ Misiones
- ➔ Diálogo
- ➔ Narrativa
- ➔ Plantas
- ➔ Objetos 3D
- ➔ Reglas
- ➔ Juegos

7. Enfoques basados en la búsqueda (Search-based approaches)

7.1. Pathfinding: A veces queremos "encontrar" un camino que aún no existe(cueva, laberinto, entre otros) o para conectar dos ciudades con carreteras.

¿Qué pasa si cada nodo en nuestro grafo es un "nivel" para un juego? Podemos conectar dos niveles con su diferencia, si sabemos diferenciar de entre nuestros niveles dentro del grafo cuáles son un “buen nivel”.

7.2. Generación de historias: Podemos hacer que la computadora genere una historia. Las historias son secuencias de acciones que llegan a una conclusión.
Se parece mucho a la **planificación**. El problema es que en la planificación se llegue a encontrar una solución eficiente.

Enfoques basados en solucionadores (Solver-based approaches)

¿Qué sucede si conocemos ciertas propiedades que debe tener un activo "bueno" de nivel / Arte? Si expresamos nuestro problema de diseño de nivel de una manera lógica, estas propiedades se convierten en restricciones en la fórmula(que pueden ser asignadas con variables lógicas con ayuda de alguna herramienta).

¿Por qué la física?

La física se utiliza para una amplia variedad de propósitos en los videojuegos, muchos video juegos reflejan entornos realistas y los jugadores esperan lo mismo de la física (que sea una aproximación de la física real).

Las leyes del movimiento de Newton.

- ★ Primera ley: los objetos en movimiento permanecen en movimiento, los objetos en reposo permanecen en reposo, a menos que una fuerza actúe sobre ellos.

- ★ Segunda ley: la suma de fuerzas sobre un objeto es igual a su masa multiplicada por su aceleración.

- ★ Tercera ley: Toda fuerza tiene una fuerza igual y opuesta.

Bonificación: La ley de la gravitación universal: cada dos cuerpos se atraen entre sí con una fuerza igual al producto de sus masas dividido por el cuadrado de su distancia.

Leyes de Newton

$$\sum \vec{F} = 0 \Leftrightarrow \frac{\mathrm{d}\,v}{\mathrm{d}\,t} = 0$$

$$\vec{F} = m \cdot \vec{a}$$

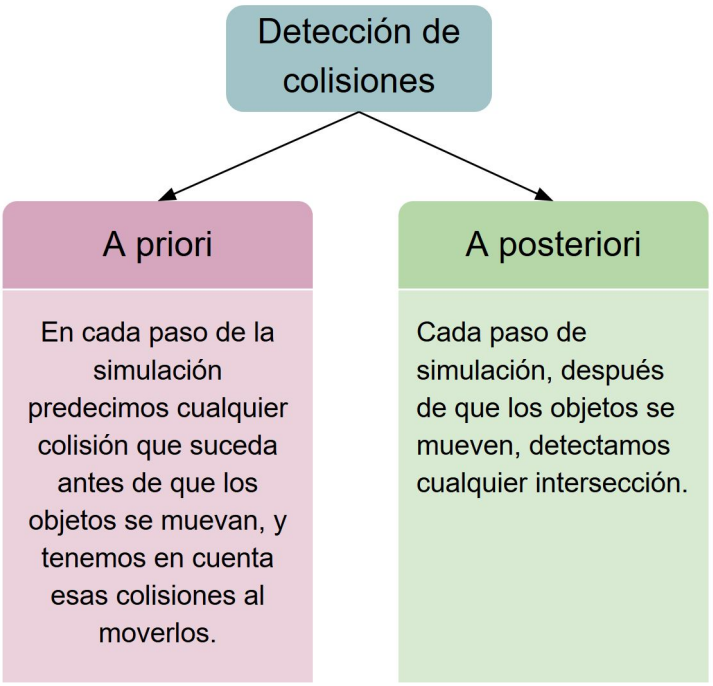
$$\overrightarrow{F_A} = -\overrightarrow{F_B}$$

$$\overrightarrow{F_G} = G \cdot \frac{m_1 m_2}{r^2}$$

Detección de colisiones

Consiste en: ¿Cómo detectar si dos objetos chocan entre sí?
Al igual que con el movimiento, una aproximación de una colisión realista es suficiente.

