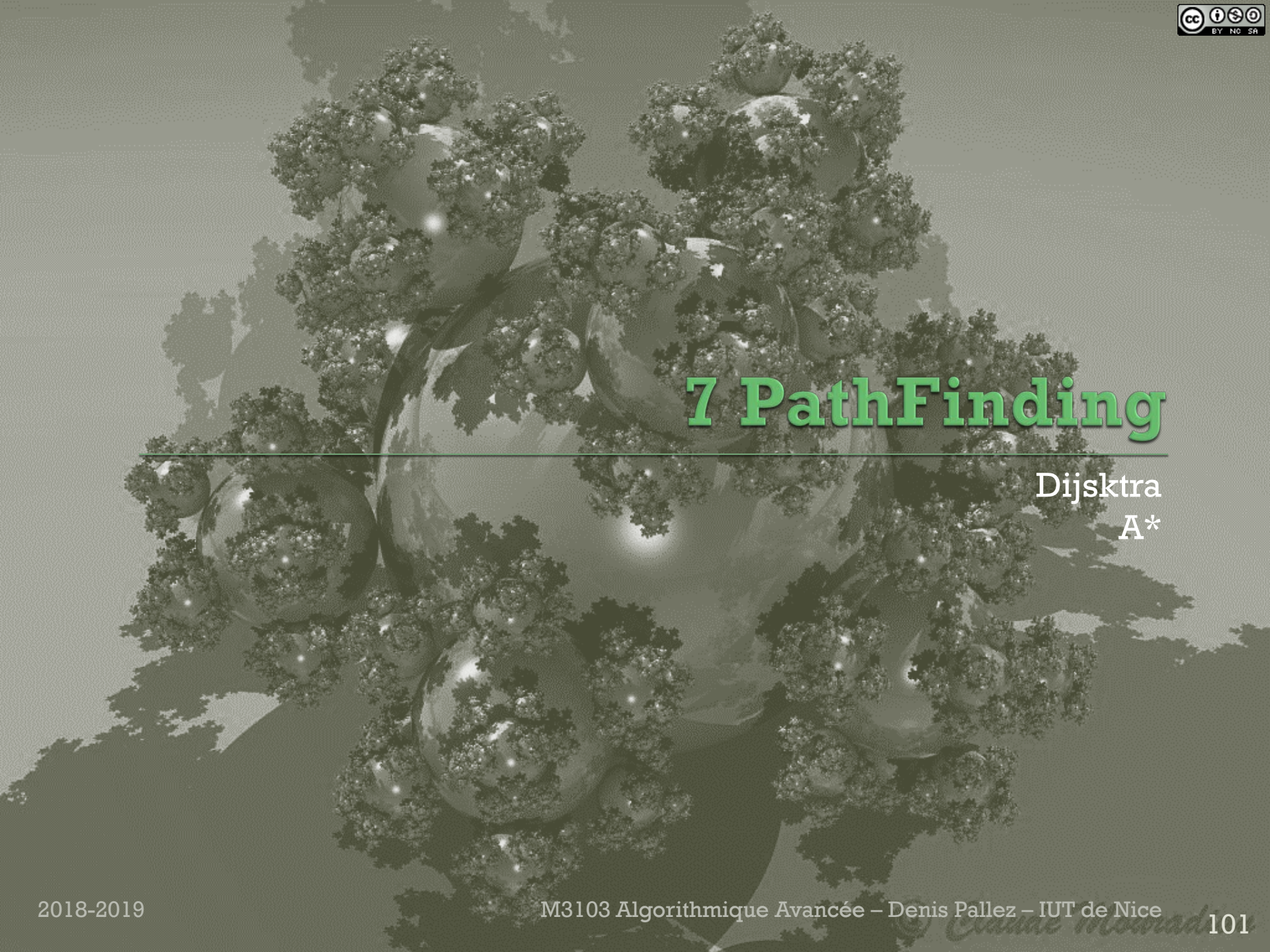


7 PathFinding

Dijsktra

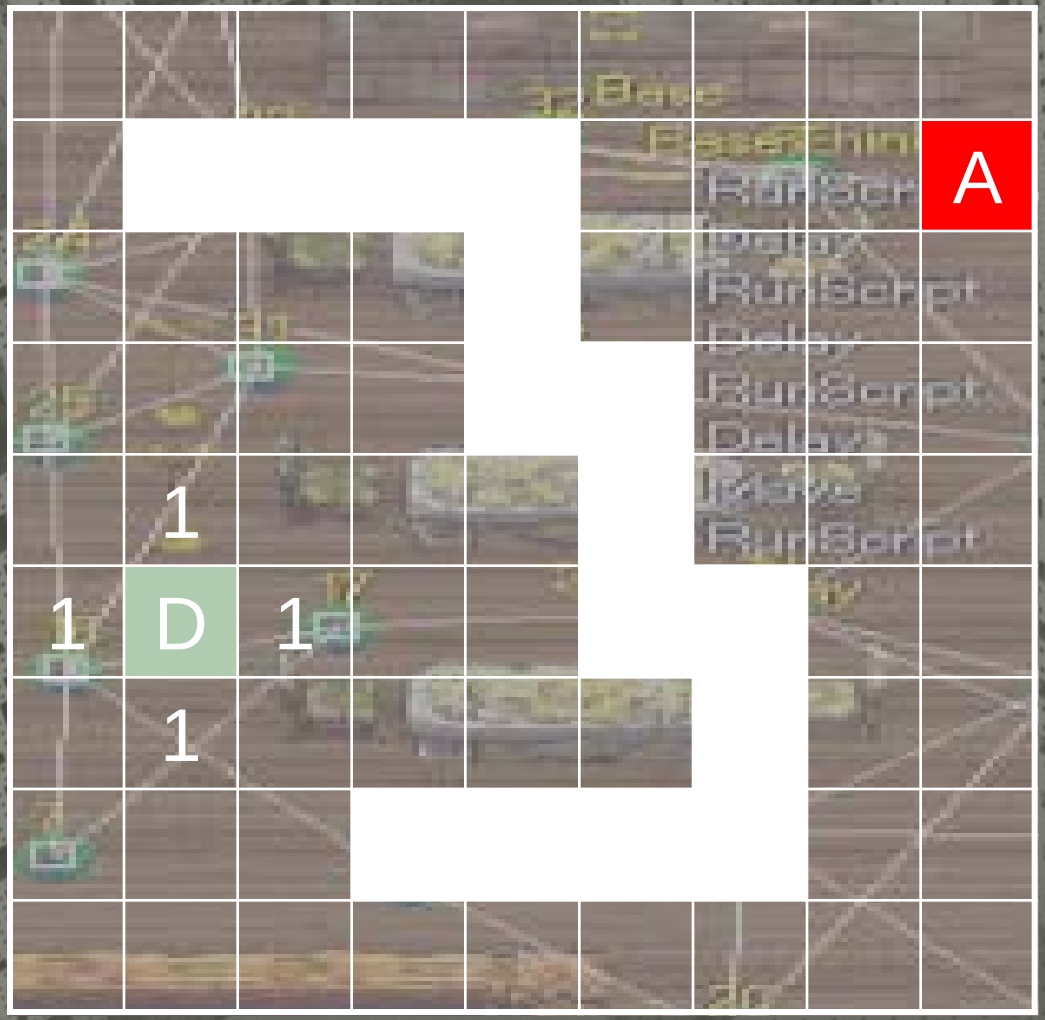
