

Intro Desarrollo de Videojuegos 2024 - UNQ v.I

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



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

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

```
1050 REM FOR I=DLSTART TO DLEND
1060 REM PRINT I,PEEK(I)
1070 REM NEXT I
1080 REM
1090 POKE 512,0
1100 POKE 513,6
1110 REM
1120 FOR I=1536 TO 1550
1130 READ A
1140 POKE I,A
1150 NEXT I
1160 REM
1170 FOR I=DLSTART+6 TO DLSTART+28
1180 POKE I,130
1190 NEXT I
1240 POKE 54286,192
2000 REM
2010 DATA 72
2020 DATA 173,11,212,141,10,212,141,24,2
08,141,26,208
2030 DATA 104,64
READY
█
```

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



[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?

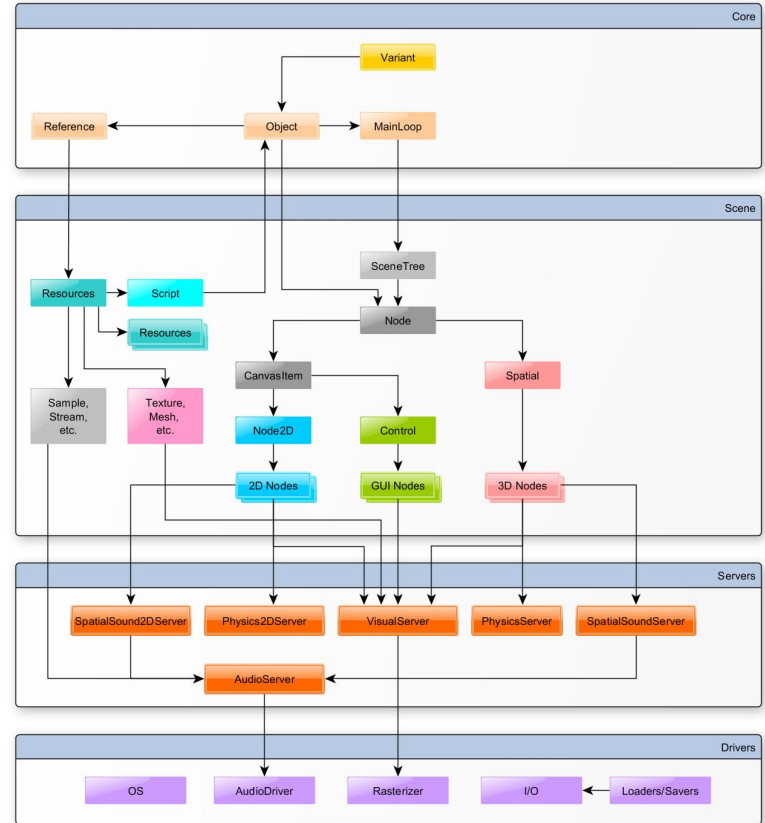
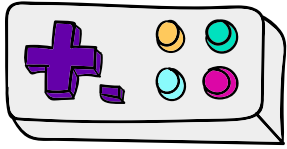
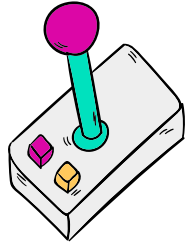


Diagrama de la arquitectura que utiliza Godot

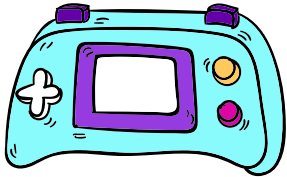


Patrones de Programación en Videojuegos

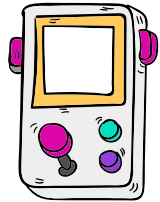


Sin repetir y sin soplar, ¿Qué patrones de programación se les ocurren que se pueden aplicar al desarrollo de videojuegos?

