



Inteligencia Artificial para Videojuegos

Navegación
Representación del entorno

Motivación

- El *merodeo* y los *caminos predefinidos* no son suficientes para que un agente pueda moverse **a cualquier punto del entorno**



Representación del entorno

Motivación

- La habilidad de ^{NAVIGATION} **navegar** el entorno, consiste en eso, en conocerlo y poder **buscar** ^{PATHFINDING} **caminos** (= *planificar rutas*)
 - Normalmente el **destino** es una *decisión ya tomada*
- Estos **algoritmos de búsqueda** no trabajan *directamente* con el entorno, necesitan traducir su geometría a un *modelo que lo represente de forma simple*
 - Típicamente se generan **grafos** como este modelo

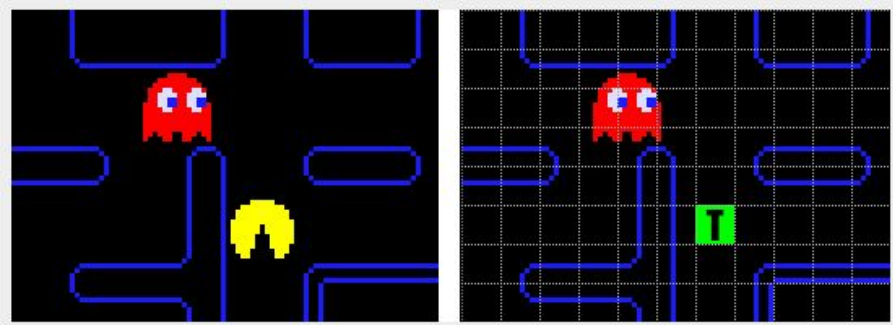


Puntos clave

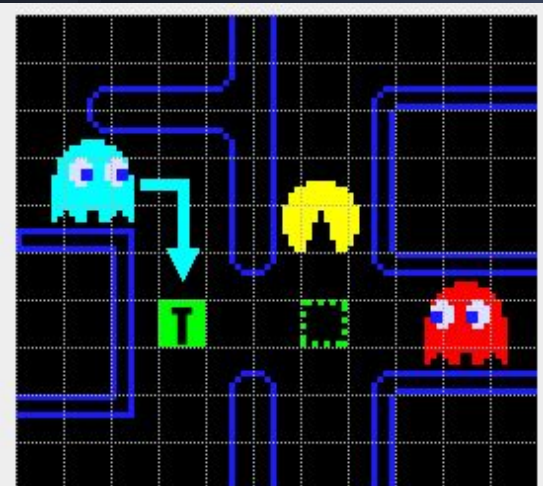
- Hitos históricos
- Grafo de navegación
- Esquemas de división
 - Grafo de baldosas
 - Teselación de Dirichlet
 - Puntos de visibilidad
 - Mallas de navegación
- Entornos inteligentes

Hitos históricos

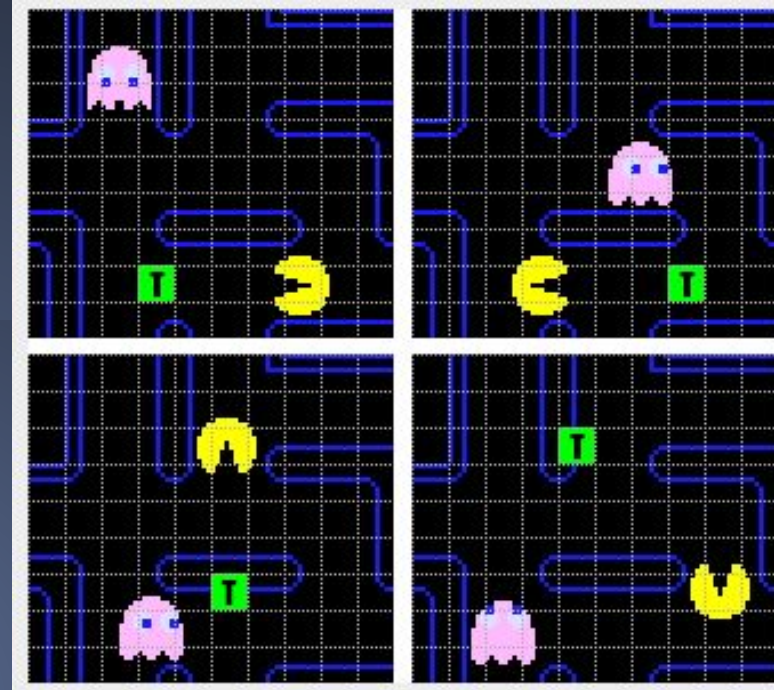
- Búsqueda de caminos en Pac-Man (1980)