A*

PathFinding

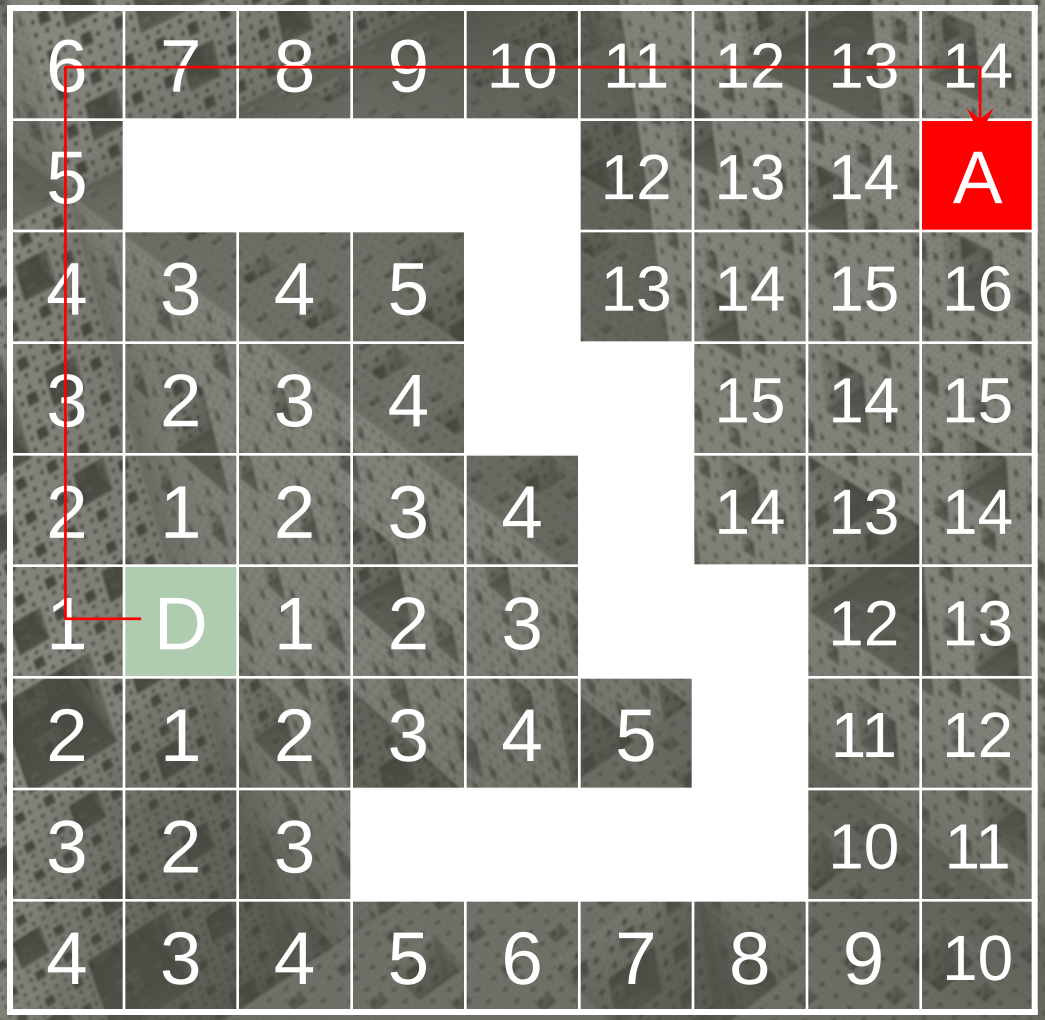
PathFinding