Tenemos que detectar colisiones "en el camino". Dos opciones:



Nota: La detección de colisiones a posteriori es mucho más simple, y se usa típicamente en juegos.

Networking

Los jugadores disfrutan interactuando con otros jugadores.

- ❖ A los Killers les gusta competir con gente real.
- ❖ A los Achievers les gusta tener tablas de ranking y mostrar sus logros.
- ❖ A los socializadores les gusta interactuar con otras personas.

Requisitos de red

- Rápida: no debería afectar negativamente el juego.
- Confiable: los problemas de conexión no deben arruinar el juego.
- Regular: ningún jugador se ve afectado por problemas de conexión de otro jugador.
- Segura: los jugadores no pueden hacer trampa.

Peer-to-Peer Games

¿Cómo podemos sincronizar los movimientos, para que todos los jugadores tengan el mismo estado del juego?

Lockstep Networking

Asumamos que nuestro juego tiene "turnos".Cuando es el turno de un jugador, envían la acción que realiza a todos los demás, sincronizando los estados del juego.

Limitaciones

- El juego debe ser determinista.
- Los problemas de conexión producen tiempos de espera notables.
- Los problemas de conexión de cualquier jugador afectan a todos.

Quick Aside: Reproducibility

Los estados de juego también pueden volverse inconsistentes por razones benignas.
Por ejemplo: la simulación física puede producir diferentes resultados.

Juegos cliente-servidor

En lugar de dejar que cada jugador haga un seguimiento del estado del juego, usar un servidor central, siendo el servidor la autoridad sobre la corrección del estado del juego, los clientes(jugadores) solo lo muestran.

Cuando un jugador desea realizar una acción, se envía una solicitud al servidor, y la acción solo se realiza si el jugador tiene permiso para hacerlo.

Limitaciones

- El servidor debe responder a las solicitudes de los clientes rápidamente(pueden ocurrir retrasos).
- Los clientes deben (idealmente) no saber nada que el jugador no deba saber

Predicción del lado del cliente

En el modelo original, si presiona un botón, tiene que esperar a que el servidor responda para moverse realmente, pero el cliente podría predecir que la pulsación del botón causará movimiento(será aceptada por el servidor), el servidor confirma las coordenadas y el cliente se sincroniza con esa información.

Problema: el cliente le informa al servidor acerca de las actualizaciones "en el pasado".

Dead-Reckoning

La primera ley de Newton indica que los objetos en movimiento permanecerán en movimiento (a menos que una fuerza actúe sobre ellos), el cliente podría asumir que eso aplica también para objetos que no son jugadores.

Cuando el servidor envía otra actualización, el cliente utiliza la información real con su predicción, podría usar la mecánica del juego para predecir cambios en el movimiento.

> Por ejemplo: si un personaje avanza, el cliente puede seguir avanzando hasta que el servidor indique lo contrario.

Lag Reduction

El servidor realiza un seguimiento de pasados estados del juego.

Cuando un jugador realiza una acción, el servidor la aplica al **estado del juego** en el momento adecuado(en el pasado). Esto significa que el servidor también debe volver a aplicar las acciones de otros jugadores que suceden después.

Ventaja: Mayormente, la acción de un jugador, tendrá el efecto que ve en su pantalla.

Trade-off

Este sistema es justo: ambos jugadores tienen las mismas posibilidades.

Favorece el comportamiento activo, porque las propias acciones de un jugador se reflejan en el estado del juego con mayor facilidad que las acciones que se realizan en el jugador, la demora tampoco es lo suficientemente grande como para que los jugadores la noten, ya que también necesitan algo de tiempo para procesar lo que les está sucediendo.

El mantener un registro de los estados históricos es costoso para muchos jugadores.