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



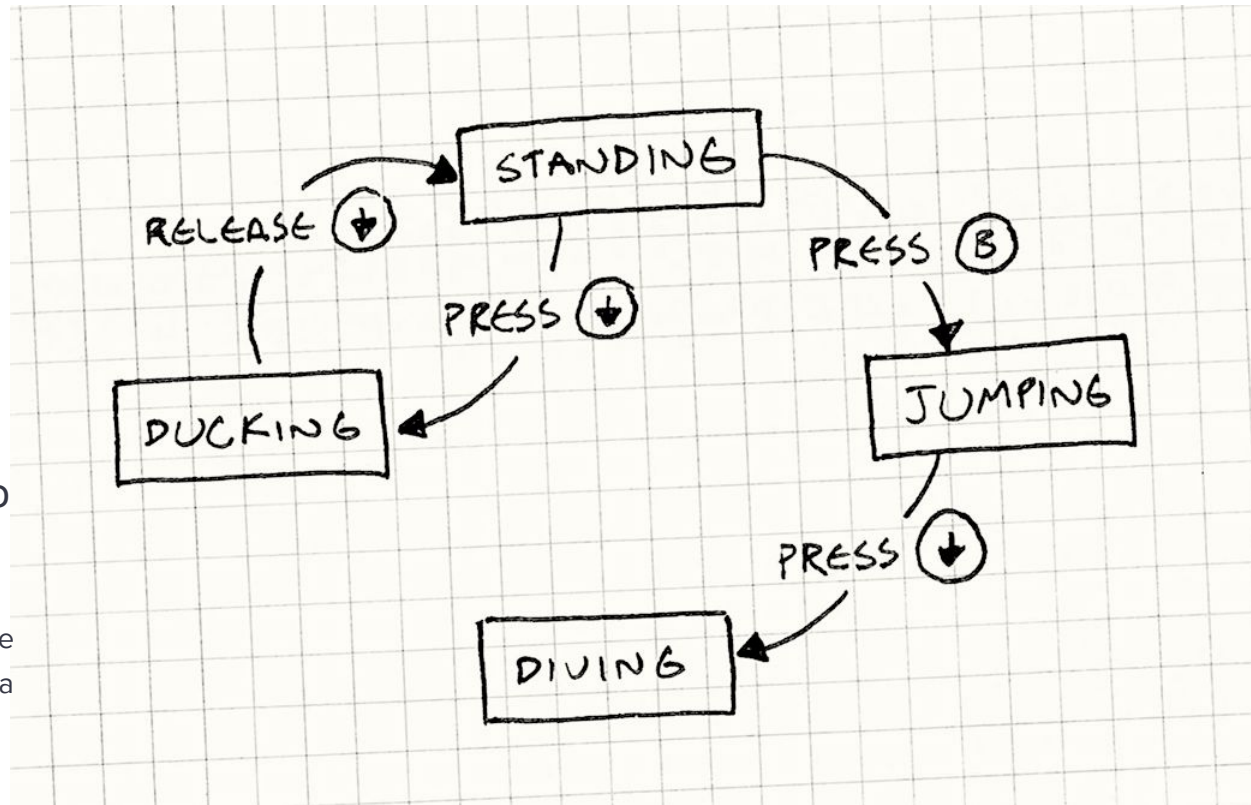
Veamos algunos...

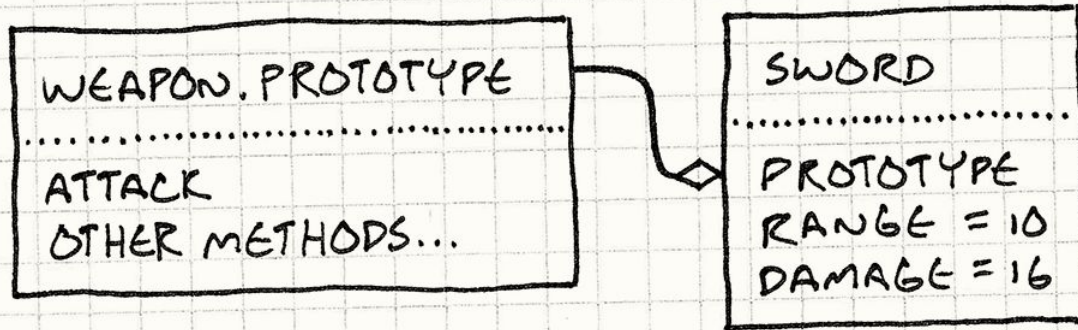


State

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.