- Discrétisation de l'environnement

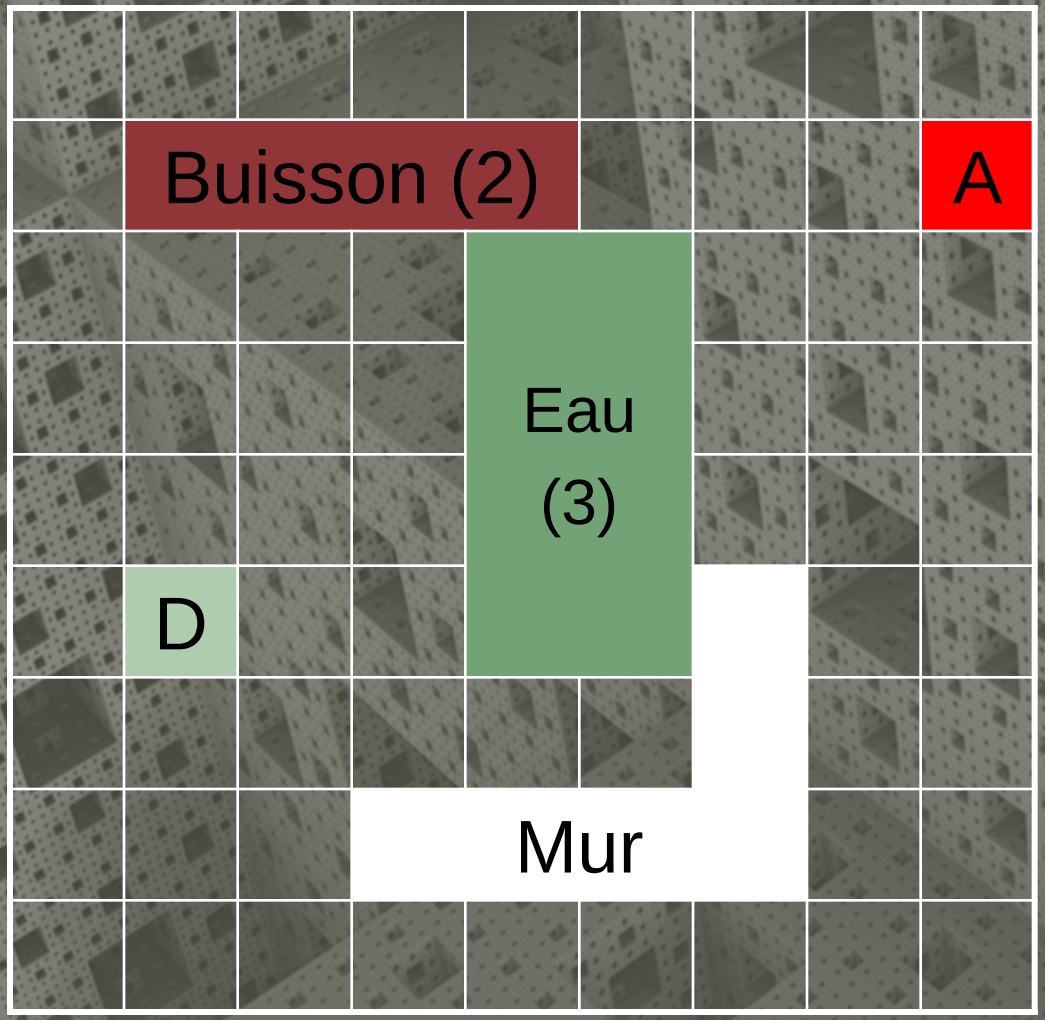
PathFinding – Dijsktra



