Inteligència Artificial en els videojocs

Professors de VJ

Introducció

En el joc podem distingir entre:

- Personatges jugadors (Players).
- Personatges no jugadors (NPCs – Non-Player Characters).

Tots els NPCs necessiten algun mecanisme per determinar el seu comportament.

És important perque l'èxit dels jocs multijugador demostra la necessitat humana de:

- Oponents inteligents.
- Experiència més rica i diversa.

Introducció

Com en altres apartats hi ha la distinció:

- Recerca en IA – Comportament inteligent.
- IA en jocs – Aparença de inteligència.

Aquesta diferència es menys clara del que sembla:

- Si una IA sembla inteligent ho és realment?
- Què vol dir inteligència?

Introducció

Recerca en IA

- Minimax, αβ → Deep Blue
- Cerques de Montecarlo
- Xarxes neuronals
- Xarxes Bayesianes → Watson
- ...

IA per videojocs

- Path-finding
- Sistemes basats en regles
- Màquines d'estats finits (FSM)

Introducció

Perque no fer servir tècniques de recerca:

- Cost (desenvolupament, execució i manteniment).
- Les IAs dels jocs poden fer trampes:
  - Coneixer tot l'espai del joc.
  - Tenir habilitats superiors a les del jugador.
  - Resultats favorables en events aleatoris.
- Alta complexitat dels jocs:
  - Mon no discret, possiblement obert.
  - Múltiples jugadors amb moviment simultani.
  - Regles (game mechanics) complexes.
  - ...

Pathfinding

És el conjunt de tècniques més basic de la IA per jocs.

- Els NPCs necessiten poder arribar als seus objectius.

Si tenim un punt d'origen A = (Ax, Ay) i volem arribar al punt de destí B = (Bx, By) podem:

- Anar incrementant la posició en:

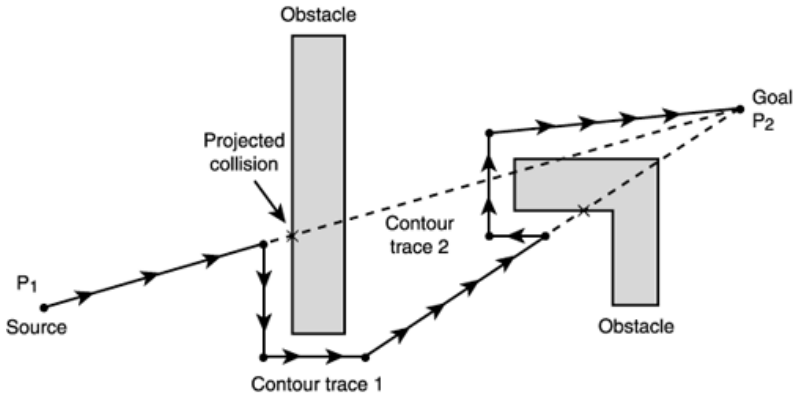
  $\lambda \cdot (Bx-Ax, By-Ay)$ on λ depèn de la velocitat

- Però estem presuposant que no hi ha obstacles.

Pathfinding

Alternativa → Traçat de contorns

- Moure's cap l'objectiu en línia recta
- Si col·lisionem amb un obstacle:
  - Anem seguint el contorn de l'obstacle.
    - Mantenint una certa distància.
  - Comprovar cada cert temps si la trajectòria recta a l'objectiu intersecta l'obstable:
    - No → Tornem a dirigir-nos cap el destí en línia recta.
    - Sí → Continuen “traçant” el contorn de l'obstacle.



Pathfinding

Per a poder cercar camins mínims primer ens cal representar el mon com un graf on:

- Nodes → Punts 2D o 3D en el mon.
- Arestes → Veïnatge entre nodes.
- Les arestes porten distàncies associades.
  - Això pot ser útil per representar el cost associat al moviment.

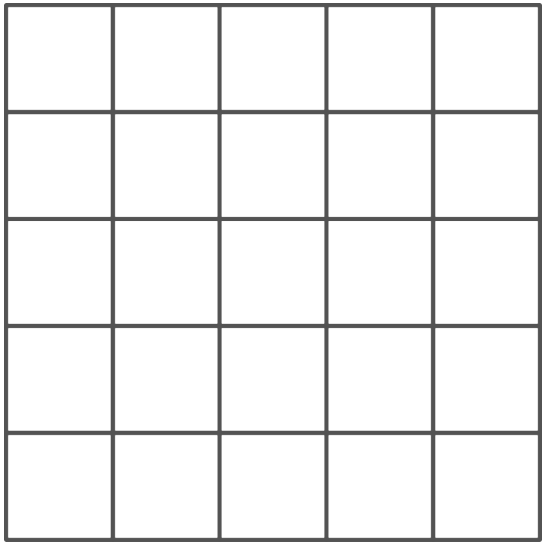
Pathfinding

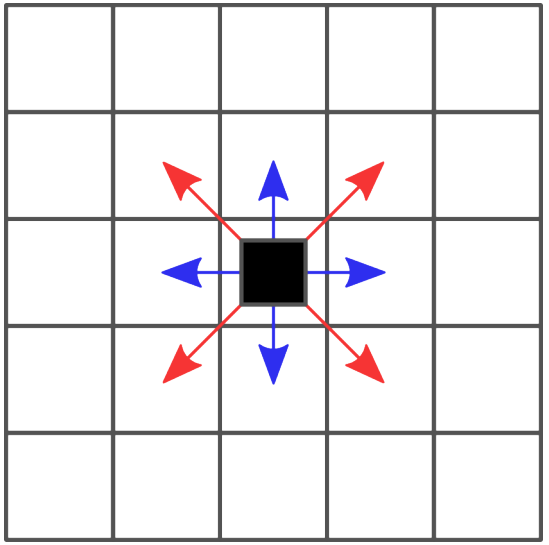
Per generar el graf tenim diverses solucions:

