# Intro Desarrollo de Videojuegos 2023 - UNQ v.II

5 | Arquitecturas y paradigmas en engines de videojuegos. Patrones de programación para Videojuegos. IA.

Universidad Nacional de Quilmes

# Los 4 tipos de jugador.

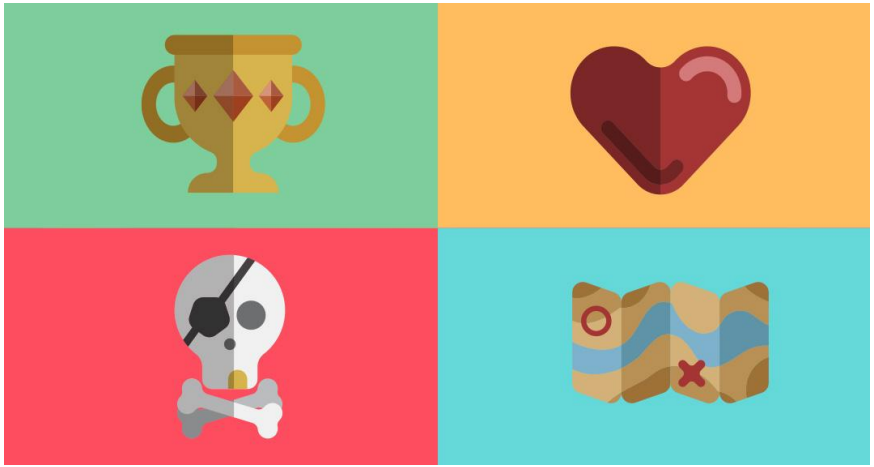
La mejor manera de establecer una buena comunicación con tus usuarios es conocerlos bien.
Dentro de las clasificaciones de perfiles de jugadores podemos encontrar diferentes definiciones. Una de las más conocidas es la clasificación de **Richard Bartle**, escritor y profesor británico que se especializa en game design e investigaciones dentro de la industria de los videojuegos.

Universidad Nacional de Quilmes

# La Matriz de Bartle.

Bartle clasifica a los jugadores en cuatro grupos principales:

- Killers
- Achievers
- Socializers
- Explorers



Universidad Nacional de Quilmes

# La Matriz de Bartle.

**Los Killers:** son los jugadores más acérrimos. Aquellos que se toman muy en serio el hecho de jugar y competir. Pueden incluso tener hardware (equipos) específico para optimizar su rendimiento y son grandes consumidores de la industria de videojuegos. Los killers quieren ser los mejores, incluso algunos de ellos compiten en torneos. Por eso los mejores elementos para atraerlos son las tablas de posiciones y los rankings.


Universidad Nacional de Quilmes

# La Matriz de Bartle.

**Los Achievers o Recolectores:** ellos solamente quieren conseguir *status* dentro del juego. Si hay logros, quieren desbloquearlos todos. Son los que vuelven a jugar cada nivel para lograr conseguir las tres estrellas y el desafío es consigo mismos. Las mejores herramientas para lograr atraer a los achievers son justamente los logros, los bagdes (insignias), las estrellas y las recompensas. Ellos no pueden ver que hay una lista de elementos con un candado, quieren desbloquearlo todo.


Universidad Nacional de Quilmes

# La Matriz de Bartle.

**Los Socializers o Sociales:** no son tan competitivos como los killers ni tan perfeccionistas como los achievers. Ellos quieren hacer sociales. Se suman a un juego porque un amigo los invitó. Son los que te envían un ladrillo si es que te estaba faltando uno mientras construís tu ciudad virtual en Cityville, o que comparten y viralizan contenidos. A los socializers llegamos mediante herramientas sociales como los chats, las listas de amigos y las novedades.


Universidad Nacional de Quilmes

# La Matriz de Bartle.

**Los Explorers o Exploradores:** ellos están aquí para ver de qué se trata. Curiosos por naturaleza, quieren conocer el mundo. Están de paso en tu juego y probablemente lo abandonen para explorar otro. No son perfiles competitivos ya que están ocupados explorando para nutrir su conocimiento. Aunque pueden transformarse en alguno de los otros perfiles. Para atraerlos podemos agregar sorpresas, recompensas difíciles (hallazgos) y elementos atractivos desde el comienzo del juego.


Universidad Nacional de Quilmes

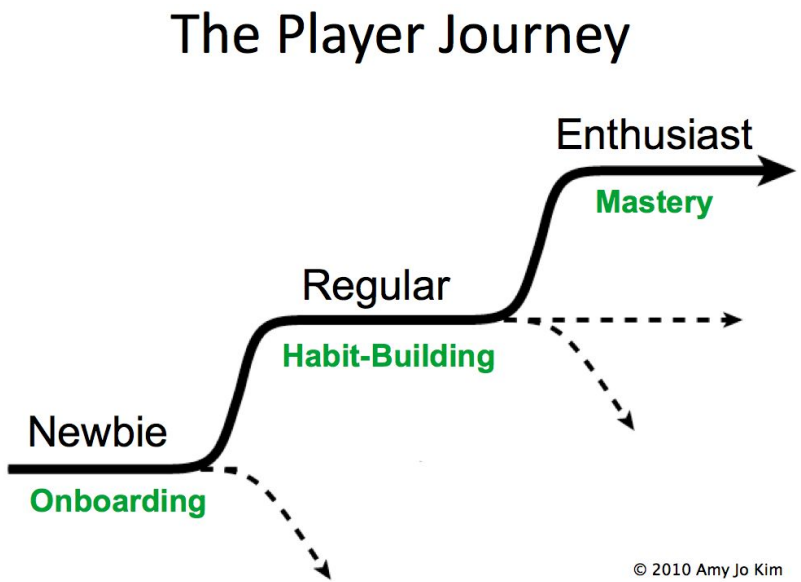
# La Matriz de Bartle.

Para desarrollar el guión de un videojuego o el enfoque de un sistema gamificado debemos tomar en cuenta esto. También utilizamos esta información para delimitar el producto final para nuestro target específico.
Cada tipo de jugador posee sus preferencias en cuanto a géneros y estilos, muchas veces pudiendo coincidir varias de estas en más de un perfil.


Universidad Nacional de Quilmes

# The player journey.

La ruta del jugador describe la travesía que emprende éste desde el momento en que tiene su primer acercamiento al juego hasta el momento en que se vuelve un experto.
La progresión en el tiempo.
Se divide en distintas etapas relacionadas con el grado de expertise del usuario.

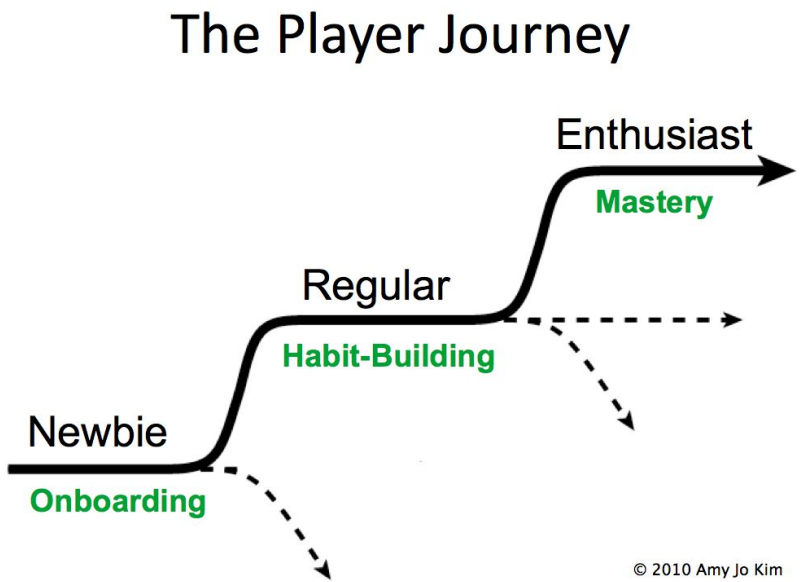



Universidad Nacional de Quilmes

# The player journey.

Intenta clasificar a los jugadores en base a conjuntos de habilidades adquiridas y aprendizajes logrados durante la ruta y poder direccionar mejor la comunicación, adaptándola a su estadío.