PathFinding – Dijsktra

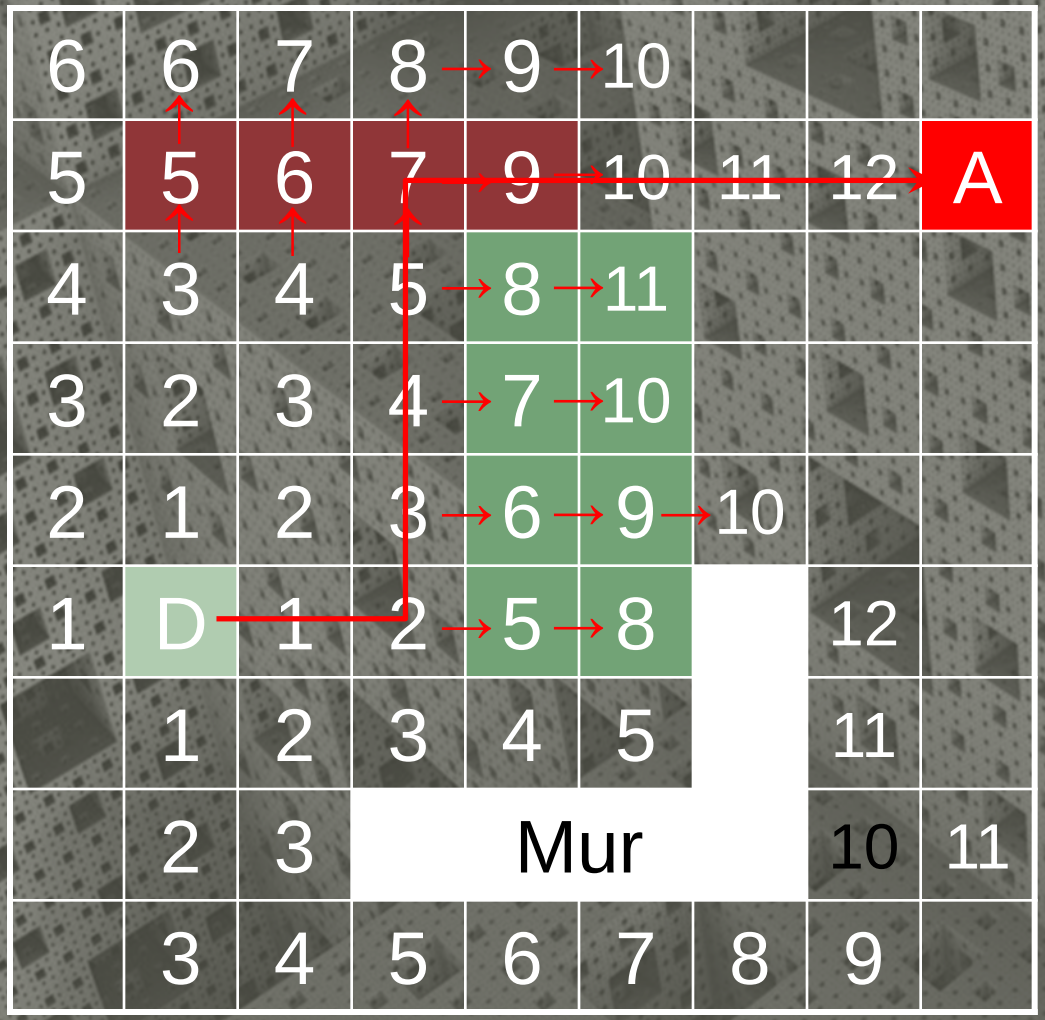


PathFinding – Dijsktra



?