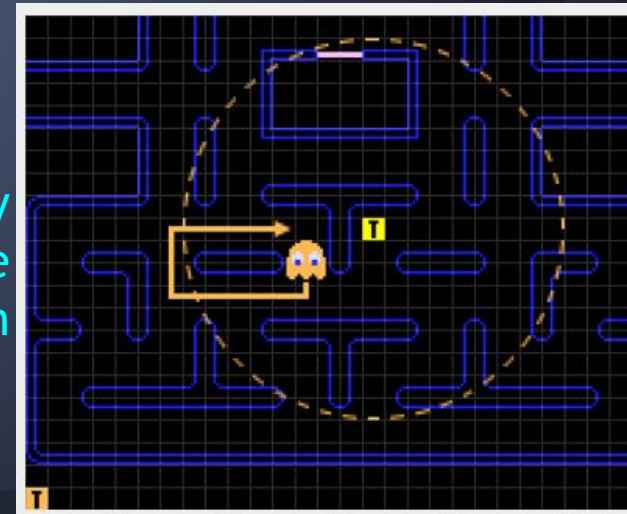
Blinky, seguimiento



Inky, seguimiento a un punto dependiente de Pac-Man y Blinky (con *bug*)



Pinky, seguimiento con pseudopredicción (con *bug*)



Representación del entorno

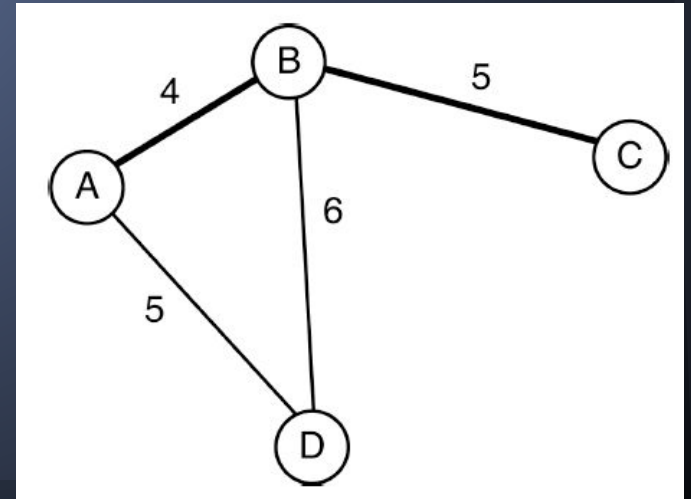
Hitos históricos

- **La Abadía del Crimen (1987)**
 - Búsqueda de caminos *a dos niveles*, entre regiones y dentro de una misma región (tamaño pantalla)
 - Entre regiones se permitían “trampas” para evitar colisiones
- **First Queen (1988):** Primer RPG táctico con **NPCs que te siguen** (*buscan caminos*)
 - A diferencia de *los cruciales enemigos*, *los compañeros* han cobrado importancia hace poco



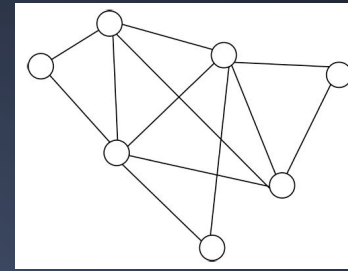
Grafo de navegación

- Es un **grafo ponderado no-negativo** (habitualmente **dirigido**)
 - Traducimos *regiones* (bueno, su punto más representativo) a **nodos**, y vecindad por **conexiones**
 - Un **camino** es una *serie de conexiones* desde un nodo origen a un destino
 - El **coste** de un camino es la *suma* de los costes de las conexiones que lo forman
 - Ej. Coste de A-B + B-C
 $= 4 + 5 = 9$