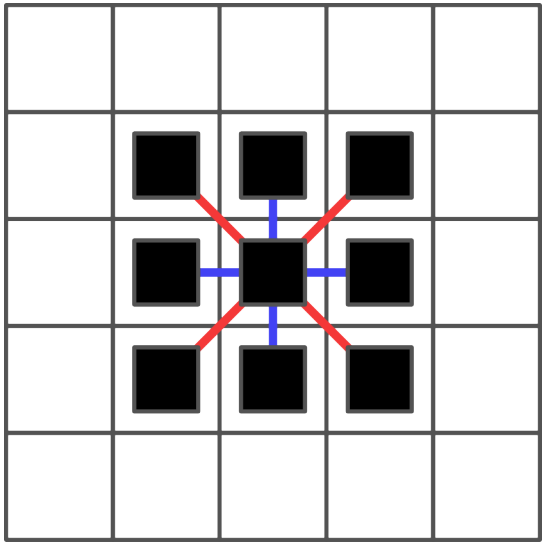
- Divisió en cel·les convexes.
- Discretització en una graella regular.
  - Graella rectangular (normalment quadrada).
    - Cal considerar quina connectivitat es fa servir.
  - Graella hexagonal.

Pathfinding

Graelles rectangulars



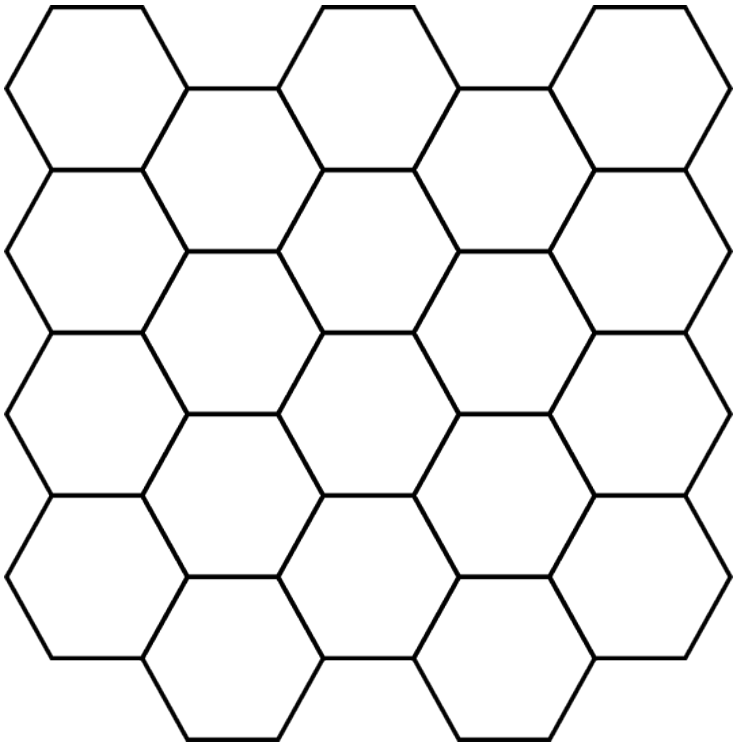


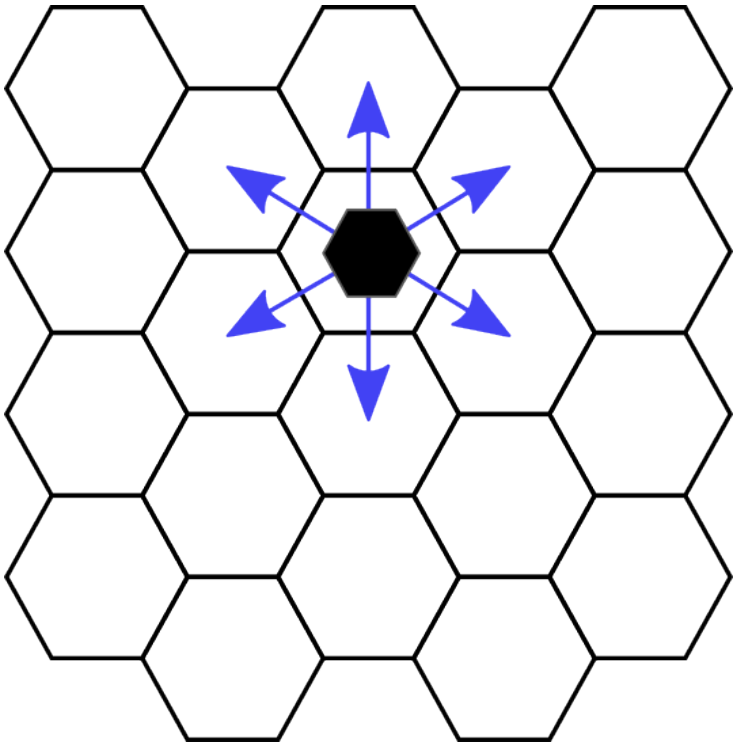


Les distàncies diagonals són diferents ($\sqrt{2}$).