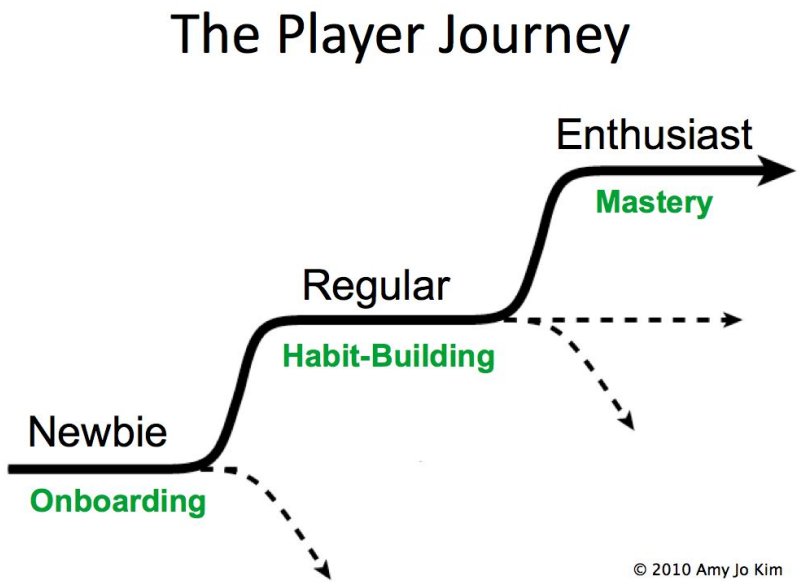
The Player's Journey es también un framework que se divide en tres, cuatro o cinco estados (dependiendo del autor).




Universidad Nacional de Quilmes

# The player journey.

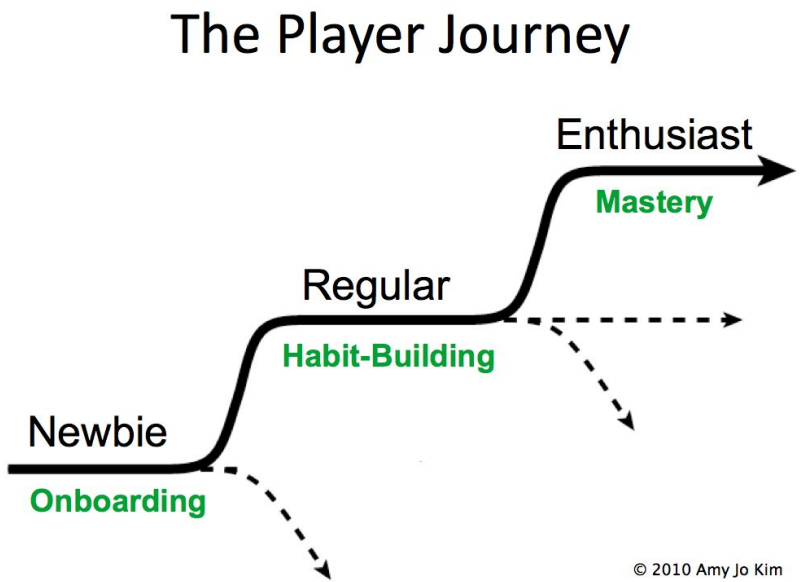
**Onboarding:** comenzamos por un estadío inicial donde se va adentrando en el mundo ficticio, empieza a conocer los escenarios y los objetos, por ejemplo, se va familiarizando con el ambiente y va incorporando habilidades.




Universidad Nacional de Quilmes

# The player journey.

**Scaffolding:** un segundo estadío donde domina las habilidades básicas que inicialmente no dominaba y logra mantenerse, puede ser por ejemplo que se convierta en un usuario regular del producto.

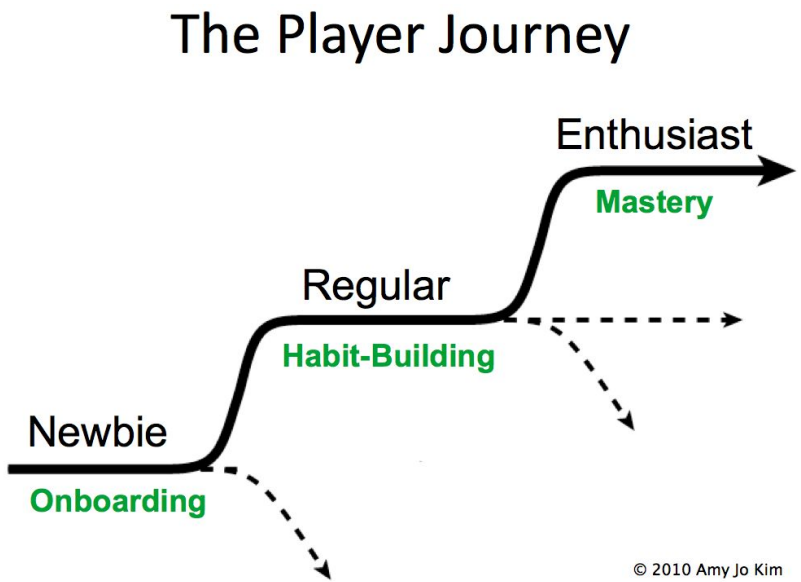



Universidad Nacional de Quilmes

# The player journey.

**Mastering:** para cerrar con un tercer estadío donde ya se ha familiarizado con el juego, domina los controles, conoce trucos y le queda masterizarlo, convertirse en un experto.

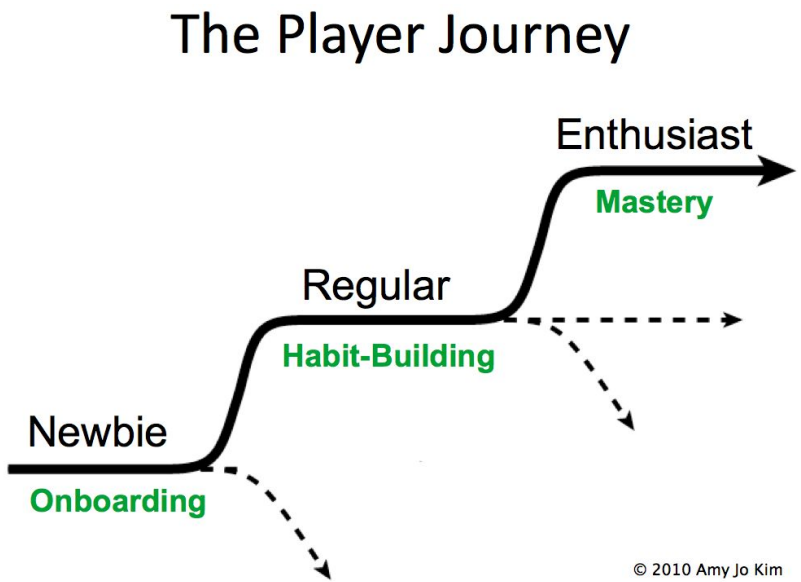
Ami Jo Kim lleva esta división mientras que Yu-Kai-Chou agrega una cuarta etapa que es la de **finalización del juego (end game).**




Universidad Nacional de Quilmes

# El camino a la maestría.

Según **Amy Jo Kim** podemos identificar tres estadíos del jugador: **Newbie. Apprentice. Wizard.** Es decir: novato, aprendiz o regular y experto o entusiasta.
Se podría decir que estos tres estadíos se encuentran directamente relacionados con los tres escenarios que hemos descrito en el ciclo del juego.




Universidad Nacional de Quilmes

# Diseñando la ruta del jugador.

Tener en cuenta las “motivaciones intrínsecas”:
-**Pertenecer:** sentirse parte integrante de un grupo.
-**Aprender:** ¿qué le propones aprender al jugador?
-**Divertirse:** estrategia de game design y una buena historia. Al final, el objetivo del juego es la diversión.

Y las “motivaciones extrínsecas”:
-Ganar **dinero**.
-Obtener **bonus** extras.
-Completar **tareas**.

Universidad Nacional de Quilmes

# Diseñando la ruta del jugador.

Identificar el perfil de jugador según la matriz de Bartle y el estadío según la propuesta de Ami Jo Kim.