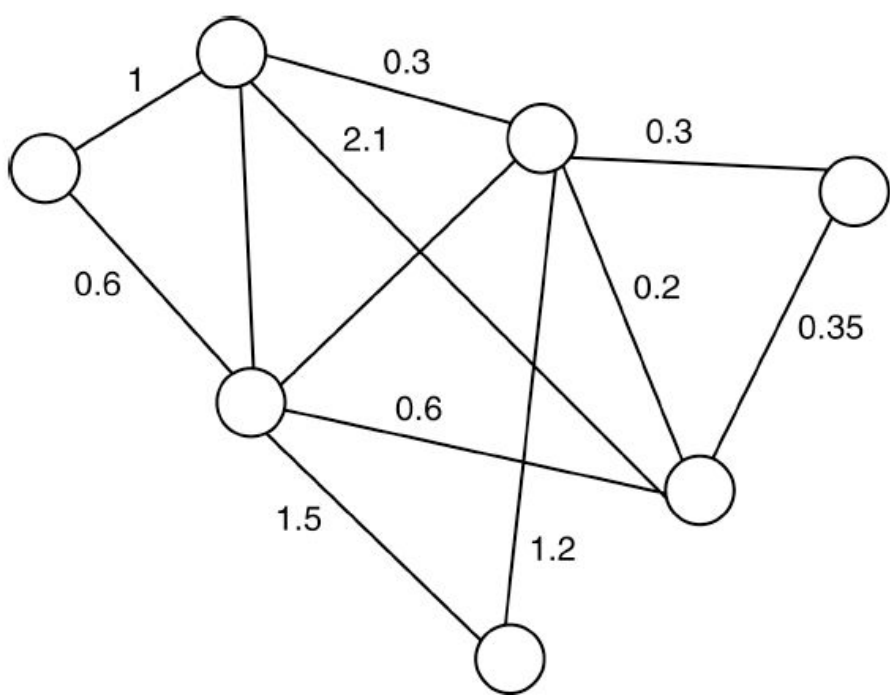


Representación del entorno

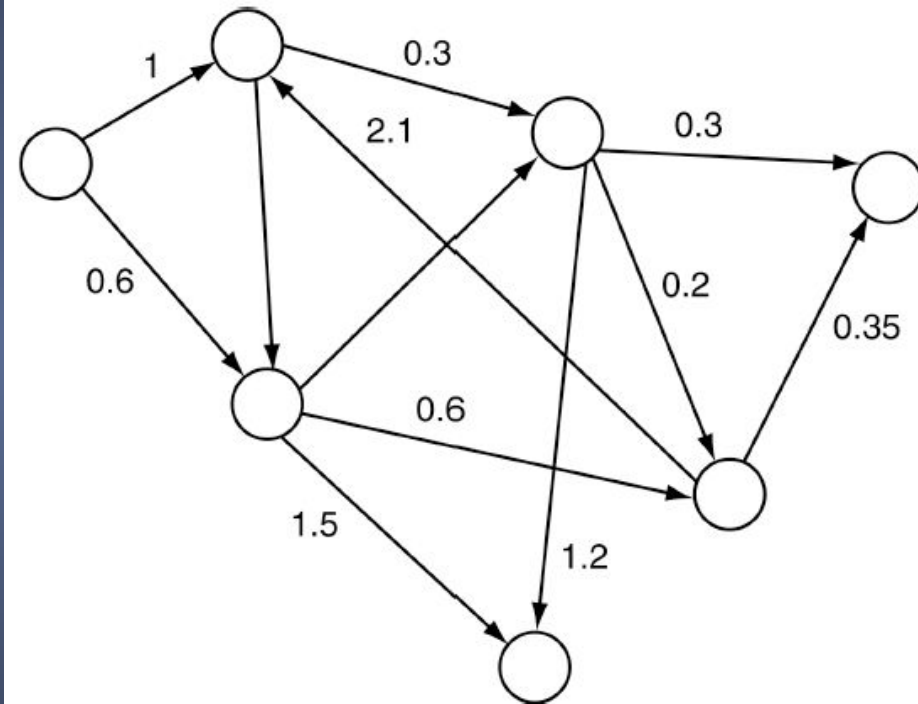
Grafo de navegación



Grafo general



Grafo ponderado (con pesos) *no negativo* (típicamente son *costes* como tiempo o distancia)