Pathfinding

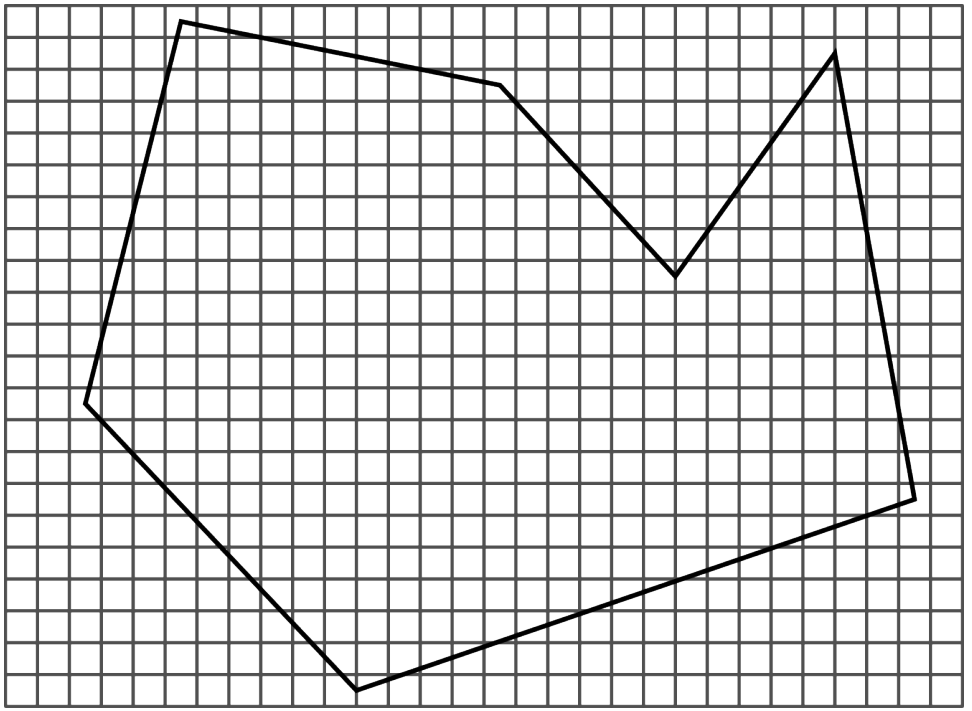
Graelles hexagonals





Pathfinding

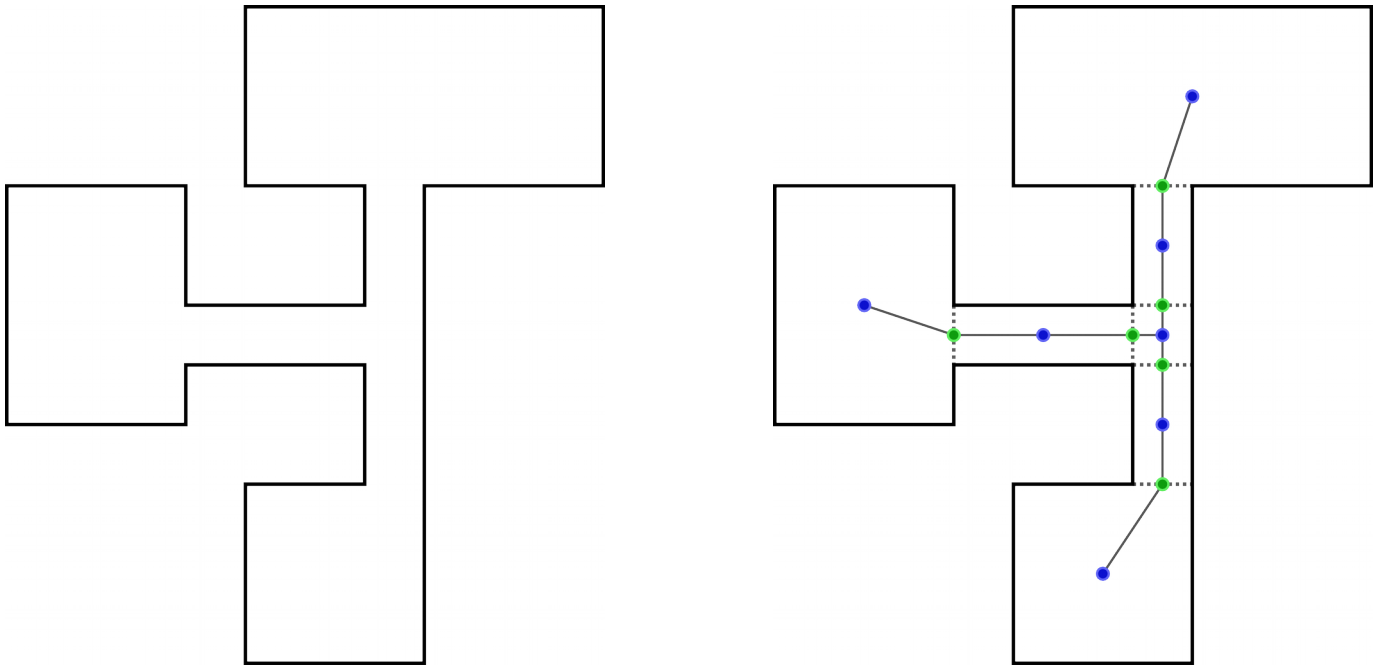
Graelles rectangulars per formes generals



La precisió depèn de la resolució.

Pathfinding

Subdivisió del mon en cel·les convexes i portals



Formes generals, menor consum de memòria.

El graf codifica l'arquitectura del nivell

Pathfinding

Un cop tenim un graf podem calcular camins:

- Necessitem un origen (A) i un destí (B).
- Ens interessa el camí més curt.

Si el nivell és estàtic podem precalcular una taula amb els camins més curts:

- Usant l'algorisme de Dijkstra.
- Per a cada combinació de A i B, només ens cal emmagatzemar el primer pas del camí més curt.

Pathfinding

Taules de camins precalculades

Avantatges

- La consulta és molt ràpida.
- Es poden combinar amb un algorisme de cerca.