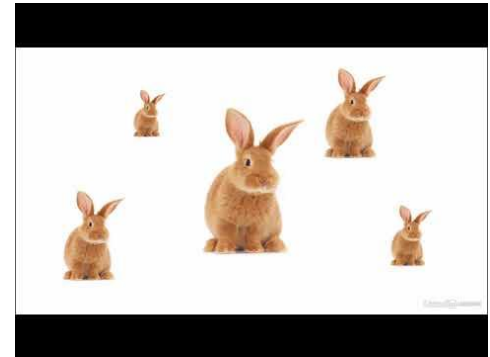
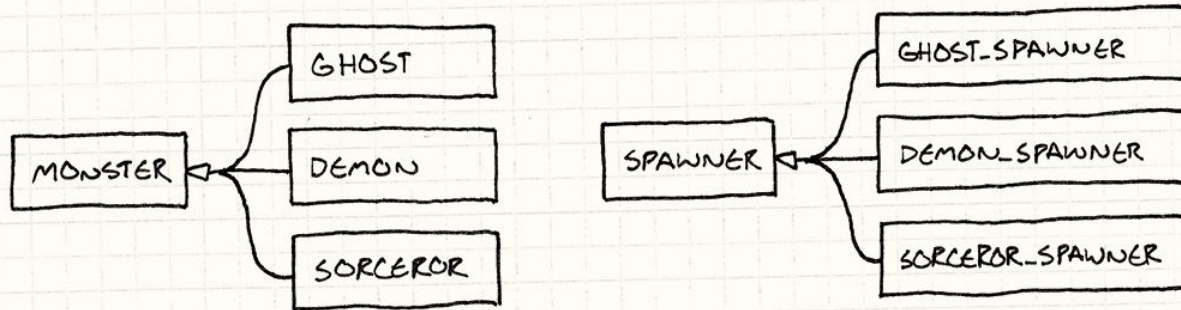




Prototype

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

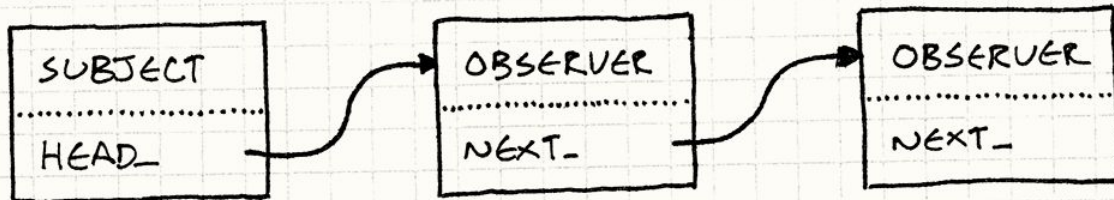
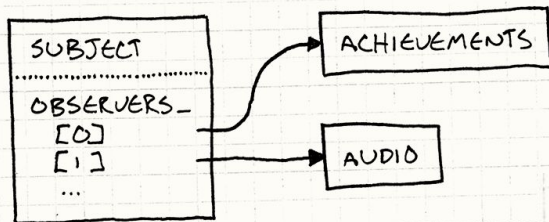
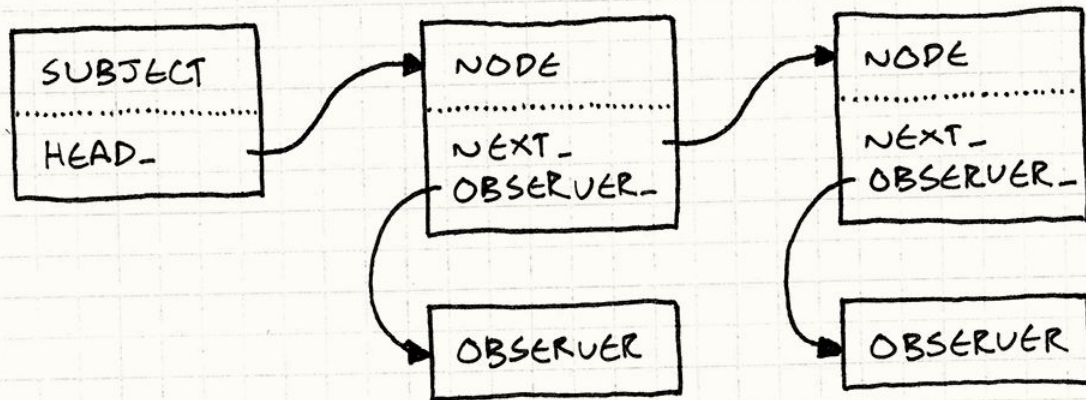
Godot ya lo posee nativo con el método `Node.duplicate(flags:int)`



