UQÀM
Université du Québec à Montréal

AD*

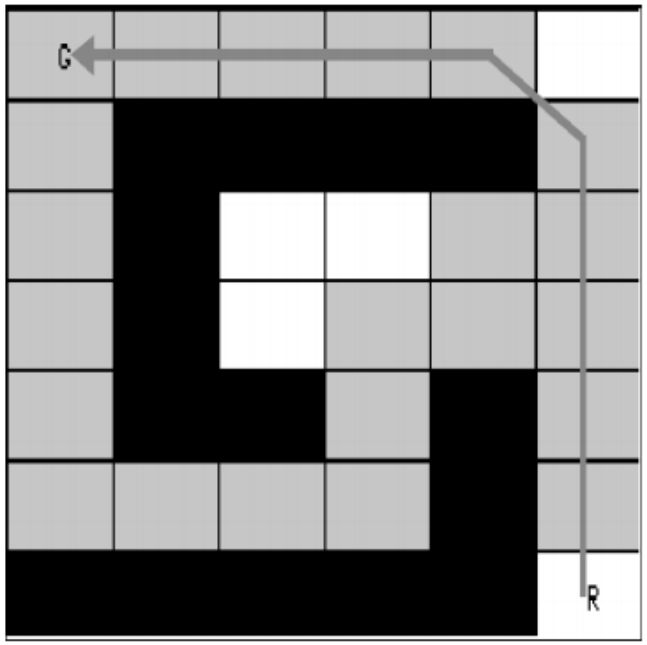
ANYTIME DYNAMIC A*

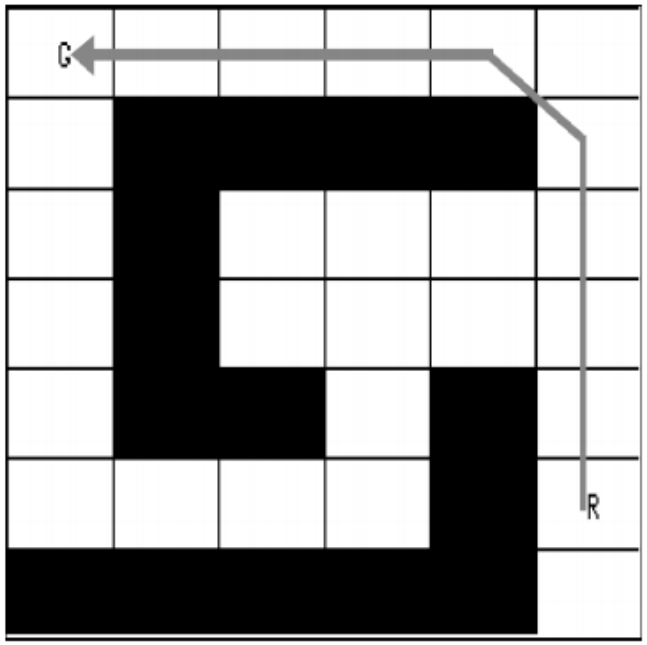
Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. *(2005) Anytime Dynamic A*: An Anytime, Replanning Algorithm*