Un jugador **killer experto** va a esperar una experiencia más especial que un killer que recién está en onboarding.

Los **exploradores** muchas veces se quedan en onboarding.

El **achiever** repite el ciclo muchas veces, hasta conseguir todos los logros.

Universidad Nacional de Quilmes

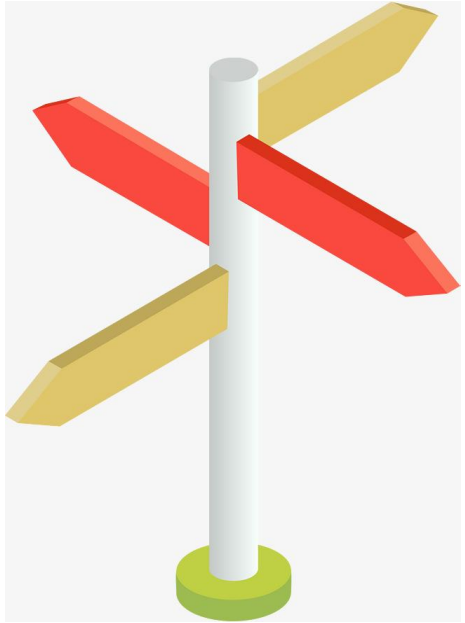
# Diseñando la ruta del jugador.

Hemos aprendido sobre los elementos de **game design** y las formas de aplicarlos en gamification. Luego conocimos los **tipos de jugadores** y su relación con el juego a través de la matriz de Bartle. En este capítulo hemos mencionado que todos esos jugadores pasan por un proceso denominado **“el camino a la maestría”** en el cual atraviesan los escenarios clave del ciclo del juego.

Universidad Nacional de Quilmes

# Design choices (diseño de opciones).

En *gamification* llamamos *design choices* a las opciones o **posibles soluciones de un problema.** Cuando estamos jugando un juego nos topamos generalmente con un punto donde se plantea algún conflicto que debemos resolver. Esos **desafíos** deben ser interesantes.
Para ello, debemos ofrecer al jugador múltiples caminos u opciones.
Veamos cómo hacerlo...

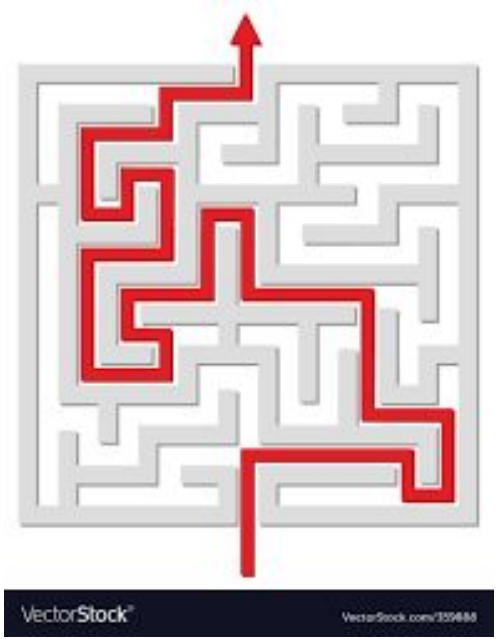


Universidad Nacional de Quilmes

# Design choices (diseño de opciones).

**-Motivación:** ¿Cuál es el valor que le entregamos al usuario? ese valor que lo motivará a “jugar” o no. Consiste en transformar lo aburrido en divertido, crear conexiones emocionales y trabajo en equipo.

**-Meaningful choices o decisiones importantes:** ¿Las actividades que le proponemos realizar son los suficientemente interesantes? ¿O le estamos proponiendo que cliquee en 30 anuncios para desbloquear un badge?

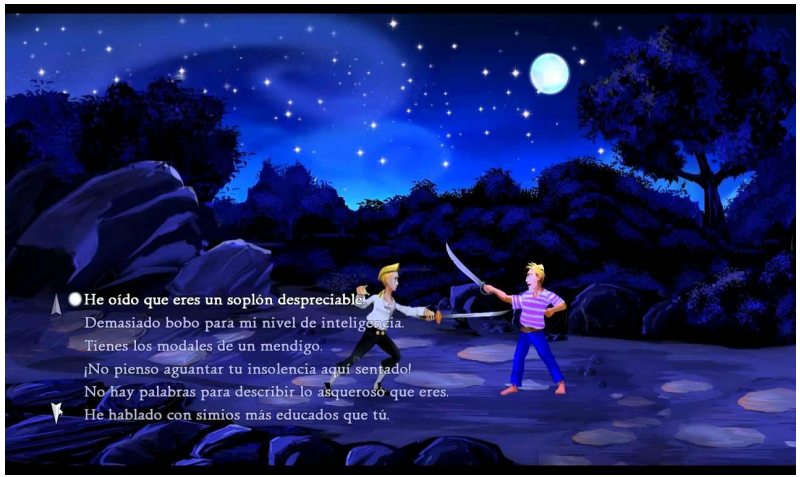



Universidad Nacional de Quilmes

# Design choices (diseño de opciones).

**-Estructura:** los comportamientos que buscamos que tengan los usuarios, ¿Pueden ser diseñados mediante reglas y/o algoritmos? (por ejemplo: ganar puntos por compartir algo).

**-Conflictos potenciales:** ¿Podrá el juego evitar entrar en conflicto con las estructuras? Por ejemplo el conflicto entre: ¿Lo hago porque me gusta jugar o por ganar puntos?




Universidad Nacional de Quilmes

# Design choices (diseño de opciones).


“Un juego es una colección de decisiones importantes”.
Sid Meier, creador de la saga de juegos de estrategia *Civilization*.

Universidad Nacional de Quilmes

# ¿Qué es un Motor de Videojuego?

En su forma más básica, un motor de videojuegos es el software base que nos permite crear y correr un videojuego, haciendo uso de rutinas, funciones y un main loop para el procesamiento de los gráficos y la lógica de interacción.

Usualmente eran custom-made, creados con un juego en mente y optimizados para tal


BASIC ofrece librerías y un loop principal, además de una memoria donde cargar funciones

Los motores de videojuegos modernos (principalmente los de Propósito General, como Godot, Unity o Unreal) ofrecen todo un set de herramientas visuales y librerías que permiten agilizar el desarrollo y facilitar la exportación multiplataforma


Retrospectiva de juegos a 50 años de BASIC

# Paradigmas y Arquitecturas en Game Engines

¿A qué nos referimos cuando decimos “la API de Godot es orientada a Objetos, y el motor es Orientado a Servidores”?

¿Y cuando se dice que “Unity usa un paradigma de Entidad-Componente”?

Hablamos de los principios en los que se basa el armado de los sistemas que componen al software base, sus interrelaciones y la manera en la que interactúan con el usuario desarrollador para la construcción del software final.

Paradigmas existen muchos, pero todos los motores necesitan componentes y una estructura base para funcionar.

¿Se les ocurren algunos?