PathFinding – Dijsktra



PathFinding – Dijsktra

- **Dijsktra**
  - Analyse tous les voisins du nœud de départ et re-considère chaque voisin comme le nouveau départ
  - Est-ce nécessaire ?
- **A***
  - Utilise une *Heuristique* pour guider la recherche d'une bonne solution
    - Méthode qui ***estime*** la distance entre 2 nœuds
      - Ex: Distance de Manhattan
  - Minimise le nombre de nœuds à regarder

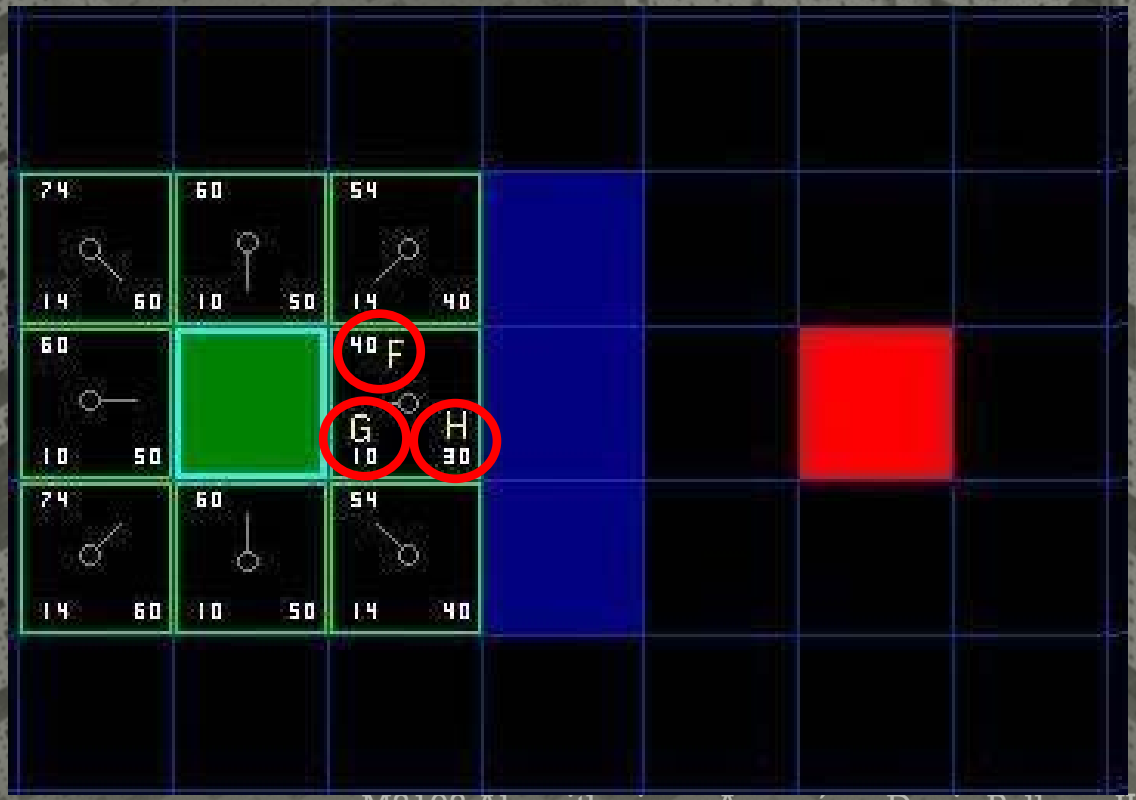
A*

- Algorithme pour aller de Start à Goal
  - Utilise 2 listes
    - Liste « Ouverte » LO
      - Nœuds à considérer comme points de départs pour une extension du chemin
    - Liste « Fermée » LF
      - Nœuds dont les voisins ont déjà été ajoutés à LO
  - Score G
    - Contient coût pour aller de start au nœud courant
    - Un petit nombre est préférable
    - Chaque nœud possède ce score
  - Score H (Heuristique)
    - Même idée que G mais est une estimation entre nœud courant et goal
  - Score F = G+H
  - Au début
    - LF = ∅, LO = {Start}
    - Pour chaque nœud, F=G=∞
    - Pour Start, G=0 ⇒ F=H, Parent(Start) = ∅