Inconvenients

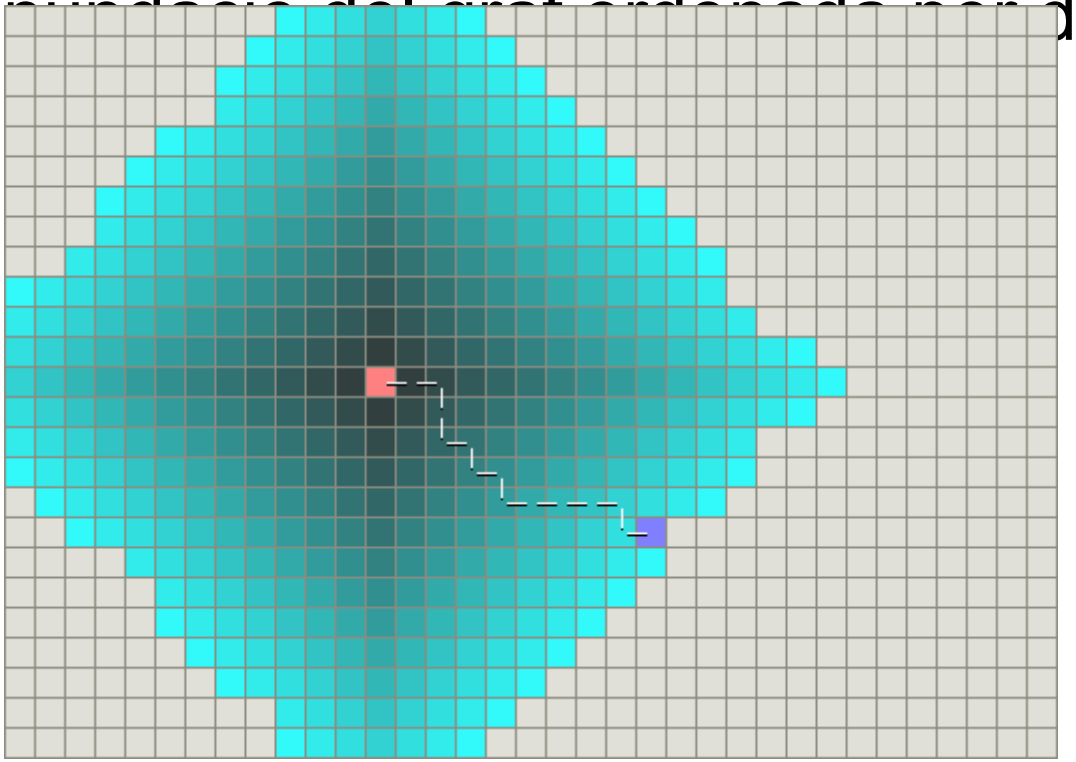
- Molts nodes → Molt consum de memòria.
- Requereixen que el nivell sigui estàtic.

Dijkstra

Serveix per a calcular distàncies d'un node origen A fixat a qualsevol altre node B del graf.

- Es pot modificar per obtenir camins de distància mínima.

Realitza una inundació del graf ordenada per distàncies.



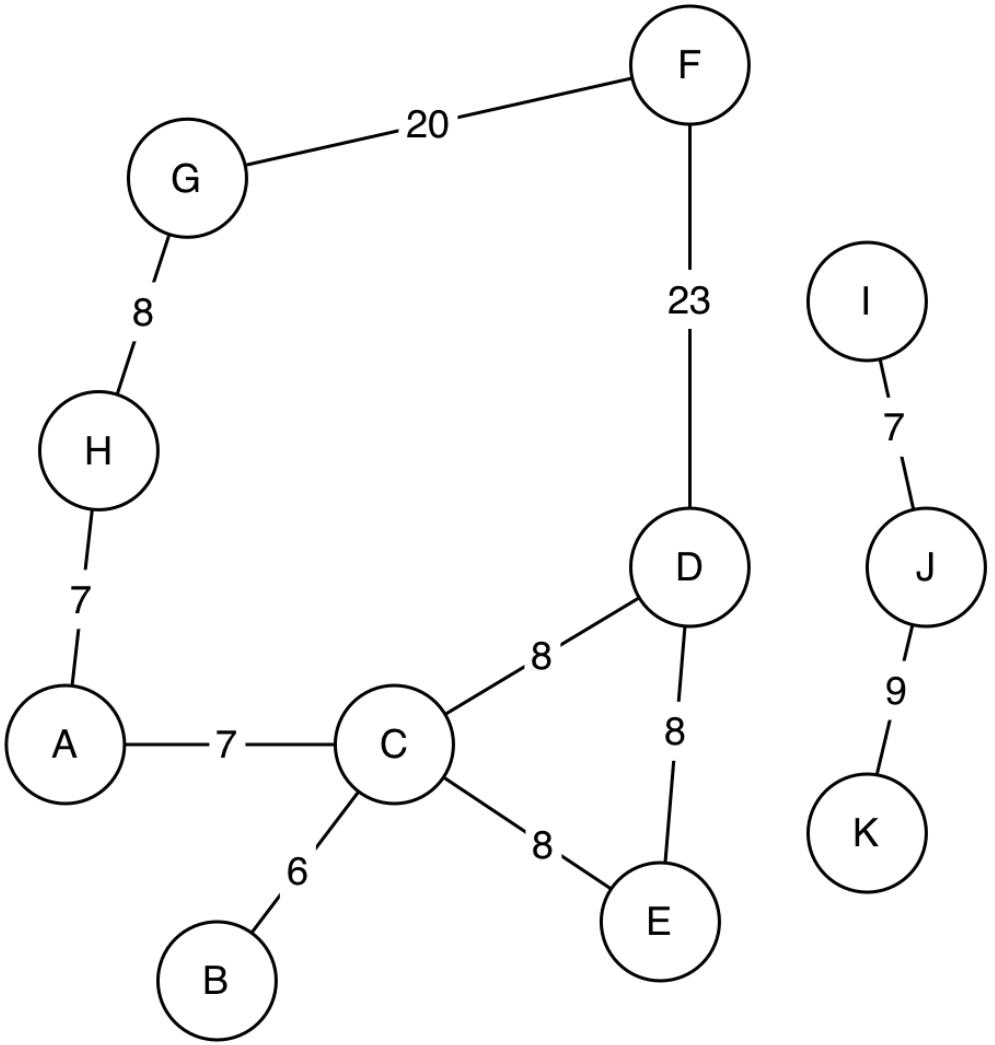
Dijkstra

Algoritme (Distància des de A)