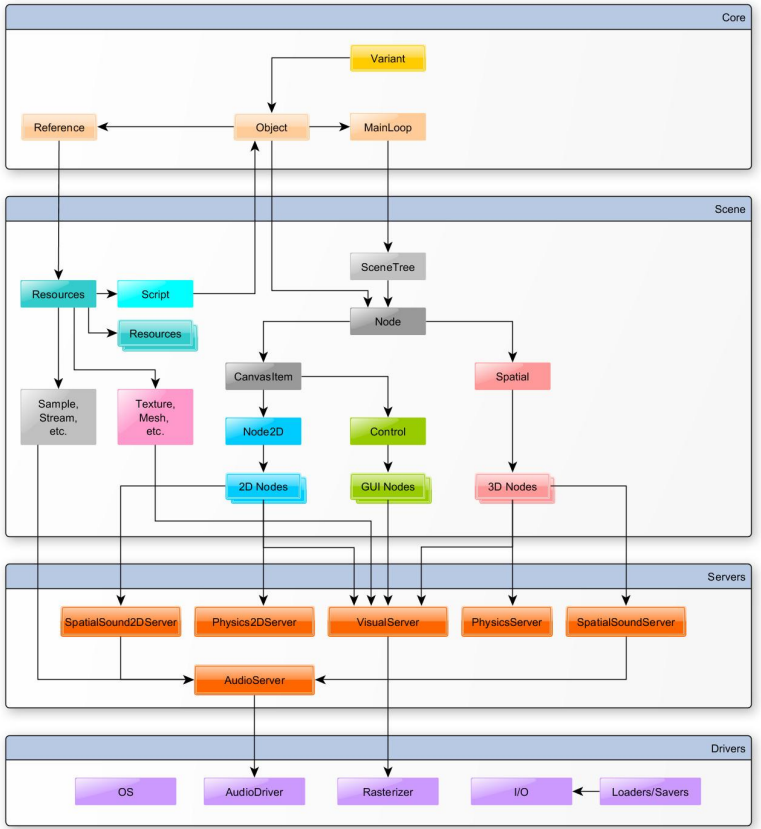
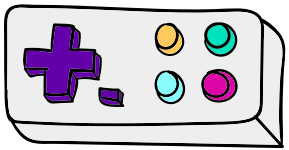
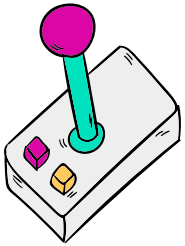



Diagrama de la arquitectura que utiliza Godot

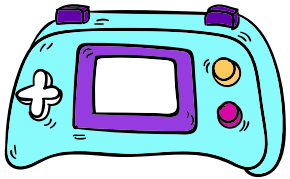


# Patrones de Programación en Videojuegos

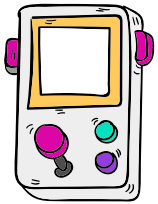


Sin repetir y sin soplar, ¿Qué patrones de programación se les ocurren que se pueden aplicar al desarrollo de videojuegos?

Como todo software, los patrones aplicables son incontables, pero existe un grupo que fundamentalmente aprovecha la naturaleza recursiva e interactiva de los videojuegos, los que llamamos Patrones de Programación de Videojuegos



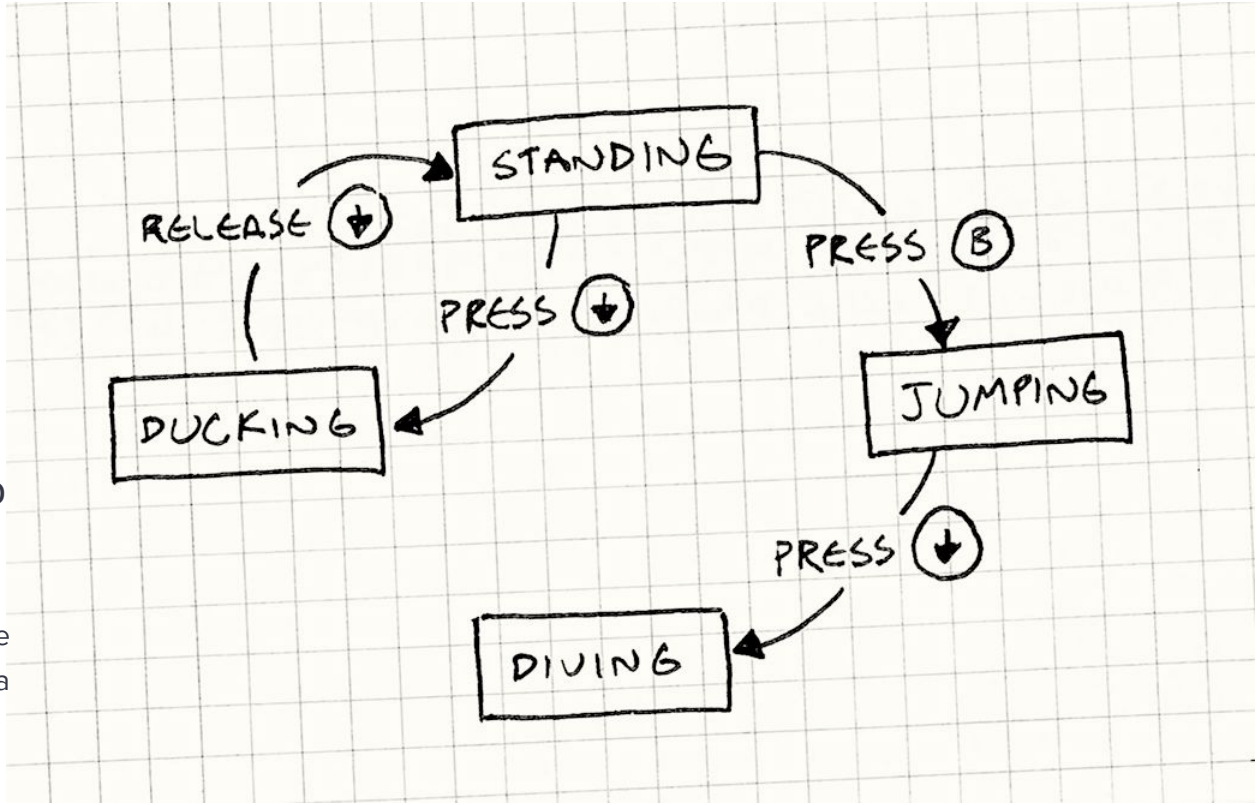
Veamos algunos...

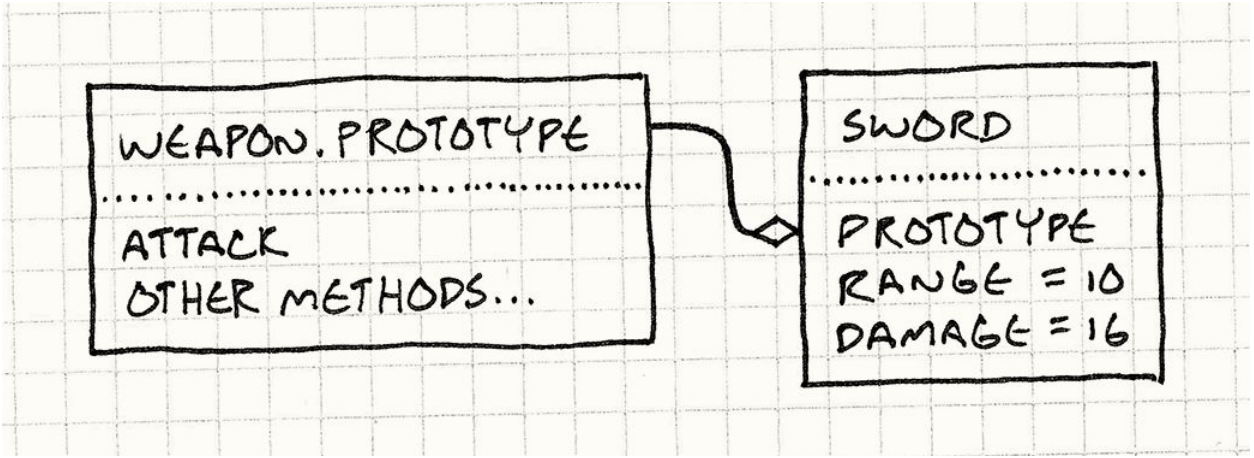


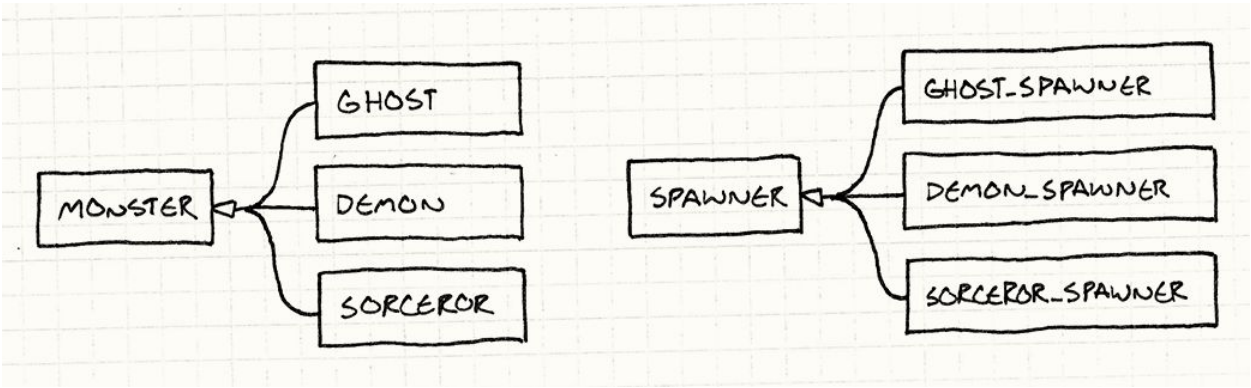
State

Relacionado al concepto de ‘Máquina de Estados Finita’, se utiliza cuando queremos que el comportamiento de un objeto cambie dependiendo del estado interno del mismo.