Grafo ponderado *no negativo* dirigido (la conexión A-B no implica que exista B-A, ni que tengan ambas el mismo coste)

Grafo de navegación

- Pseudocódigo de la representación

```
1 class Graph:
2     # An array of connections outgoing from the given node.
3     function getConnections(fromNode: Node) -> Connection[]
4
5 class Connection:
6     # The node that this connection came from.
7     fromNode: Node
8
9     # The node that this connection leads to.
10    toNode: Node
11
12    # The non-negative cost of this connection.
13    function getCost() -> float
```

* En IA clásica suele haber clase Nodo, con referencias tan sólo a su padre; en videojuegos los nodos suelen ser simplemente un entero

Participación

tiny.cc/IAV

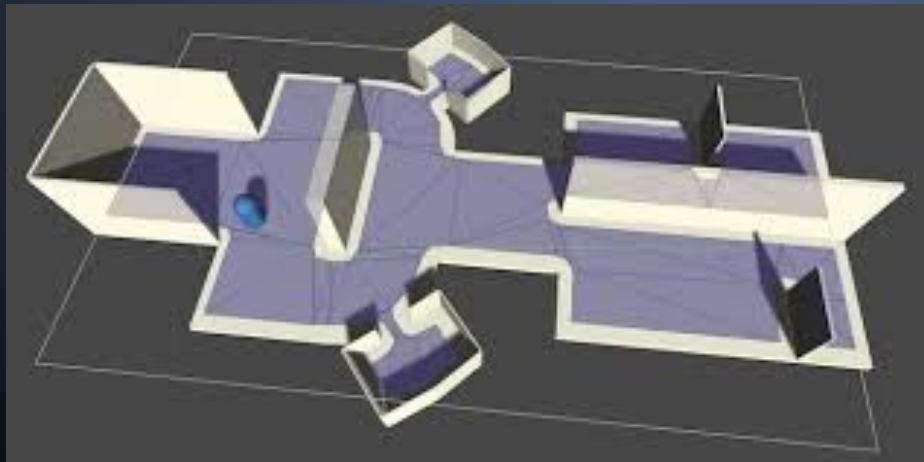
- ¿En qué **situación** *coste $F-G > \text{coste } G-F$* ?
 - A. Si F es piscina y G bordillo, donde la escalerilla
 - B. Si F es alcantarilla y G otra alcantarilla
 - C. Si F es trampolín alto y G piscina
 - D. Si F es callejón y G montaña alta
- Desarrolla tu **respuesta** (en texto libre)



Esquemas de división

- Lo que representa un nodo en el grafo de navegación del videojuego no es toda la región sino *su punto más representativo*
 - ¿Cómo podemos hallar ese punto?
- Los *esquemas de división* son *maneras de dividir el entorno* y hallar esos puntos

DIVISION
SCHEME



Representación del entorno

* Un nivel pequeño de un RTS puede tener
cientos de miles de nodos

Esquemas de división

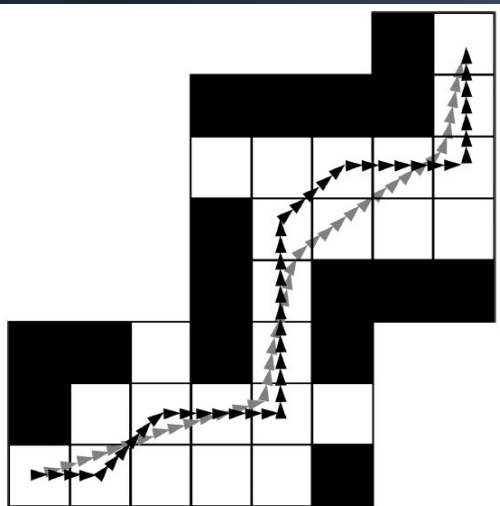
- Antes de *buscar caminos* debemos **traducir el entorno (espacio 3D) a un simple grafo**
 - Puede hacerse según varios *esquemas de división*, cada uno con sus propiedades
 - ↕ ■ **Cuantificación y localización**, cómo traduce de *posiciones del espacio a nodos* y viceversa
 - **Generación**, cómo divide el espacio en regiones (**manual**, **semiautomática** o **automáticamente**)
 - ✓ ■ **Validez**, cómo garantiza que *siempre* se llega de un punto a otro en regiones conectadas