Cheating

No se puede confiar en los jugadores, podrían hacer trampa y no hay un método que sea 100% efectivo para encontrar todas las formas de trampa, pero algunas cosas que se hacen son:

- Escanear binarios del juego en busca de modificaciones.
- Busque software de trucos conocido y revisar si se está ejecutando.
- Ejecutar el juego en una *sandbox* en la que no se puede hacer trampa.
- Comprobar que no hayan modificaciones/configuraciones (que no deberían existir) tarjeta gráfica.

Escalabilidad

Los juegos varían en el número de jugadores si solo se tienen pocas docenas de jugadores, un servidor es suficiente.

Servidores de juego MMO

- ❏ Normalmente el juego se divide en mapas / regiones / áreas
- ❏ Cada área puede estar en múltiples servidores físicos.
- ❏ Cuando un jugador se mueve de un área a otra, son transferidos al servidor para esa área.
- ❏ Dos jugadores en la misma área todavía pueden estar en diferentes servidores, cuando un servidor se llena demasiado, se puede dividir en dos ("fragmentación").
- ❏ Tener múltiples servidores ahorra recursos.

- ❏ A medida que el juego se desenvuelve, algunas áreas se abandonan.

Las compañías de MMO han comenzado a usar conceptos como "mega-servidores" o "fragmentación (dinámica)" para combinar jugadores de múltiples servidores en áreas de baja población.

Player Analytics

Podemos recopilar datos mientras los jugadores juegan, para que estos datos puedan ser analizados para responder preguntas/confirmar hipótesis, siendo informadas al game designer o tal vez, sólo para saber qué tipo de contenido debería hacer más para que los jugadores no se aburran, manteniéndolos entretenidos para que sigan jugando e incluso ganar dinero.

Player Ratings

Imaginemos que se tiene un juego multijugador, los jugadores son(competitivos) y quieren saber lo buenos que son, pero tampoco quieren jugar contra otro jugador que tenga una diferencia de fuerza mucho más alta o más débiles que ellos. Sin embargo, si un jugador le gana a un jugador más fuerte, debería contar más que si le gana a un jugador más débil.

ELO

- ❖ Es un sistema de clasificación llamado donde a cada jugador se le asigna una calificación numérica.
- ❖ Cuando dos jugadores juegan, la diferencia en su calificación actual se usa para calcular sus "posibilidades estimadas de ganar".
- ❖ Cada partido vale un número de puntos.
- ❖ Al final del partido, el jugador ganador obtiene un porcentaje de esos puntos, igual a las posibilidades estimadas de perder del jugador perdedor.
- ❖ Un jugador que tenía menos probabilidades de ganar (debido a que su calificación era mucho más baja que la de sus oponentes) puede obtener más puntos por ganar, que un jugador que compite con alguien del mismo rango.

Limitaciones

- Las calificaciones de los jugadores solo proporcionan una estimación de las probabilidades de ganar, pero no nos dicen qué tan buena es esa estimación.
- Los jugadores pueden ser incentivados a mantener su calificación.
- El ELO en juegos de equipo, podría no reflejar con precisión la habilidad de cada jugador.
- Los nuevos jugadores deben tener alguna calificación, que se puede abusar *("smurfing")*.

Notas: Los sistemas de clasificación **Glicko**/**Glicko-2** constan de dos números:la calificación y la confiabilidad, esta adición ayuda a comparar mejor la habilidad "verdadera" de dos jugadores.
Al crear *matches* para jugar, hay que tener en cuenta varios aspectos:

1. Calificaciones*(Ratings).*
2. Equipos.
3. *Parties.*
4. Tiempos de cola*(Queue Times).*

Dos enfoques principales:

- *Timed Queues.*
- *On-demand matches.*

Retención del jugador

★ Es importante una buena primera impresión/nueva experiencia para el jugador.
★ Buena curva de desafío/flujo.
★ Recompensas demoradas (Ejm: bono de inicio de sesión diario).
★ Dar al jugador personalización/opción progresión.
★ Características sociales(Social features), invitaciones y recompensas de juego en equipo (ver también: adquisición de jugadores).