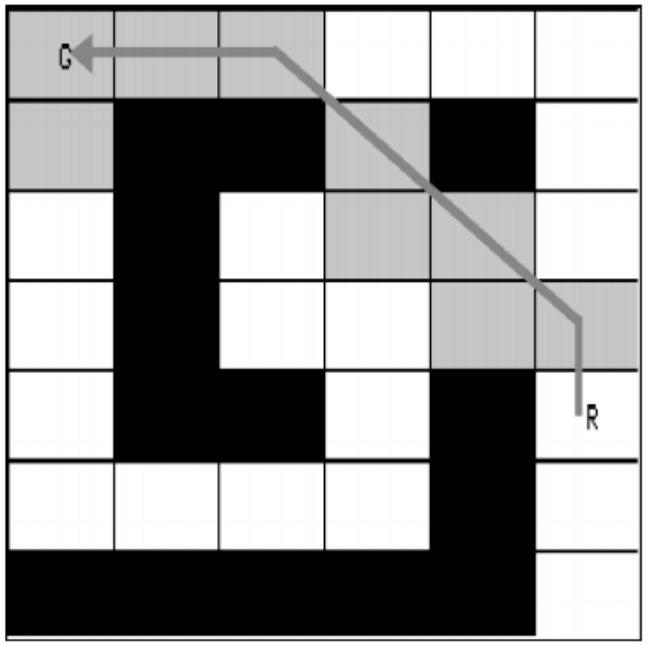
Sommaire

Les défauts de A*

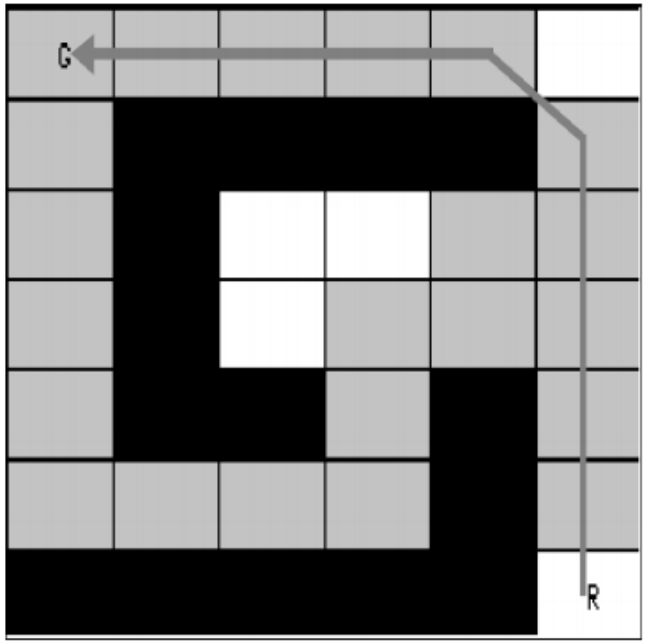
A*

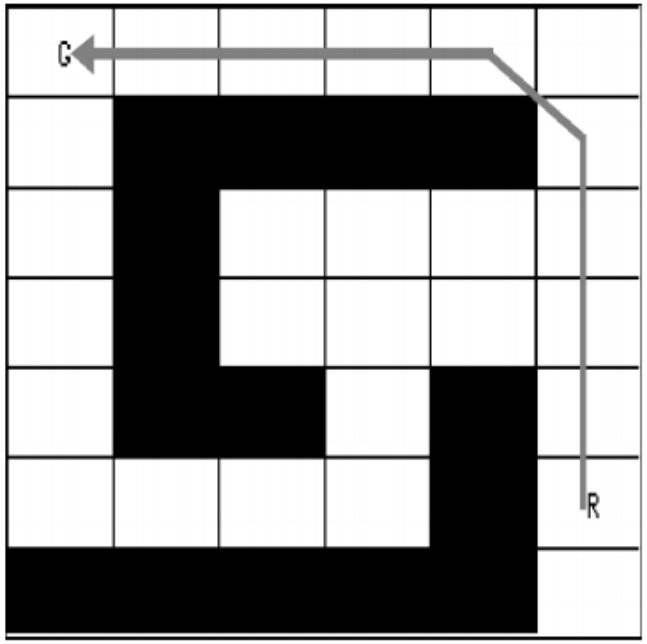


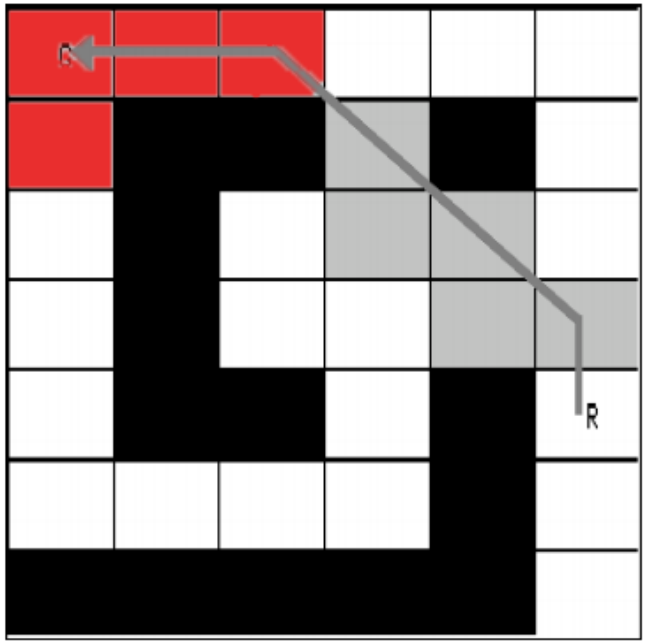




A*







L'algorithme D* Lite

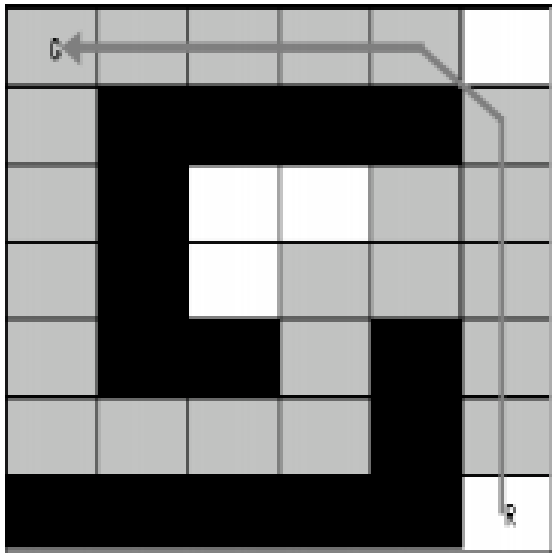