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

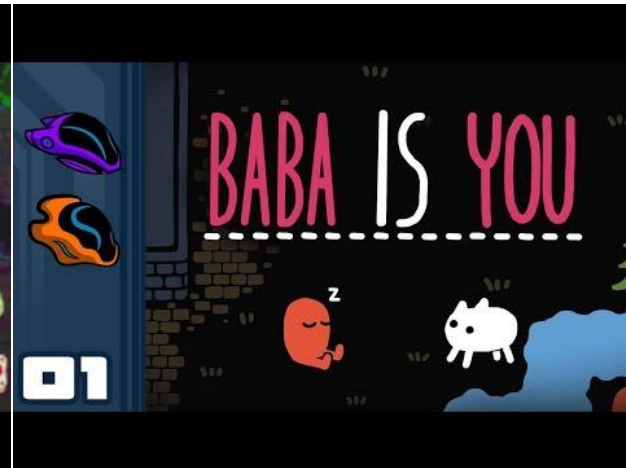
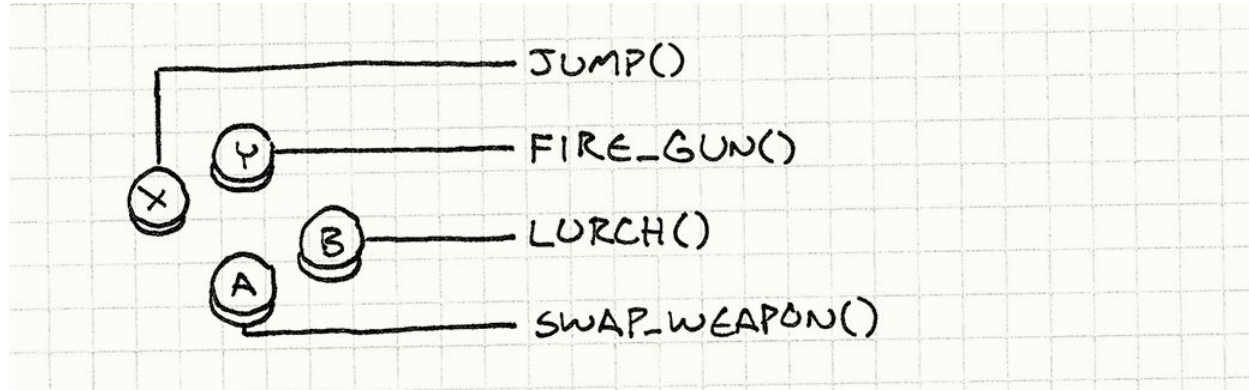