Modelado de jugadores(Player Modeling)

Se puede personalizar el contenido, o simplemente centrarse en los tipos más comunes de jugadores ¿Cómo se determinan los "tipos" de jugadores?
R/ Tu juego puede rastrear las acciones del jugador, especialmente en juegos de mundo abierto, determinar en qué gastan su tiempo los jugadores es beneficioso para enfocar el desarrollo futuro o guiar a los jugadores a áreas poco exploradas.

Perfiles de jugadores

Por otro lado, también quieres saber quién juega tus juegos. ¿Por qué? (por ejemplo, para que puedas orientar anuncios a la demografía de tu juego).

- La región geográfica la puedes obtener con geolocalización de la IP.
- Puedes averiguar rango de edad (Proporcione un bono de cumpleaños).
- Los datos personales puedes obtenerlos incentivando que se conecten a redes sociales(Facebook).
- Los patrones de juego también son importantes: hora del día, duración de la sesión, cantidad de juegos por semana, entre otros.

Nota: Los modelos estadísticos han demostrado que el estilo de juego y la personalidad están relacionados.

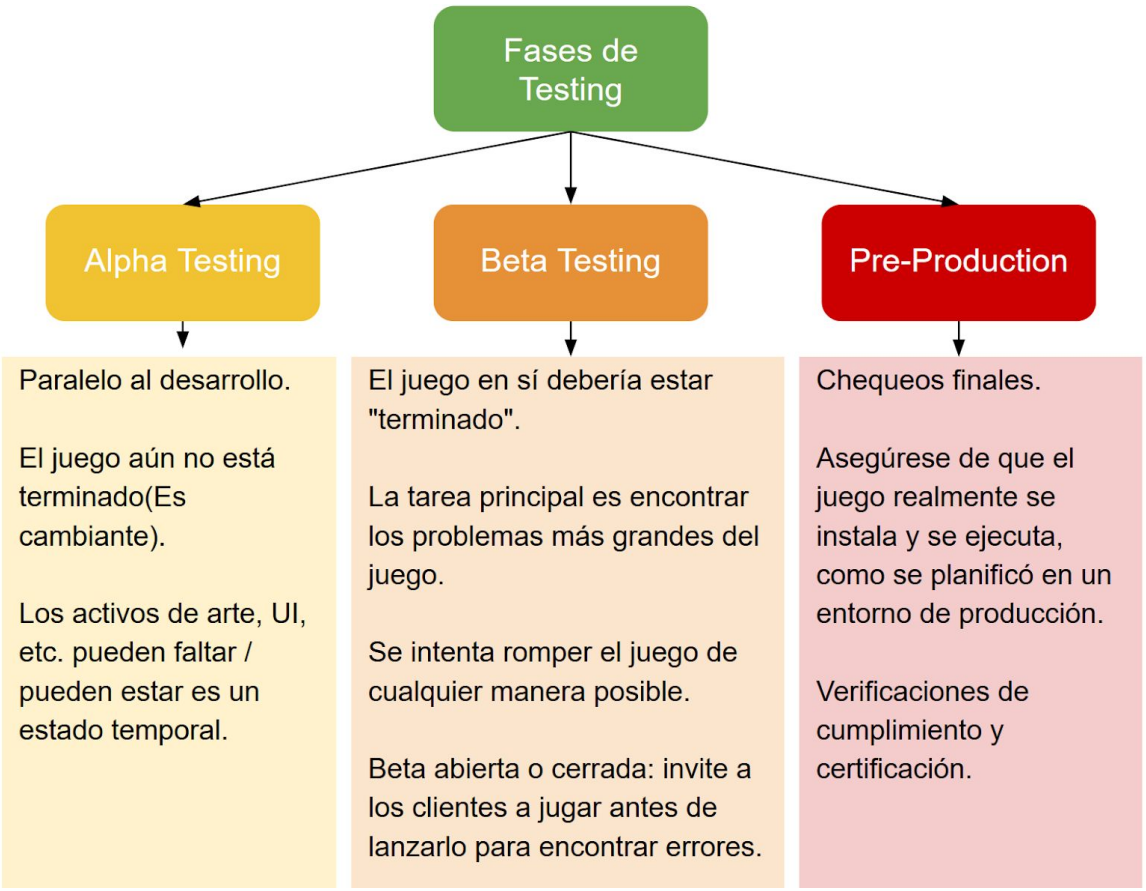
Machine Learning for Player Analytics

- ❏ Se pueden utilizar algunas técnicas de aprendizaje automático(**machine learning**) para analizar los datos recopilados.
- ❏ Por ejemplo, si queremos saber qué "tipos" de jugadores existen, podemos usar **clustering.**