1. Assignem una distància inicial als nodes, zero pel node A, infinit per la resta.
2. Marquem tots els nodes com no visitats.
3. Pel node actual N, calculem la distància dels seus nodes veïns com la suma de la distància de N més la longitud de l'aresta que els uneix. Si la distància calculada és menor que la que tenim al node veí, l'actualitzem.
4. Marquem el node actual com a visitat. La seva distància ja és definitiva.
5. Agafem el node que tingui la mínima distància com a node actual i tornem al pas 3.

Dijkstra

Exemple



A*

Dijkstra està pensat per trobar les distàncies de tots els nodes a un node fixat.

- Bo per precalcular taules.
- Dolent per generar camins durant la simulació.
- Passa perque Dijkstra no té en compte com d'aprop es troba del node objectiu.

A* fa servir un heurístic per tenir en compte a la vegada:

- Distaǹcia del camí mínim a l'origen.
- Distància euclidea al destí.

A*

Volem calcular el camí mínim entre A i B.

A* selecciona el següent node N a considerar agafant el que te mínima:

$$f(N) = g(N) + h(N)$$

On:

N ≡ Node que estem considerant

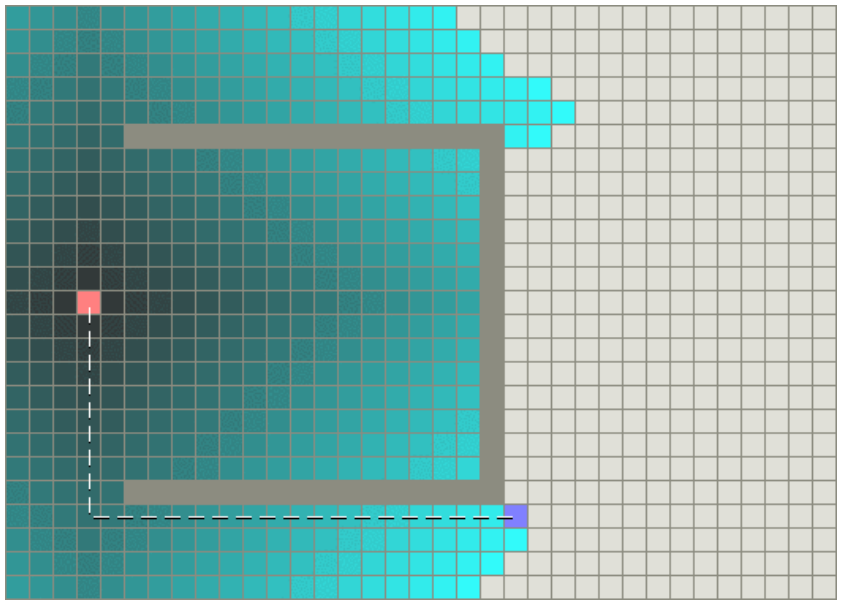
g(N) ≡ Cost del camí mínim des de A a N

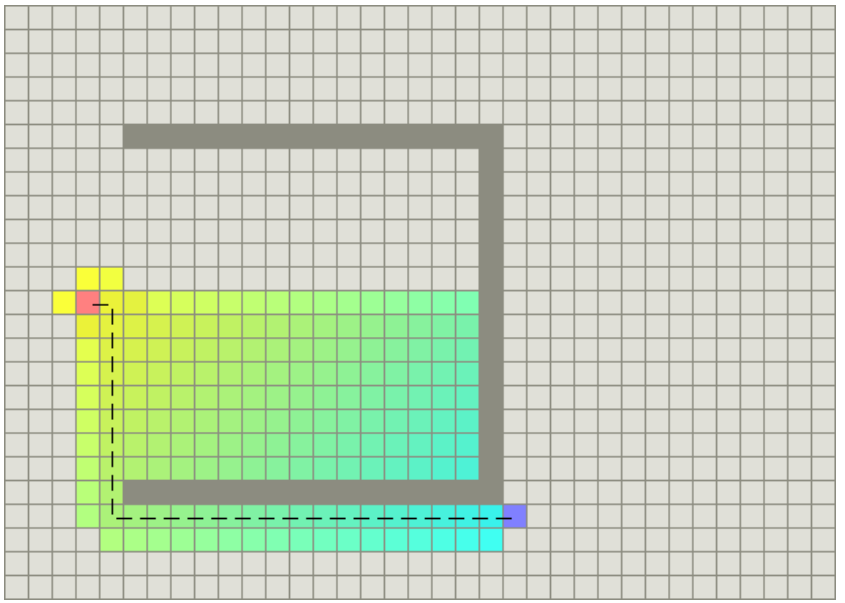
h(N) ≡ Cost estimat de N a B

h(N) és la funció heurística.