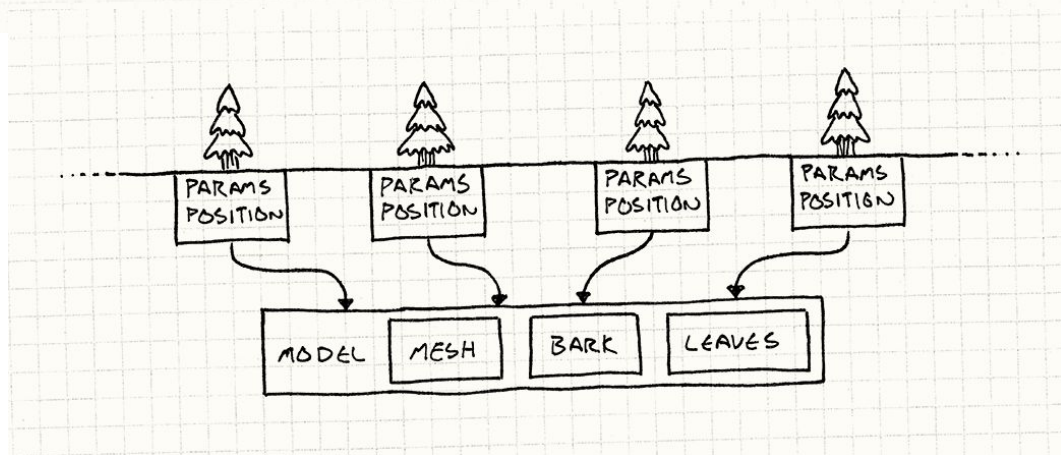
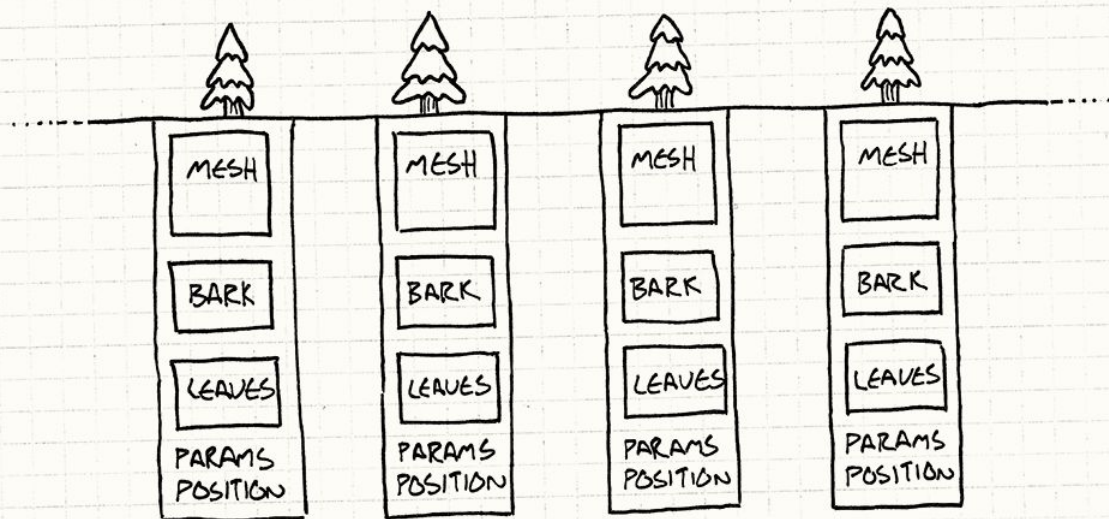
Command

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



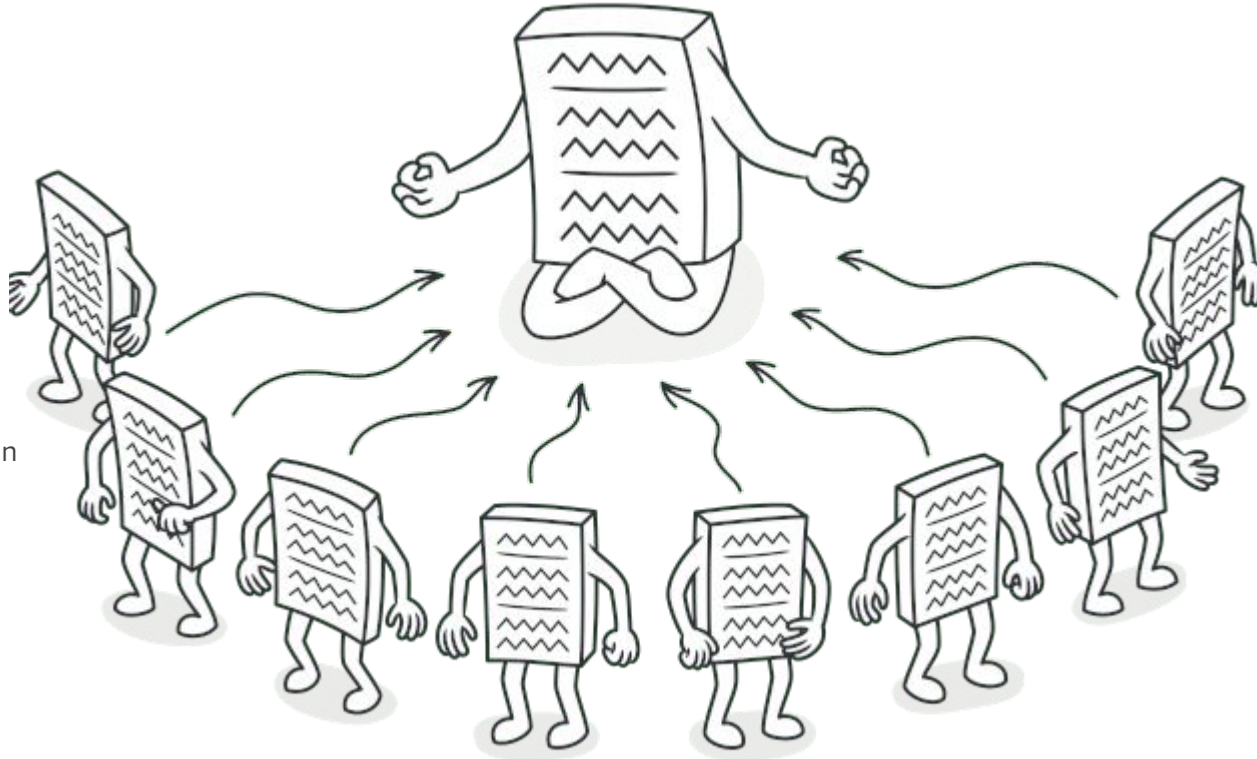
Flyweight Design Pattern - Real Time Example