Uno de los patrones más versátiles, se suele utilizar para player controllers, para IA y para transición entre animaciones.



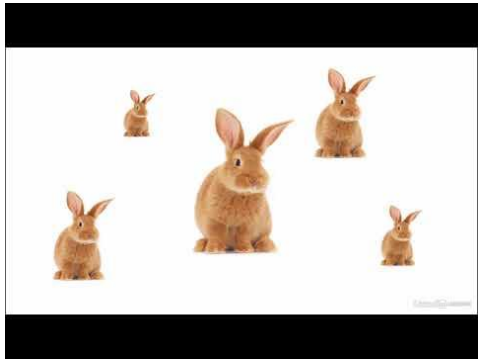



[Prototype]

Es un patrón de diseño creacional que nos permite copiar objetos existentes sin que el código dependa de sus clases.

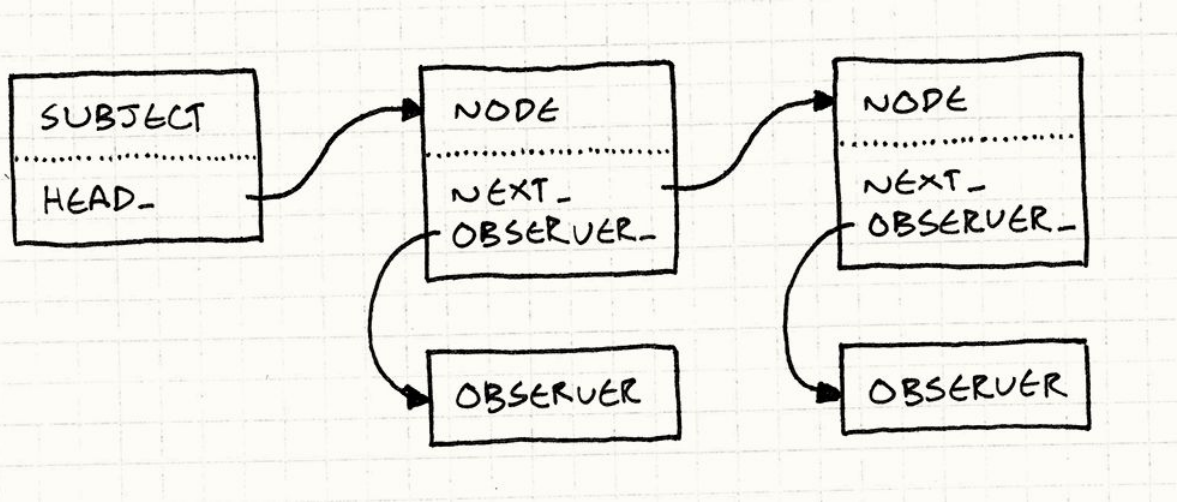
Godot ya lo posee nativo con el método **`Node.duplicate(flags:int)`**

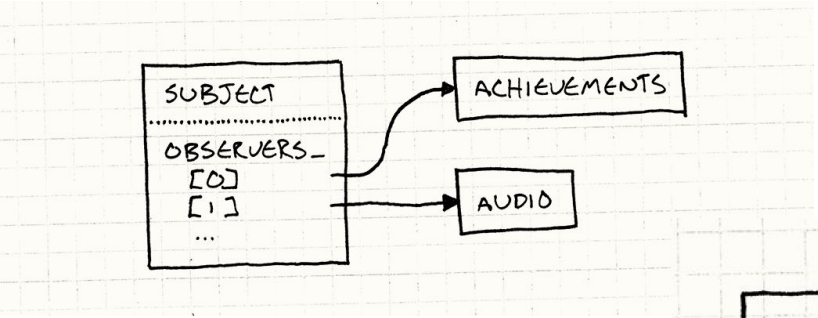


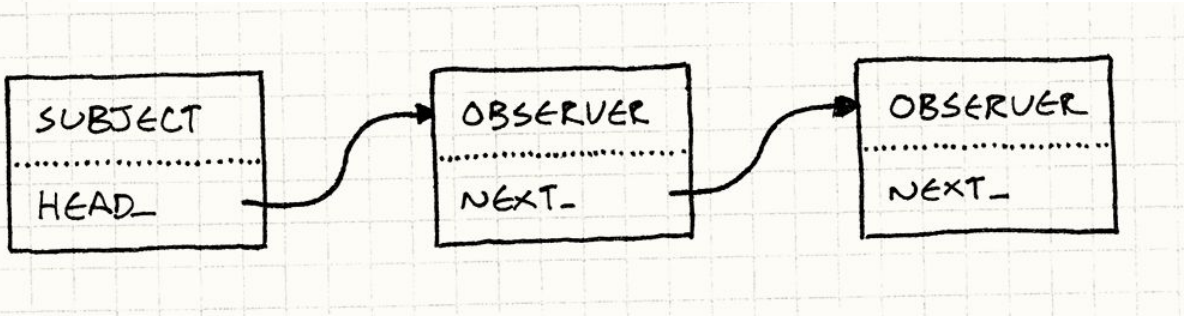
## Observer

Es un patrón de diseño de comportamiento que te permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le suceda al objeto que están observando.