A*

Obstacles





A*

Selecció d'heurístic

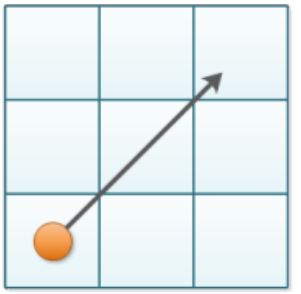
- Si h(N) = 0, llavors només es fa servir g(N).
  - Equivalent a l'algorisme de Dijkstra.
- Si h(N) és sempre menor que la distància restant (del camí mínim) llavors A* ens tornarà el camí mínim.
  - Quan més petit sigui h(N), a més nodes s'expandirà la cerca.
- Si h(N) és igual que la distància del camí mínim, llavors A* segueix el millor camí.
- Si h(N) no és menor que la distància del camí mínim no hi ha garantia d'obtenir el camí mínim.

A*

Heurístics (Graella rectangular)

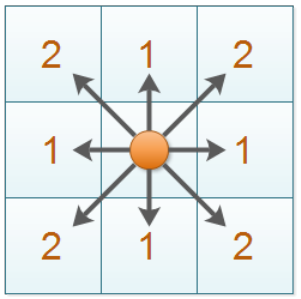
- Connectivitat: 4 veïns
  - Distància euclidea.
  - Distància Manhattan.
- Connectivitat: 8 veïns
  - Distància euclidea.
  - Distància Chebychev.

Euclidean Distance



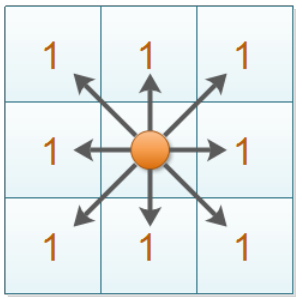
$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Manhattan Distance



$$|x_1 - x_2| + |y_1 - y_2|$$

Chebyshev Distance



$$\max(|x_1 - x_2|, |y_1 - y_2|)$$

A*

Complicacions

- Altres NPCs en moviment.
- Nivell dinàmic.
- Realisme (visibilitat i “fog of war”).

Solucions

- Recalcular parts del camí.
- Desviar al node més proper.
- Quan el bloqueig ve causat per altres NPCs:
  - Esperar un temps abans de provar res.
  - Travessar (no considerar col·lisions entre agents).
  - Esquivar (cal sincronitzar amb l'altre NPC).

Pathfinding

Més informació sobre pathfinding a:

Amit's A* Pages

http://theory.stanford.edu/~amitp/GameProgramming/

IA Determinista

Comportaments preprogramats

Algorismes bàsics

- Trajectòria fixa.
  - Projectils, ...
- Moviment aleatori
  - Cada cert interval de temps, escollir direcció (i potser velocitat) aleatòria.
  - Es pot afegir un pes a la selecció aleatòria de les direccions → Aconseguim un moviment preferent.
- Tracking
  - Seguiment d'un objecte (comunment el jugador).
  - La velocitat depèn de la posició i/o distància de l'objectiu.

Tracking

Seguiment directe.

Donada la posició del jugador (Jx, Jy)

i la d'un NPC (Mx, My).

```
Vx = Jx - Mx;
Vy = Jy - My;
length = sqrt(Vx*Vx + Vy*Vy);
Vx *= MAX_VEL / length;
Vy *= MAX_VEL / length;
```

on MAX_V és la velocitat màxima del NPC.

Desavantatges:

- Massa precís → Poc realista
- No té en compte el moviment del seu objectiu.

Tracking

Podem canviar la velocitat de forma gradual.

Suposant que el jugador tingui velocitat (Vx, Vy):

```
TVx = Jx - Mx;
TVy = Jy - My;
length = sqrt(TVx*TVx + TVy*TVy);
TVx /= length;
TVy /= length;
Vx += factor * TVx;
Vy += factor * TVy;
```

També podem tenir en compte el moviment de l'objectiu:

- Substituir (Jx, Jy) per (Jx + Vx * Δt, Jy + Vy * Δt)

Patrons de moviment

Els NPCs segueixen una llista d'accions.

Exemple:

```
enum NPCAction {Forward, Backward,
    TurnRight, TurnLeft, TurnAround, Shoot};
int nGuardPattern = 2;
NPCAction guardPattern[] = {0, 4};
Int nAttackFrenzy = 8;
NPCAction attackFrenzy[] = {5, 2, 5, 2, 5, 2, 5, 2};
```

Extensions

- Condicions per passar a la següent acció de la llista.
- Temps d'execució per a cada acció de la llista.

Patrons de moviment

- Cal garantir que els NPCs respetin les regles del mon del joc.
  - Especialment col·lisions.
- Patrons múltiples
  - Escollir entre diferents patrons aleatòriament.

    o
  - Escollir un patró segons les condicions actuals.
    - Distància al jugador.
    - Visibilitat.
    - ...