Esquemas de división

● Grafo de baldosas

TILES
GRAPH

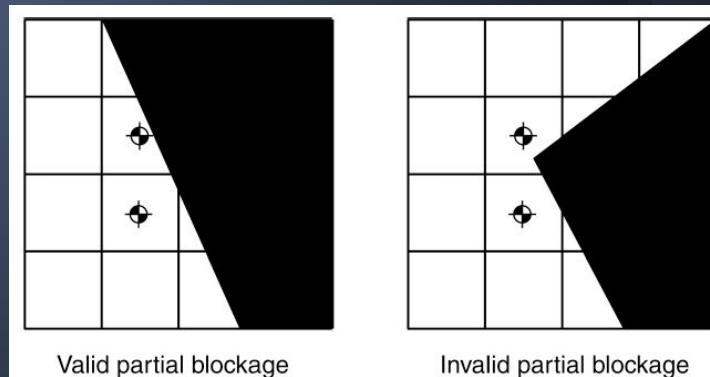
- El entorno puede ser en 3D, pero su estructura *no*
 - ➡ ■ Relación directa con las coordenadas
 - Automática y hasta en tiempo real (es fácil)
 - ✓ ■ No suele permitirse *bloqueo parcial* de baldosas (salvo que esté muy claro)



Key

- ➡ Output blocky plan
- ➡ Ideal direct plan

```
1 tileX: int = floor(x / tileSize)
2 tileZ: int = floor(z / tileSize)
```



Valid partial blockage

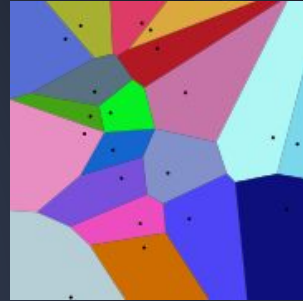
Invalid partial blockage

Esquemas de división

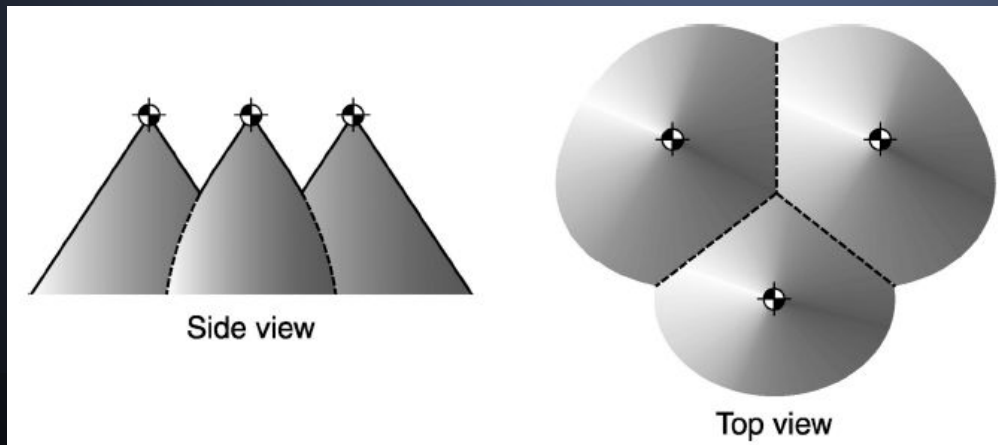
● Teselación de Dirichlet * También dominios, diagramas, polígonos... de Thiessen o Voronoi

DIRICHLET ○
TESSELLATION

Partición 2D en regiones de puntos que comparten un mismo **punto característico** más cercano



- ↻ ■ Directamente con los *puntos característicos*
- ■ **Triangulación de Delaunay**... o manualmente 🙄
- ✓ ■ Todo revisado empíricamente, no queda otra