- ❏ Si queremos predecir qué es probable que el jugador haga, podemos usar el aprendizaje supervisado (***supervised learning***), por ejemplo, una red neuronal.
- ❏ También podemos conocer las preferencias de un jugador, por ejemplo, qué arma es probable que prefiera.

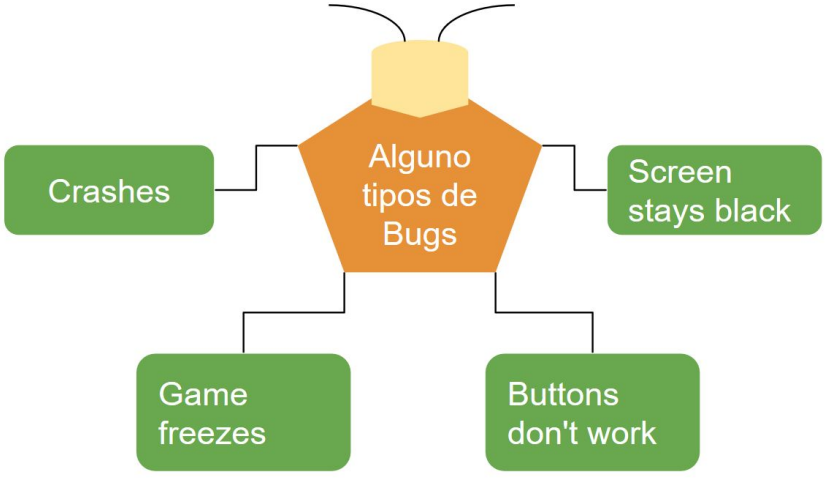
¿Cómo utilizar esta información en Game Play?

- Si sabes que tus jugadores van a preferir cierto tipo de arma, puedes concentrarte en hacer más de estas armas en el futuro.
- Incluso puede usar los modelos aprendidos, durante el desarrollo de futuras versiones, para predecir a dónde irán más los jugadores.
- Los tipos de jugadores pueden informar el tipo de contenido en el que deberías enfocarte.

Testing and Balancing



En el desarrollo independiente*(indie)* puede que las etapas no esten tan definidas, es importante implementar testing tempranamente y estarlo probando más continuamente, al trabajar más de cerca con la comunidad de jugadores.



Errores de juego: Más allá de los errores de software "clásicos", los juegos, como sistemas complejos, también tienen muchas otras fuentes de errores.
Por ejemplo: ¿Cómo juzgas si un juego es "fácil" o "difícil"?

Game Play Bugs

Se pueden encontrar problemas con la mecánica a través de las pruebas de Game Play. Para encontrar posibles exploits, los testers deben ser creativos/pensar como un adversario.

Problemas de física*(Physics Problems)*

Algunos problemas pueden ser "poco realista".
Desafío: ¿Cómo se cuantifica lo que es considerado "poco realista"?