Las signals/señales de Godot son la implementación nativa de este patrón



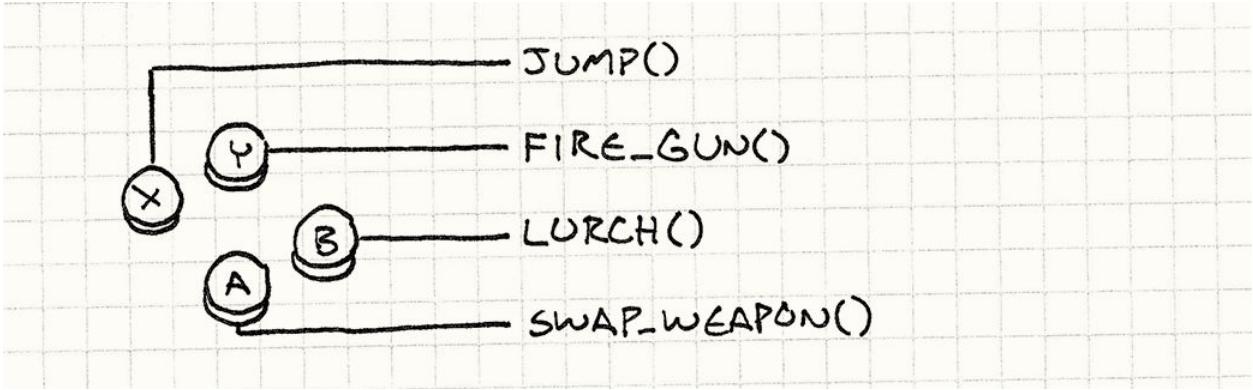



## Command

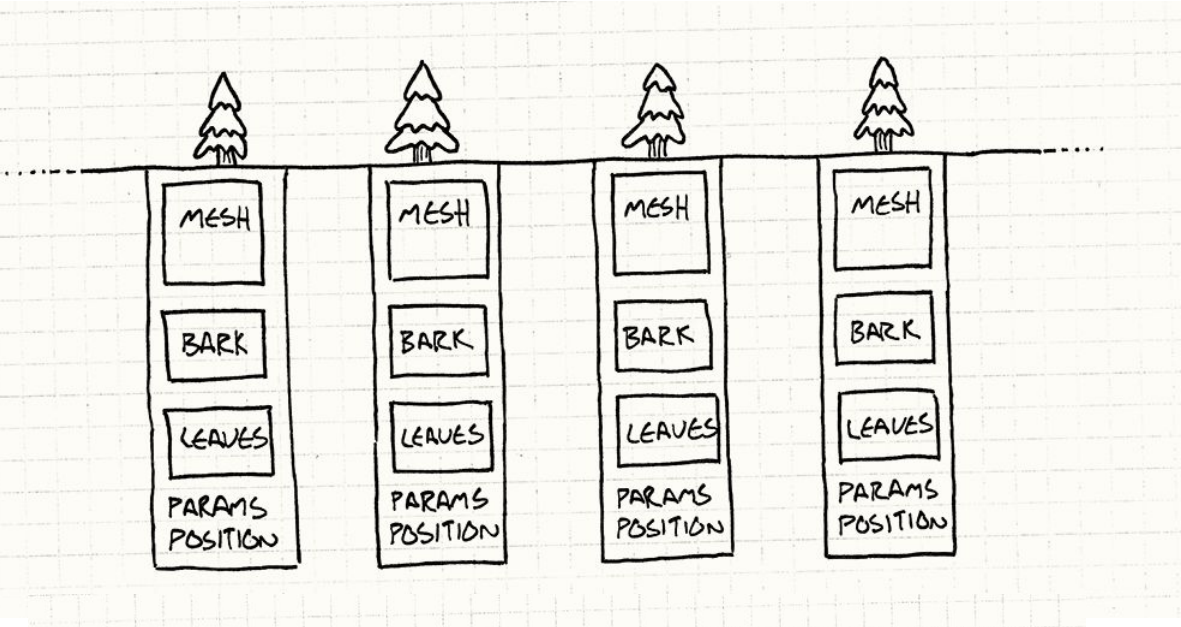
Es un patrón de diseño de comportamiento que convierte una solicitud en un objeto independiente que contiene toda la información sobre la solicitud. Esta transformación te permite parametrizar los métodos con diferentes solicitudes, retrasar o poner en cola la ejecución de una solicitud y soportar operaciones que no se pueden realizar.

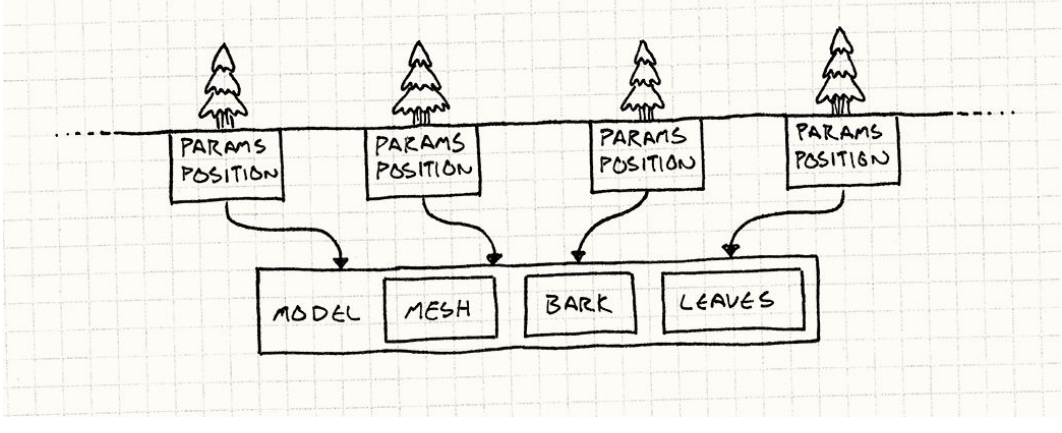
- Fidel dungeon rescue
- Baba is you
- Braid -- más o menos

Prince of Persia: arenas del tiempo





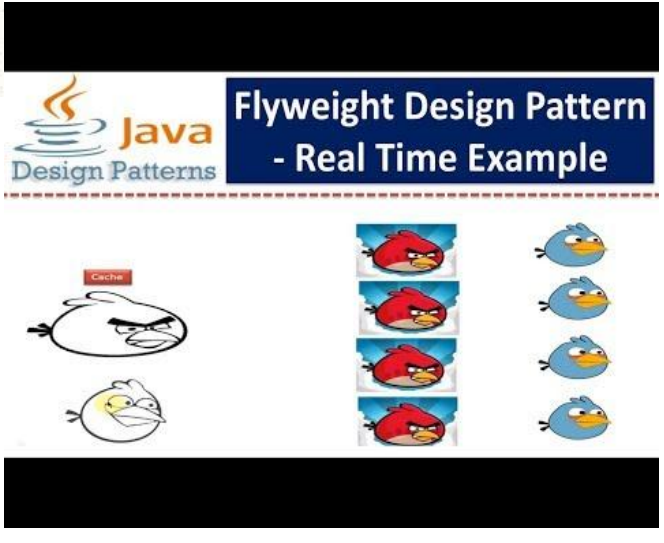



[Flyweight]

Es un patrón de diseño estructural que te permite mantener más objetos dentro de la cantidad disponible de RAM compartiendo las partes comunes del estado entre varios objetos en lugar de mantener toda la información en cada objeto.

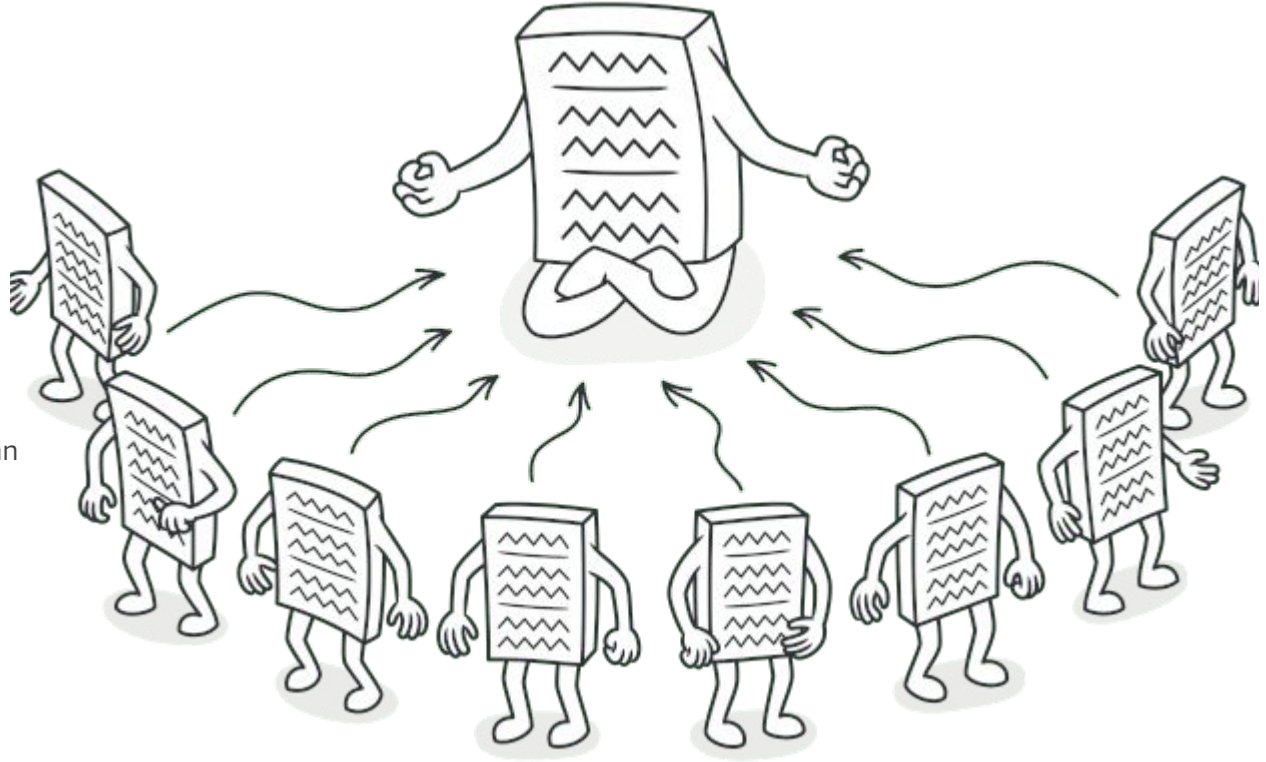
En Godot, los **tilesets** utilizan este patrón de manera nativa



Singleton

Asegurarse de que una clase tenga una única instancia y proporciona un punto global de acceso a ella.