Esquemas de división

● Puntos de visibilidad

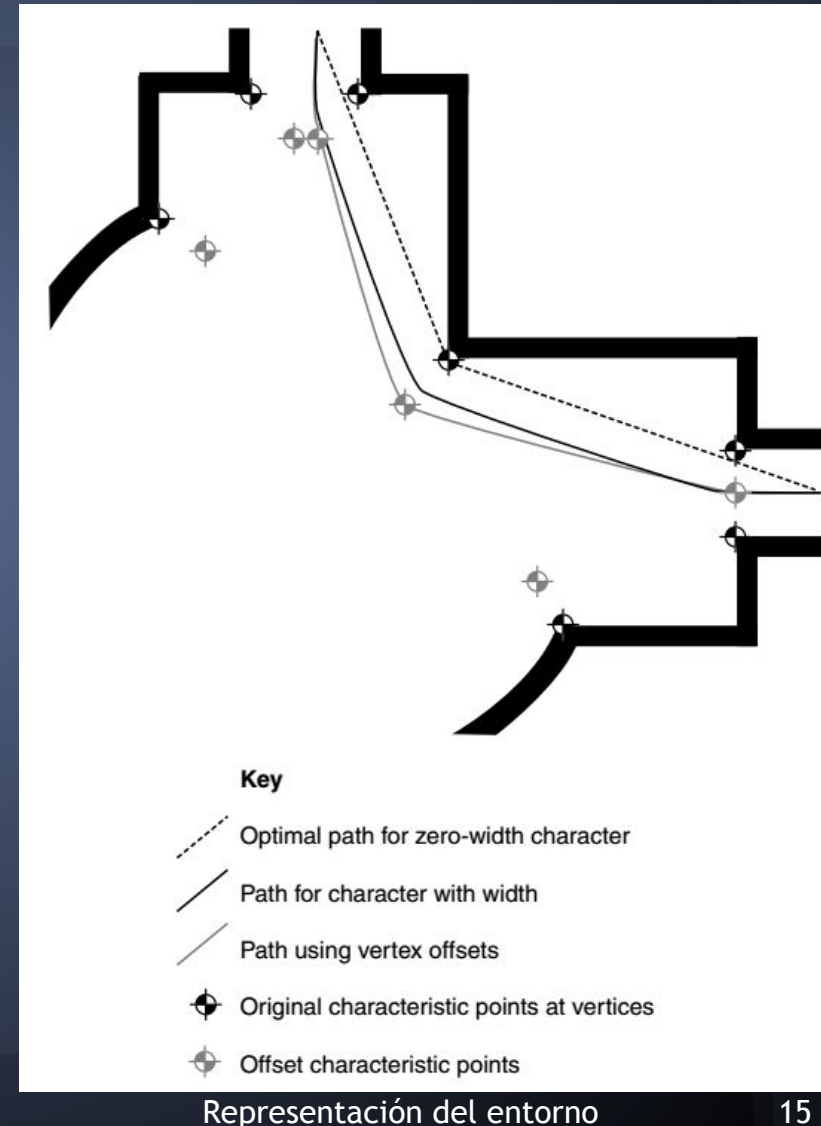
POINTS OF
VISIBILITY○

Pon puntos característicos cerca de las esquinas, ya que el camino óptimo siempre pasa por ellas