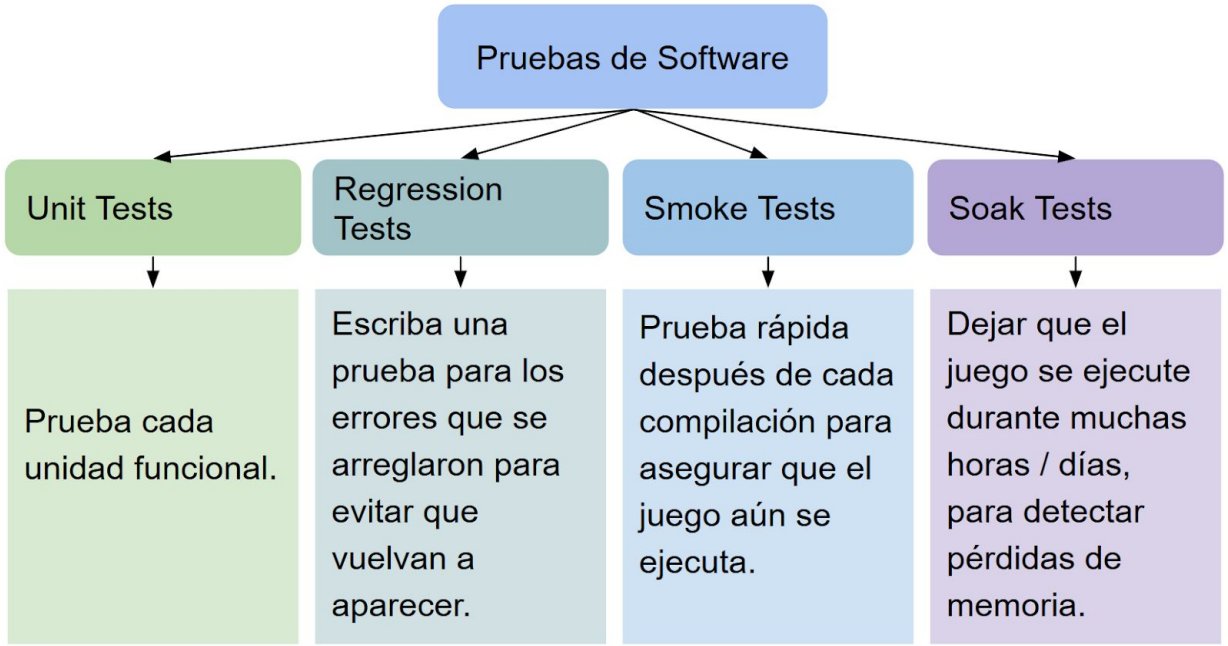
Problemas de IA

Incluso más difícil que determinar si la física es "buena", es determinar si los agentes de AI son "buenos".

Los agentes de la IA no se comportan como los humanos, puede que ni siquiera sean "los mejores", pero deberían ser lo "suficientemente buenos".

Pruebas de software

Puede utilizar las mismas técnicas que para las pruebas regulares de software.



Monkey Testing

Ingresa información aleatoria en tu juego y asegúrate de que "nada malo" suceda. Solo asegúrate de registrar la entrada para que puedas reproducir los errores.

Juego AI

Crea un agente de inteligencia artificial que juegue durante el juego (no es necesario que sea perfecto), de hecho, es mejor si no es perfecto, e intenta tácticas inusuales.

Si una mala AI puede vencer a tu juego, también tus jugadores.

Balancing

- ❏ La mejor base para el balance es la matemática.
- ❏ Uno quiere saber la fuerza relativa de cada objeto e unidad en el juego.
- ❏ Recomendación: Cree una hoja (o código) de Excel que le diga qué tan buena es cualquier combinación de elementos.
- ❏ Modifique los números en la hoja de Excel(o código) hasta que obtenga los resultados que desea y pruébelo.

Generación de números aleatorios*(Random-Number Generation)*

Muchos jugadores odian la generación de números aleatorios ***(RNG)*** en los juegos. Pero en realidad sirve para propósitos importantes. Por ejemplo:

La aleatorización ayuda con:

- ❖ La repetibilidad: Muchos juegos usan niveles aleatorios para mantener las cosas interesantes. Es utilizado en los juegos de cartas para barajar.
- ❖ La participación y retención del jugador: Para retener a los jugadores es bueno dar recompensas, pero resulta aburrido si el jugador sabe exactamente cuando las obtiene, así que sorprenderlos, dando mejoras con cierta aleatoriedad los incentiva a seguir jugando.

Desventaja: RNG es bueno para el juego, pero demasiada aleatoriedad hace que los jugadores se sientan mal.

Referencias:

Todo el material utilizado fue extraído de las diapositivas vistas en clase por el profesor Dr.Markus Eger
https://yawgmoth.github.io/CI-2807/slides/