A*

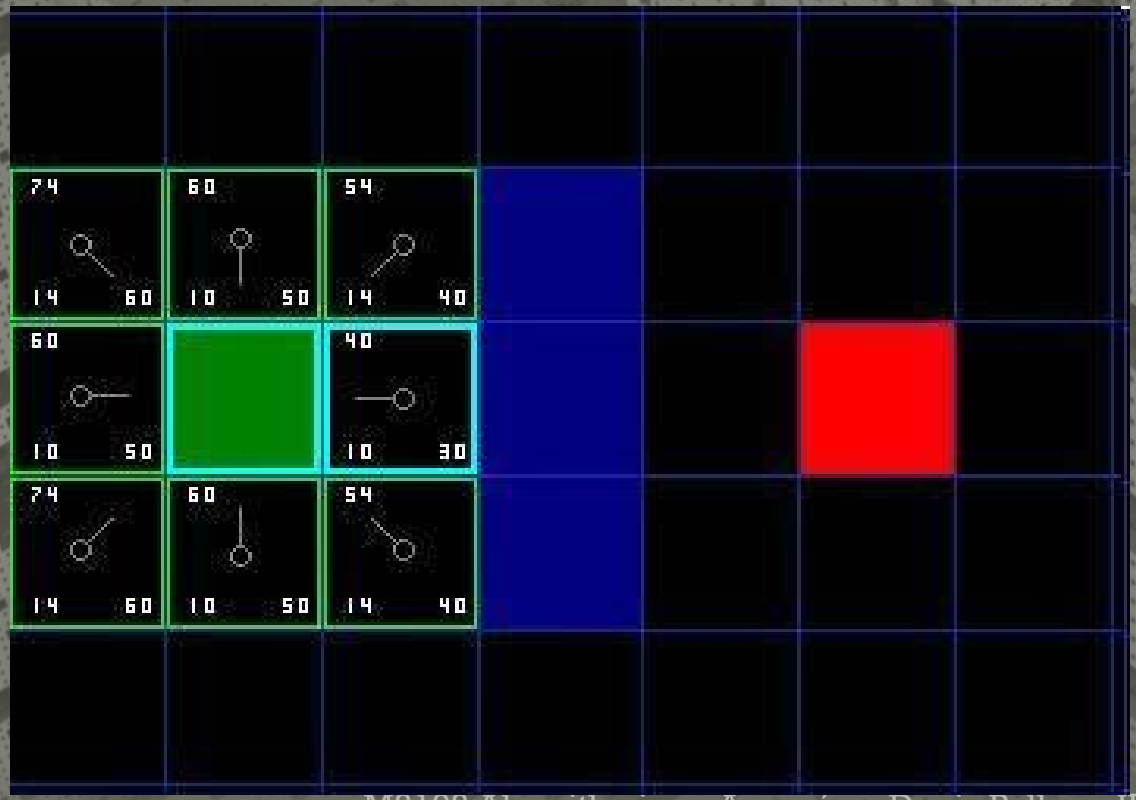
- Algorithme
  - Tant que LO non vide et pas trouvé Goal
    - Choisir le nœud (P) ds LO qui possède le plus petit F=G+H
    - Supprimer P ds LO et l'ajouter ds LF
    - Si P=Goal Alors trouvé
    - Sinon
      - Pour tous les nœuds (N) adjacent à P qui $\notin$ LF
        - Calculer G(N) = G(P) + dist(N,P)
        - Si N ds LO Alors
          - Conserver meilleur G(N) et modifier Parent(N) si nécessaire
        - Sinon
          - Ajouter N ds LO avec son G(N) et Parent(N)
  - Reconstruire chemin à partir de P et Parent(P)