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:

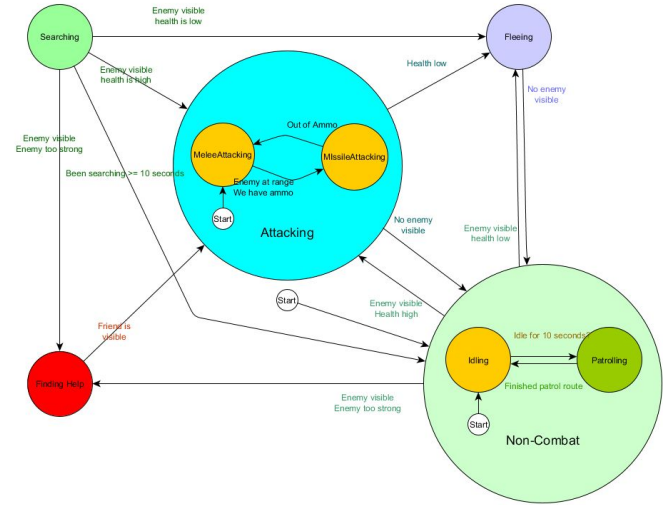
- [Libro web 'Game Programming Patterns'](#)
- [Más patrones y tutoriales en GDQuest](#)
- [Patrones generales en Refactoring Guru](#)

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”



≡ AI Dungeon

You are Nick Walton, a game developer who created an infinite text adventure game AI Dungeon full-time and are raising \$15,000 a month on Patreon. You log onto the post by someone named "The King of Trolls". The King of Trolls is an anonymous online. He's got hundreds if not thousands of followers, some of whom are profess

> *Read the forum post*

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 una 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](#)