En Godot existen como **Autoloads**, y brindan acceso global automático a la instancia del singleton desde cualquier script



Bibliografía de referencia:

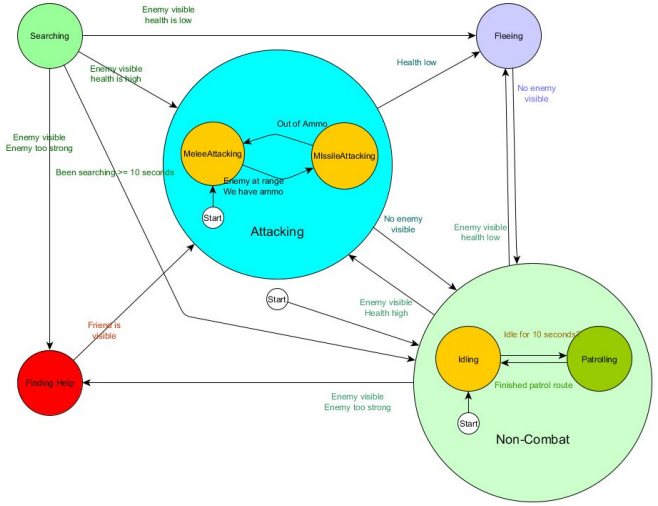
- Libro web 'Game Programming Patterns'
- Más patrones y tutoriales en GDQuest
- Patrones generales en Refactoring Guru

# Inteligencia Artificial en Videojuegos

Cuando hablamos de una “Inteligencia Artificial”, ¿Cuál es la primera idea que se nos viene a la mente?

En videojuegos, buscamos que la “Inteligencia Artificial” sea más “Artificial” que “Inteligencia”, es decir, no buscamos (a menos que sea el propósito del juego) simular una inteligencia, tan solo emularla.

¿Por qué sólo queremos emularla? La potencia computacional que se necesita para “simular” la respuesta de una IA ante un estímulo supera con creces el requisito de poder procesarse a 60 cuadros por segundo, por lo que necesitamos buscar “atajos”





Existen casos más actuales donde se usan redes neuronales para simular un NPC, como en AI Dungeon para textos o los bots de Nvidia, pero, así como la propia tecnología aplicada al rubro, la mayoría de los proyectos son solo pruebas de concepto.

¡Y esto se sigue manteniendo! Un proyecto que use ChatGPT correctamente sería uma delicia

# Componentes habituales de una IA de Videojuegos

Depende del concepto del sistema que implementemos como IA

Generalmente, cuando hablamos de IA nos referimos a los NPC (Non-Playable Character), personajes que habitan el mundo de juego

- Comportamiento del actor
- Toma de decisiones
- Pathfinding (Si pueden desplazarse)

Pero también podría interpretarse como IA a sistemas generales no evidentes reactivos al comportamiento del jugador o del mundo

(Ejemplo: La IA de Director de L4D)

Eso nos lleva a la pregunta, ¿Cómo implementamos una IA?

¿Cómo decidimos qué arquitectura implementar para nuestra IA? Charla de la GDC

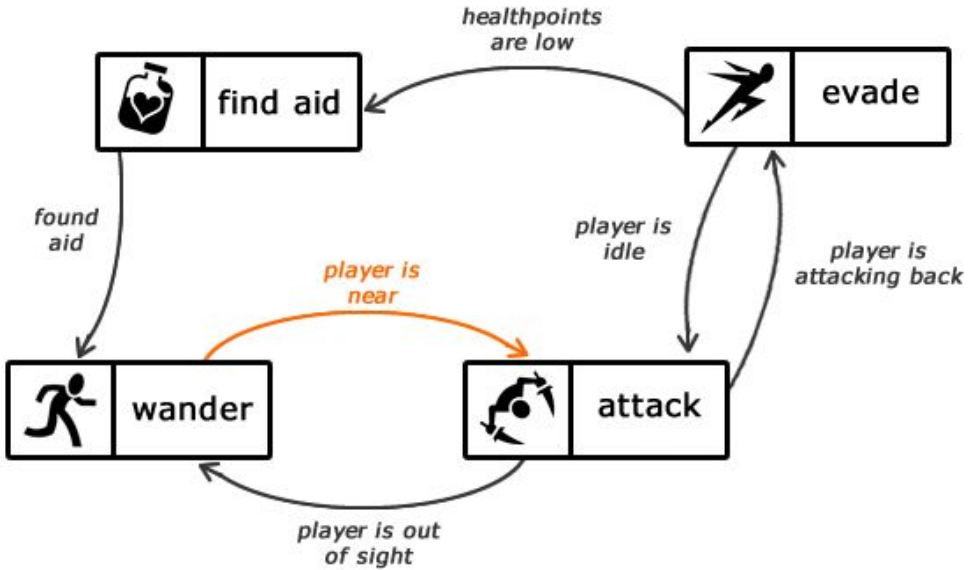
# Comportamiento de una IA

Existen diferentes maneras de programar una IA, algunos de los patrones más comunes son:

## Finite State Machine (Máquina de Estados Finita)

Una **máquina de estados finita** se basa en el modelo de una máquina abstracta que puede encontrarse en exactamente **un estado entre un número finito**, y puede **transicionar** entre ellos de acuerdo a un input determinado.

Una manera fácil de visualizar su comportamiento es con **diagramas de flujo**, donde los nodos son estados y las flechas que los conectan son los inputs que determinan la transición al siguiente estado.




(¿Qué es?, ¿Cómo se aplica a Godot?, ¿Hay más esquemas?)

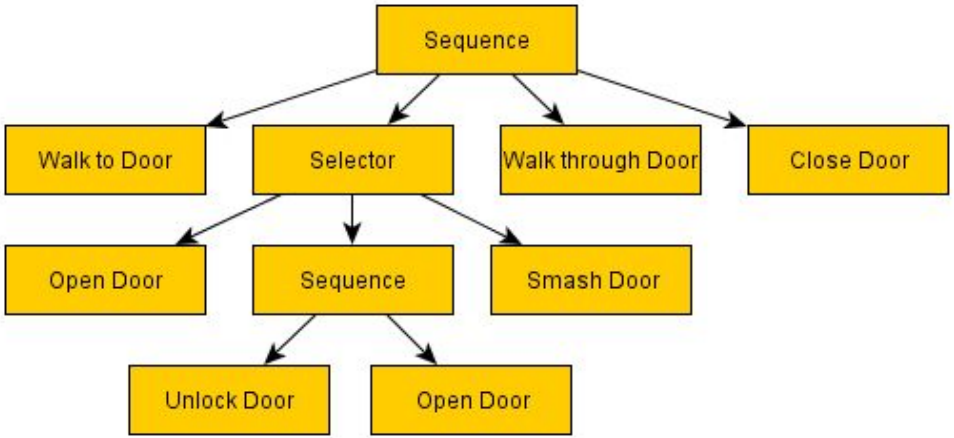
# Behavior Tree (Árbol de Comportamientos)

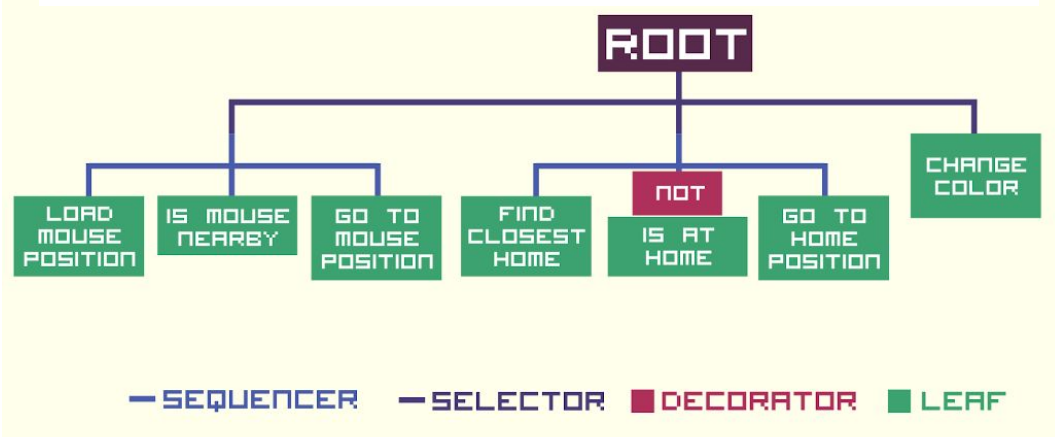
Un **Behavior Tree** es una estructura jerárquica en un **árbol de nodos**, caracterizada por permitir la creación de comportamientos complejos en base a múltiples tareas independientes.