Màquines d'estats

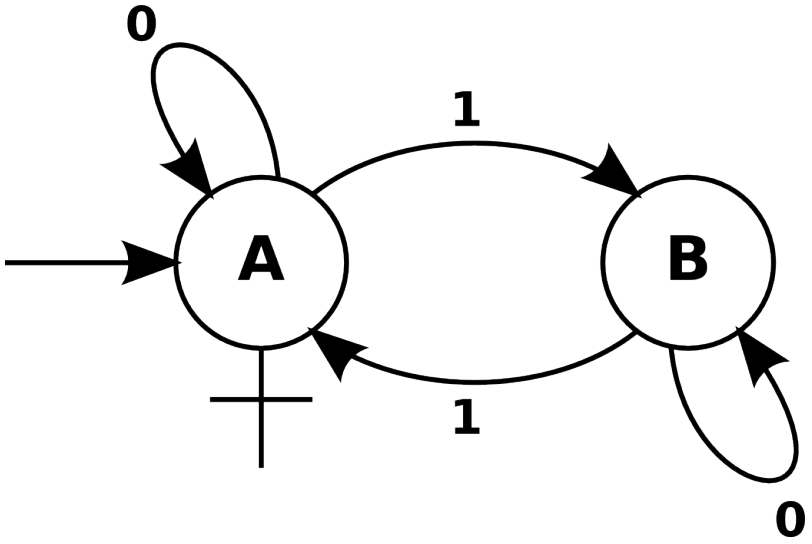
Noms

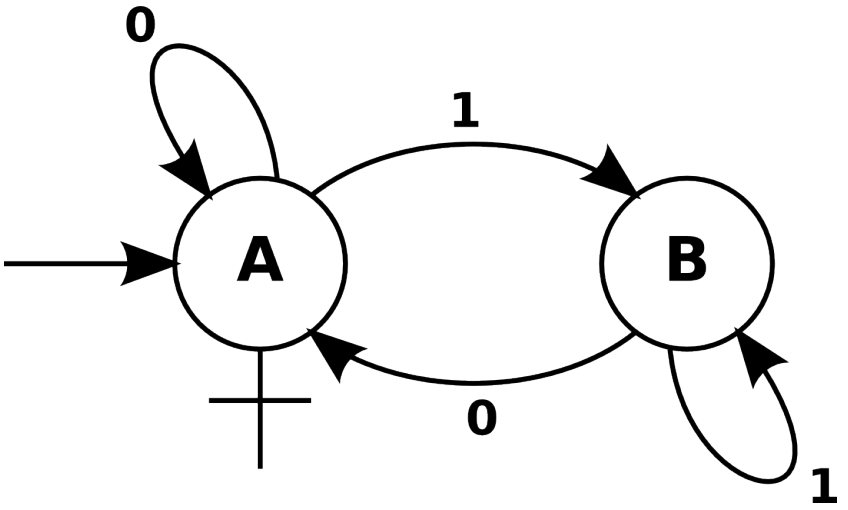
- Màquines d'estats fínits (FSM)
- Autòmates finits.

Model matemàtic de computació

- Representats com a grafs dirigits
- Conjunt d'estats
  - Es representen com a nodes del graf.
- Conjunt de transicions
  - Arc dirigit més entrada que causa la transició

Màquines d'estats





El efecte d'una entrada depèn de l'estat actual.

Memòria limitada → Menys capacitat de computació que una màquina de Turing

Màquines d'estats

En els videojocs

- Cada estat representa un comportament.
- Cada estat sap processar la informació del seu entorn per a realitzar les seves transicions.
- Els estats “finals” aturen l'execució de la màquina d'estats.
- L'execució de la màquina d'estats es fa durant tot el joc.
- La IA d'un NPC es pot representar com una seqüència de comportaments governada per la FSM.

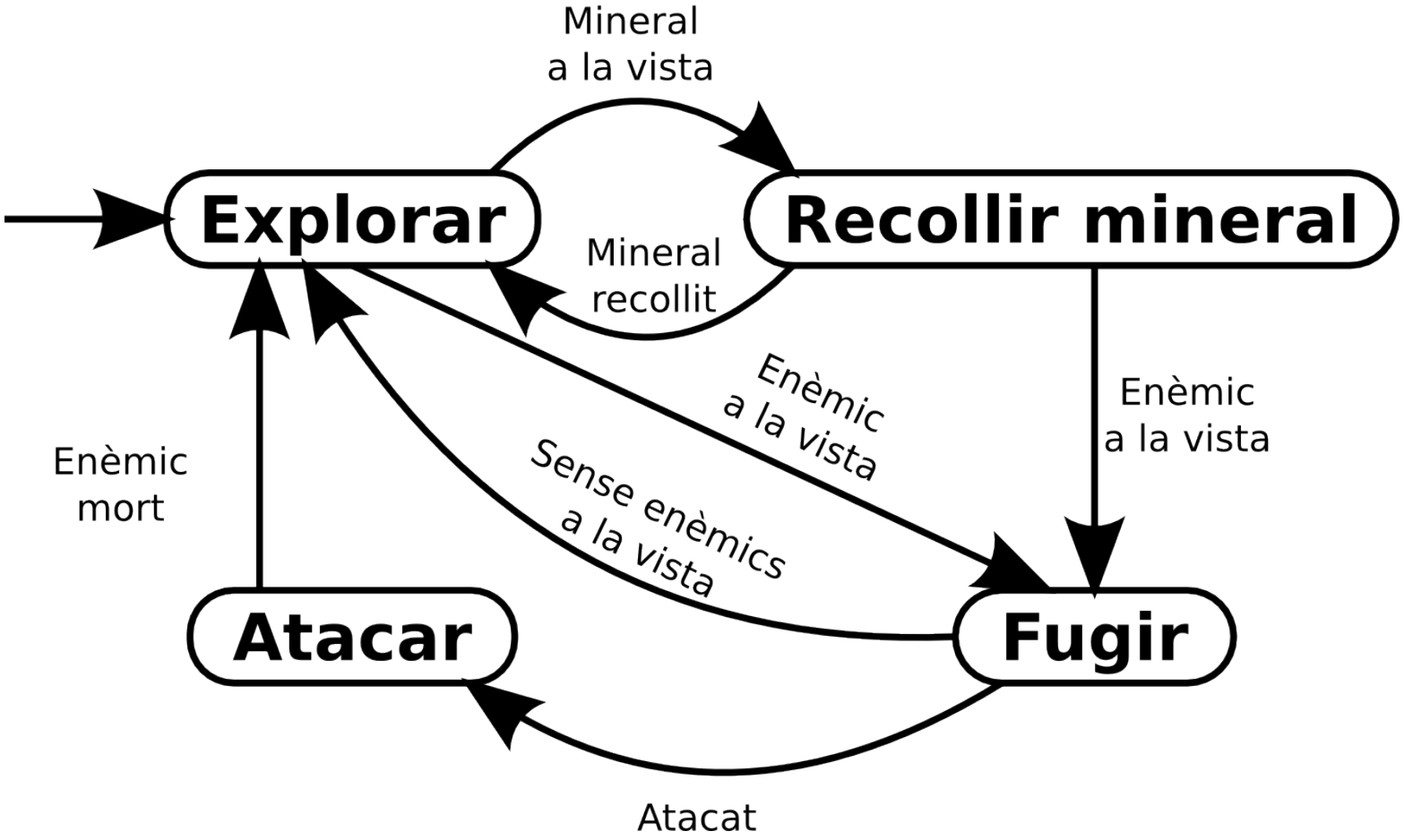
Màquines d'estats

Avantatges

- Fàcil de comprendre i desenvolupar.
- Ús de GUIs per la seva creació i edició.
- Via les possibles entrades podem afegir l'efecte del pathfinding o d'altres moduls.