↻ ■ *Teselación de Dirichlet* sobre estos puntos

■ Dos puntos conectan si se “ven” entre ellos

✓ ■ También toca revisarlo empíricamente todo



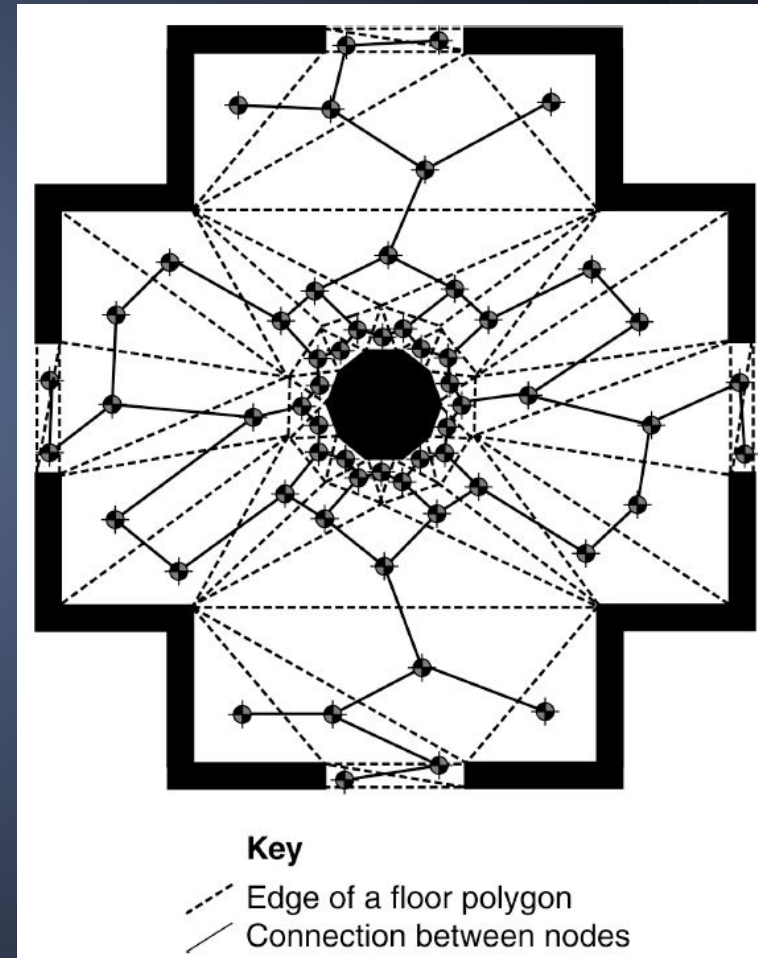
Esquemas de división

● Mallas de navegación

NAVIGATION
MESHES
/NAVMESH

- Se construyen a partir de la geometría del entorno y es el *esquema de división* más popular hoy día

- ↔ ■ Cada polígono (**triángulo**) del suelo es un nodo
- ■ Los polígonos-nodos conectan si comparten lado
- ✓ ■ Los polígonos del suelo deberían ser revisados



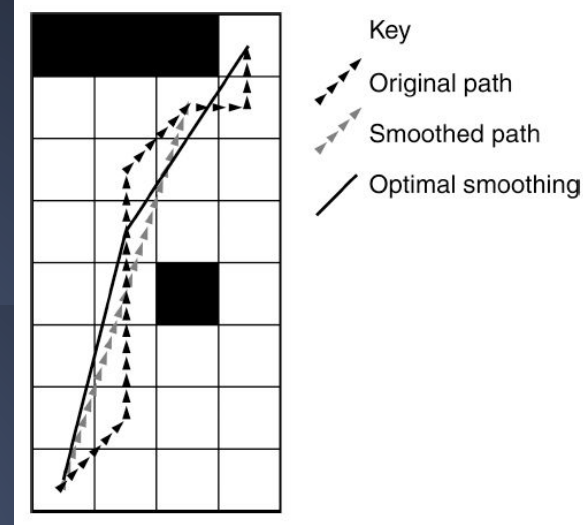
Suavizar el camino

SMOOTHING PATHS

- Algoritmo que **suaviza caminos** quitando algunos de sus puntos
 - Ayuda usar *comportamientos de dirección*
- Pseudocódigo

```
1 function smoothPath(inputPath: Vector[]) -> Vector[]:  
2     # If the path is only two nodes long, then we can't smooth it, so  
3     # return.  
4     if len(inputPath) == 2:  
5         return inputPath
```

- * Recibe todas las posiciones por las que debería moverme (ej. centros de las baldosas del camino) y devuelve sólo aquellas que necesito seguir para no parecer un robot



Suavizar el camino

```
7 # Compile an output path.
8 outputPath = [inputPath[0]]
9
10 # Keep track of where we are in the input path. We start at 2,
11 # because we assume two adjacent nodes will pass the ray cast.
12 inputIndex: int = 2
13
14 # Loop until we find the last item in the input.
15 while inputIndex < len(inputPath) - 1:
16     # Do the ray cast.
17     fromPt = outputPath[len(outputPath) - 1]
18     toPt = inputPath[inputIndex]
19     if not rayClear(fromPt, toPt):
20         # The ray cast failed, add the last node that passed to
21         # the output list.
22         outputPath += inputPath[inputIndex - 1]
23
24     # Consider the next node.
25     inputIndex += 1
26
27 # We've reached the end of the input path, add the end node to the
28 # output and return it.
29 outputPath += inputPath[len(inputPath) - 1]
30
31 return outputPath
```