Se compone de nodos de **control de flujo** (*composites* o compuestos) como los *selectors* o selectores y los *sequencers* o secuenciadores, **nodos de ejecución** (*leaves* u hojas) y **nodos decoradores**.

Su fortaleza se encuentra en su **escalabilidad** y **modularidad**, gracias al poder combinar estructuras simples, y, dado que los nodos no poseen estado propio, les brinda **portabilidad** y la posibilidad de crear **librerías de comportamientos.**

(¿Qué es?, ¿Y aplicado a Godot?)



# ¿Puede la IA hacer trampa?

La respuesta es ambigua. La IA no es más que otro sistema en nuestro juego, y como todos en nuestros juegos **hace valer más la experiencia** del mismo que su adhesión a las reglas. Eso significa que si “hace trampa” debe ser con el motivo de complementar el diseño y la experiencia de nuestro juego.


# ¿Qué es el Pathfinding?

Pathfinding es como conocemos a los distintos métodos y algoritmos mediante los cuales se puede determinar el camino más corto entre el punto A y el punto B.

Utilizamos estos algoritmos para proporcionarle a la IA la información para que encuentre el camino a su objetivo, evitando (idealmente) cualquier obstáculo en el camino.

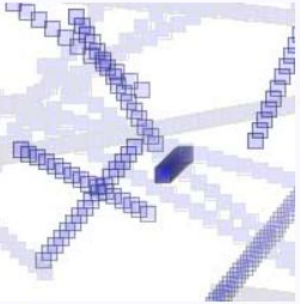
Algunos métodos y algoritmos conocidos:

- Navmesh (Navigation2D en Godot, aplicable también a Tilesets)
- Waypoints (Rutas conectadas como nodos en el espacio)
- A*/AStar (en Godot como wrapper 2D y capaz de implementarse de varias formas)
- Métodos mediante colisiones (Sin datos precomputados)

Navigation2D y AStar en Godot

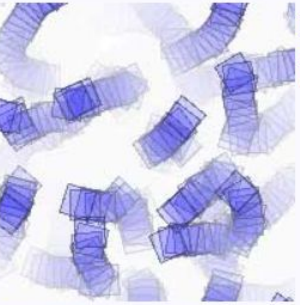

| **Emiliano Causa 2011** | emiliano.causa@gmail.com | www.biopus.com.ar | www.emiliano-causa.com.ar |

## Simulación de Ecosistemas



### Movimiento

Se explican cuestiones básicas de movimiento bidimensional. Así como la generación de agentes autónomos, capaces de desarrollar su propia historia. También se estudia la generación de un espacio toroidal.



### Territorio

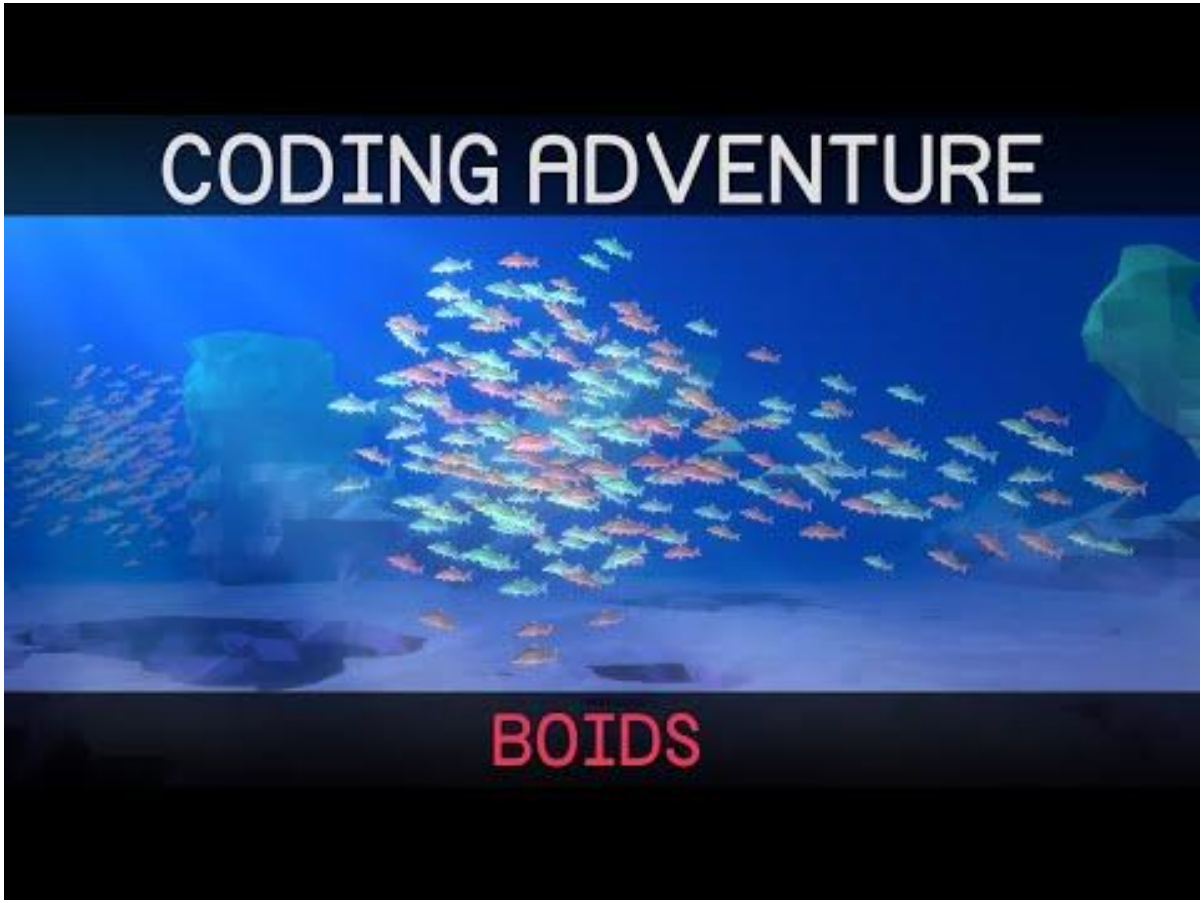
Se aprende a administrar un territorio para resolver el problema de la explosión combinatoria originada en la gran cantidad de interacciones entre individuos.

### Especies y conductas

En este capítulo se explica como crear diferentes especies, las cuales sean capaces de registrar al otro y en función de eso tomar decisiones. También se ve un modelo práctico de como establecer relaciones entre distintas especies.

http://www.emilianocausa.ar/emiliano/vidaartificialarte/

Universidad Nacional de Quilmes




[Paper sobre Boids]

Universidad Nacional de Quilmes

# Desafío #7

- Generar un proyecto en GODOT usando el Template suministrado:
  - Extender la State Machine del Player implementando el estado de “Jump”.
  - Darle la posibilidad al enemigo de moverse por el entorno. Si no posee información de movimiento, no debe moverse.
- Bonus:
  - Implementar comportamiento complejo para el enemigo aplicando patrones de programación de IAs.
  - Implementar nuevo estado “Dash” para la State Machine del Player.

Plantilla de inicio para este desafío
Demo del desafío terminado

Template final del desafío anterior

# TUTORIAL

# Gracias!
# Nos vemos en la próxima