Els comportaments (estats) es poden modelar com a IA deterministes o patrons d'accions.

Màquines d'estats



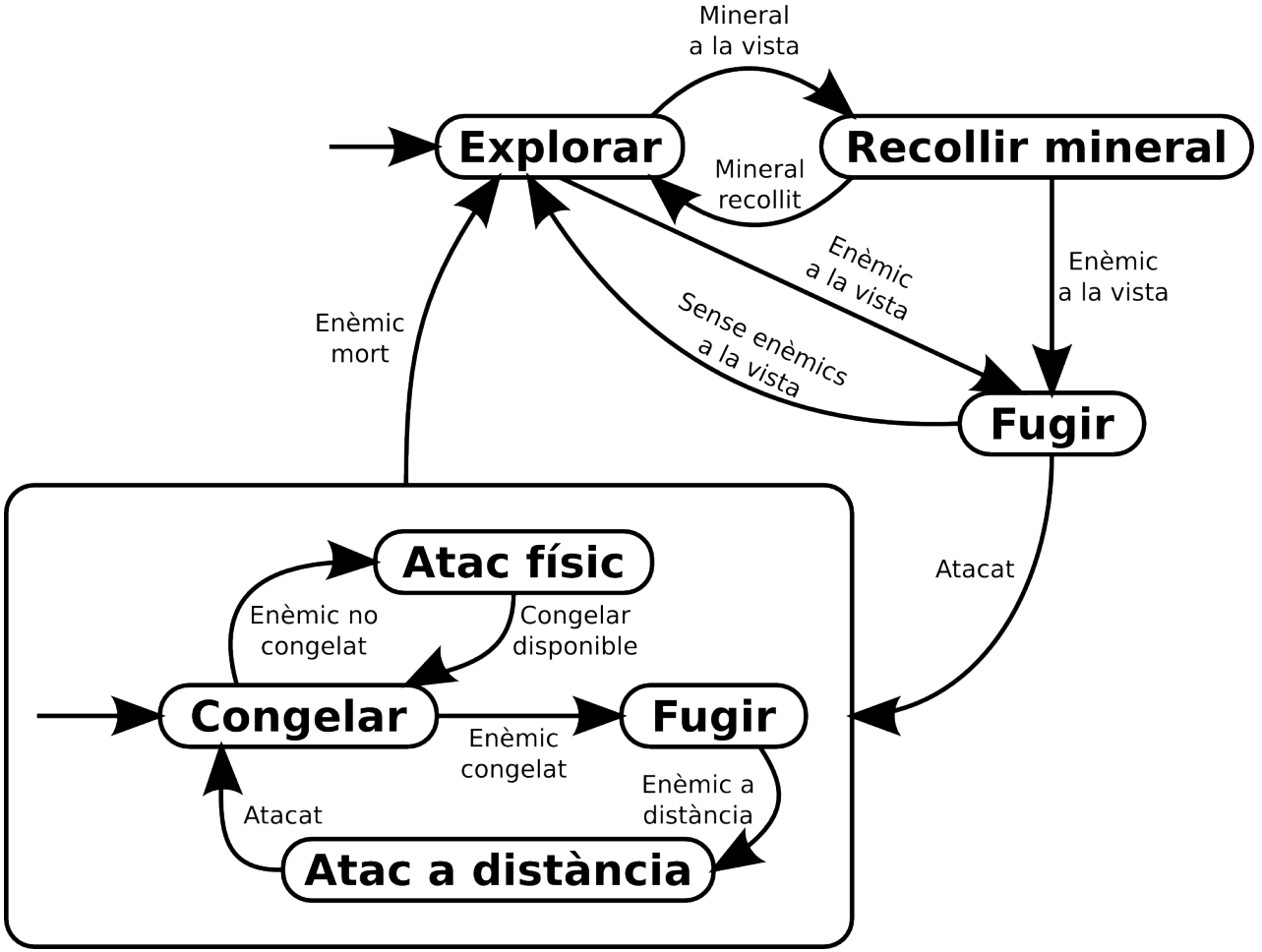
Màquines d'estats

Extensions

- Les FSMs no recorden més que l'estat actual → Afegir memòria a cada NPC
  - Fer-la servir com una entrada més
- FSMs jeràrquiques
  - Modelar els estats amb la seva pròpia FSM.
- FSMs no deterministes
  - Donat un estat actual i un conjunt d'entrades, tenir múltiples estats següents.
    - Escollir entre ells de forma aleatòria.

Màquines d'estats

FSMs jeràrquiques



Combinació

Per tipus d'objectes

- IA determinista
  - Objectes amb comportaments molt simples.
- Patrons
  - Objectes amb comportaments complexes.
  - NPCs que no interactuen amb el jugador.
- Màquines d'estats simples
  - NPCs que interactuen amb el jugador.
- Màquines d'estats complexes (memòria, no determinista)
  - NPCs principals.
  - Companys del jugador.