Tiempo = $O(n)$









Espacio = $O(1)$

donde n es el número de nodos del camino

Entornos inteligentes

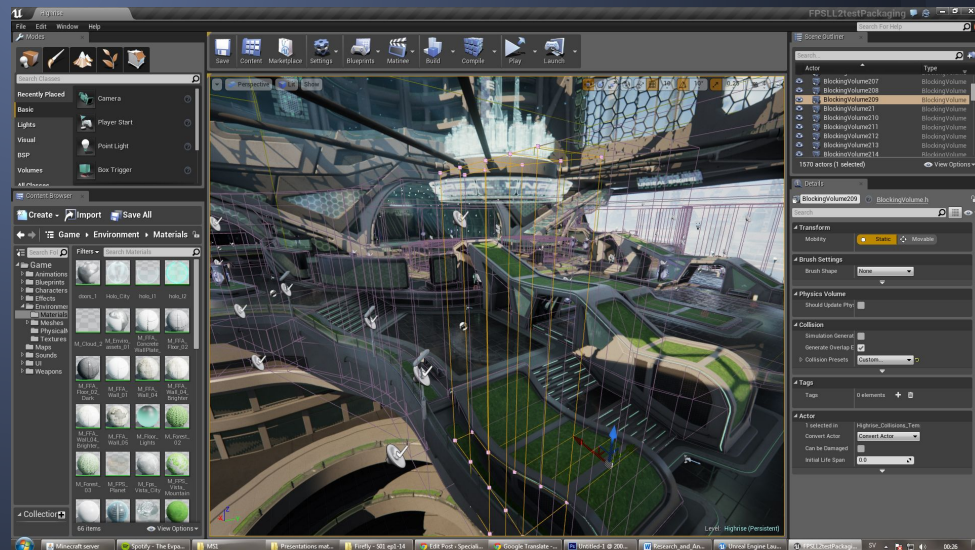
SMART TERRAIN

- La filosofía de los “entornos inteligentes” es que sean las regiones del entorno las que influyan sobre el NPC
 - Buscará un camino estableciendo como *destino* la región de mejor ratio **utilidad/distancia**

Hunger	Refrigerator, Microwave, Cookie Jar			
Energy	Bed, Sofa, Chair			
Fun	Computer, Bookshelf			

Entornos inteligentes

- Todo lo extra al hecho de moverse, **marcar puntos de salto, zonas de aterrizaje**, etc. y en general la mayor parte del mercado del entorno se podría hacer automáticamente
 - Aprovechando el propio **algoritmo de generación del esquema de división** que usemos (para luego *buscar caminos* allí)



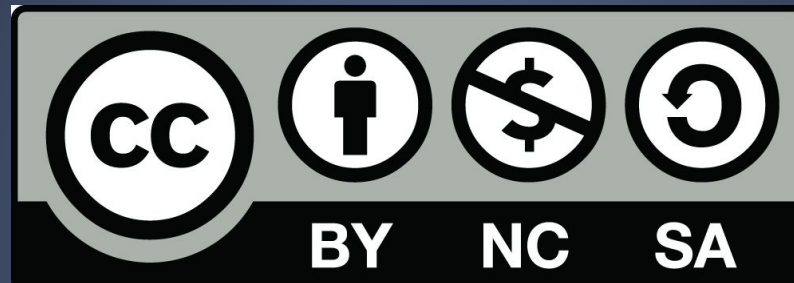
Resumen

- Como hitos históricos tenemos a Pac-Man y a títulos patrios como La Abadía del Crimen
- El grafo de navegación es el modelo utilizado para representar el entorno
- Los esquemas de división más conocidos son el grafo de baldosas, la teselación de Dirichlet, los puntos de visibilidad, y las mallas de navegación (muy populares hoy)
- En los entornos inteligentes son las regiones las que influyen en los objetivos del NPC

Más información

- Millington, I.: Artificial Intelligence for Games. CRC Press, 3rd Edition (2019)
- Pittman, J.: The Pac-Man Dossier (2009)
https://www.gamasutra.com/view/feature/132330/the_pacman_dossier.php

Críticas, dudas, sugerencias...



* Excepto el contenido multimedia de terceros autores

Federico Peinado (2019-2020)

www.federicopeinado.es