- Exemple

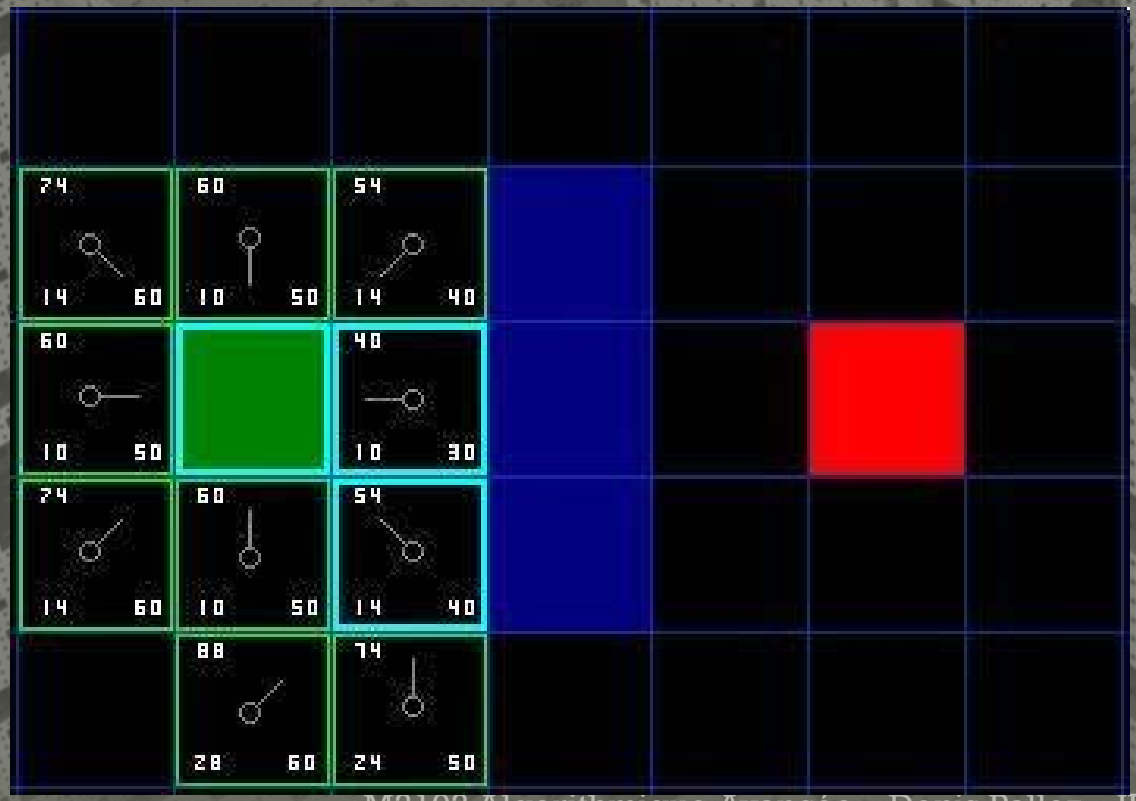

74
14 60
60
10 50
54
14 40
60
10 50
40 F
G 10
H 30
74
14 60
60
10 50
54
14 40

A*

- Exemple
  - A noter : 2 nœuds à 54 (choisir le dernier ajouté)

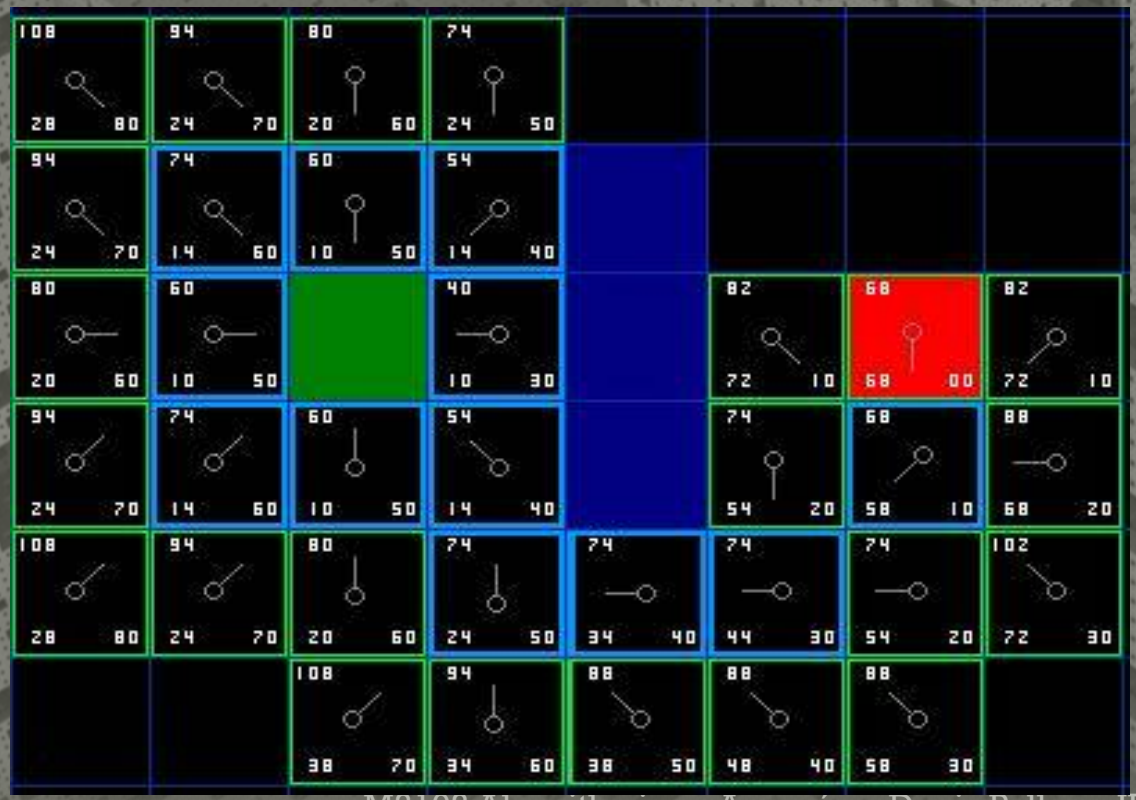


- Exemple


74
14 60
60
10 50
54
14 40
60
10 50
40
10 30
74
14 60
60
10 50
54
14 40
88
28 60
74
24 50