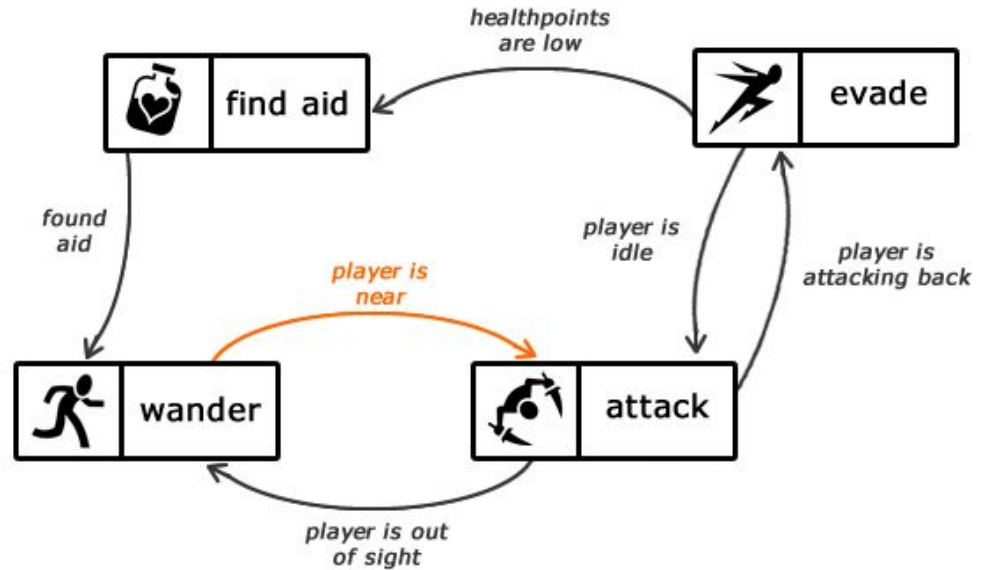
Comportamiento de una IA

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?](#))

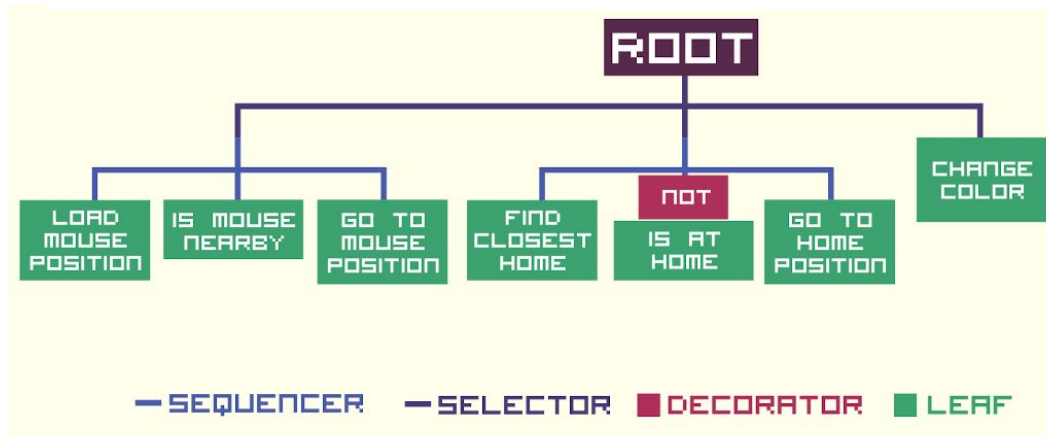
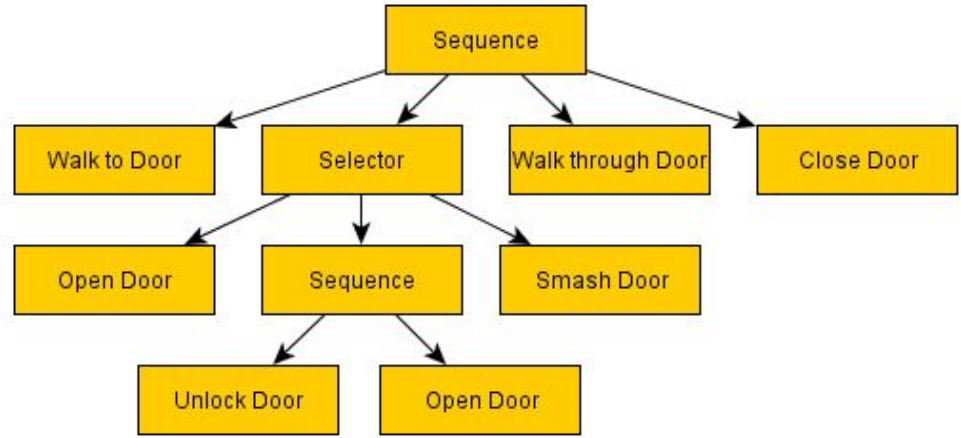
Behavior Tree (Árbol de Comportamientos)

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.

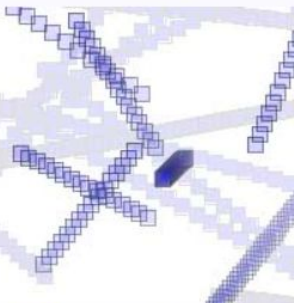
Algunos métodos y algoritmos conocidos:

- [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)

Arte y Vida Artificial

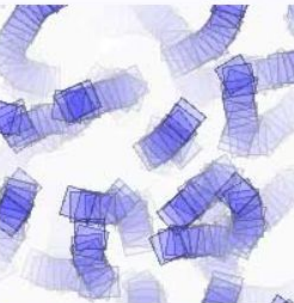
| Emiliano Causa 2011 | emiliano.causa@gmail.com | www.biopus.com.ar | www.emiliano-causa.com.ar |

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

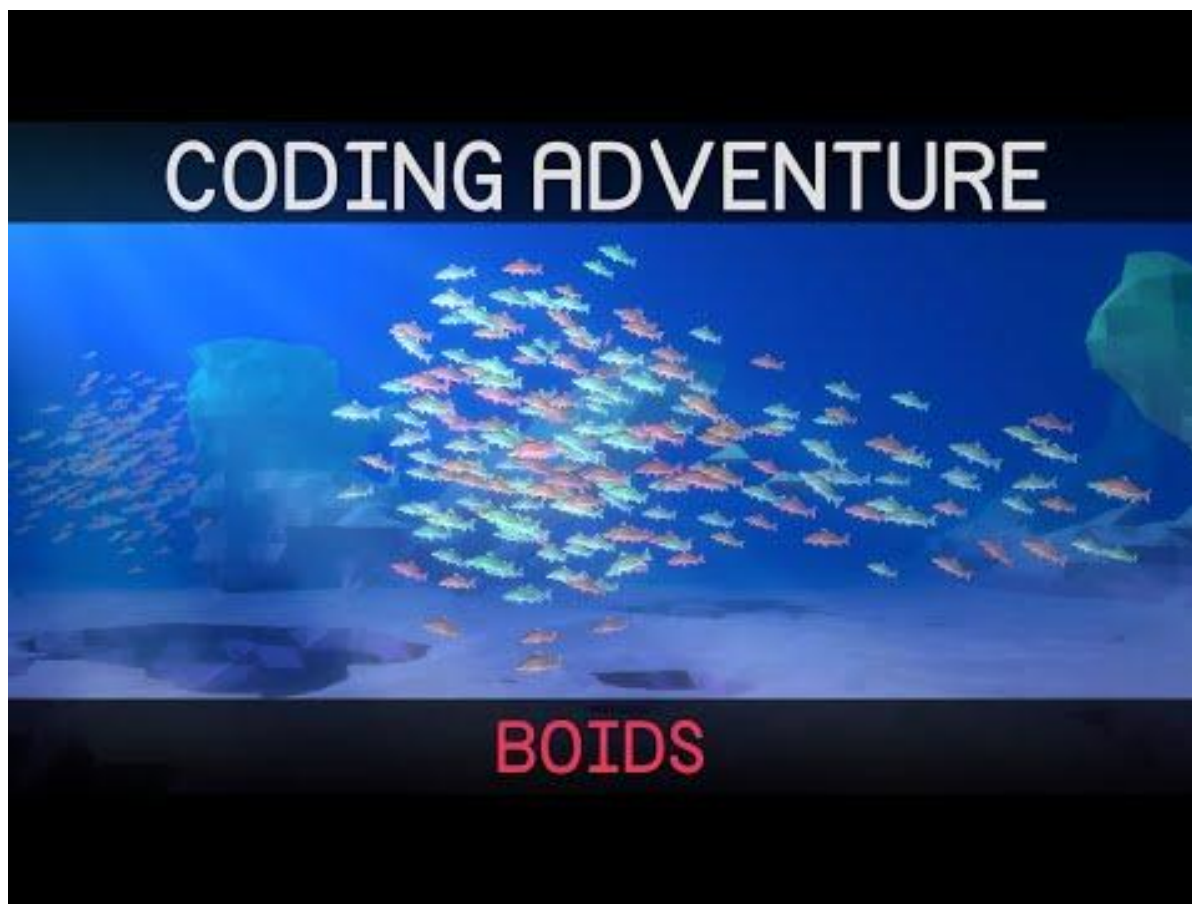
Se aprende a administrar un territorio para resolver el problema de la explosión combinatoria originada en la gran cantidad de interacciones entre individuos.



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/>



[Paper sobre Boids](#)

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