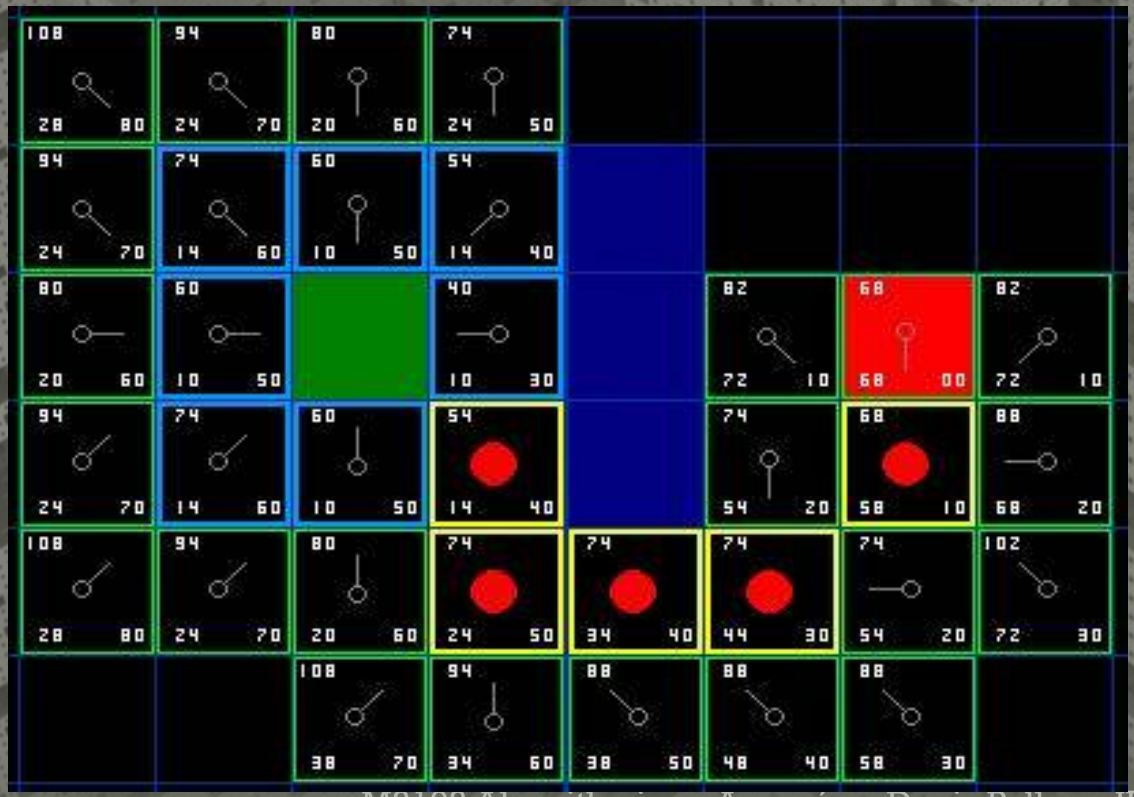
A*

- Exemple


108 28 80
94 24 70
80 20 60
74 24 50
94 24 70
74 14 60
60 10 50
54 14 40
80 20 60
60 10 50
40 10 30
82 72 10
68 68 00
82 72 10
94 24 70
74 14 60
60 10 50
54 14 40
74 54 20
68 58 10
88 68 20
108 28 80
94 24 70
80 20 60
74 24 50
74 34 40
74 44 30
74 54 20
102 72 30
108 38 70
94 34 60
88 38 50
88 48 40
88 58 30

A*

- Exemple


108 94 80 74
28 80 24 70 20 60 24 50
94 74 60 54
24 70 14 60 10 50 14 40
80 60 40 82 68 82
20 60 10 50 10 30 72 10 68 00 72 10
94 74 60 54 74 68 88
24 70 14 60 10 50 14 40 54 20 58 10 68 20
108 94 80 74 74 74 74 102
28 80 24 70 20 60 24 50 34 40 44 30 54 20 72 30
108 94 88 88 88
38 70 34 60 38 50 48 40 58 30

Dijkstra, A* & co

http://buildnewgames.com/astar/
https://youtu.be/-L-WgKMFuhE

- Dijkstra
  - explore tous les alentours du point de départ jusqu'à trouver le point d'arrivée
  - N'utilises pas d'heuristiques
  - Utile quand on ne connait pas le point d'arrivée
- A*
  - Explore moins de nœuds
  - Utilise une heuristique
  - S'exécute plus rapidement
  - SI heuristique= null ALORS A*=Dijkstra
- A* optimisé (hiérarchique)
  - Deviner la meilleure route plutôt que de recommencer depuis le début quand c'est bloqué