UN ALGORITHME DYNAMIQUE

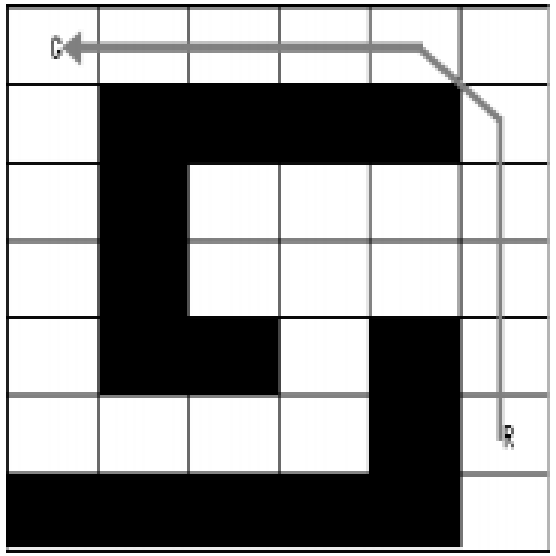
D* Lite

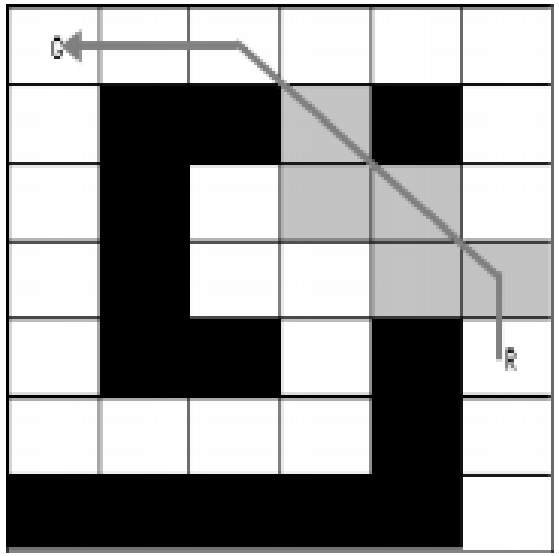
- Basé sur la recherche progressive
- Créé par Sven Koenig and Maxim Likhachev en 2002
- Utilisé dans des projets tels que Opportunity et Spirit

D* Lite

Déroulement de l'exécution :

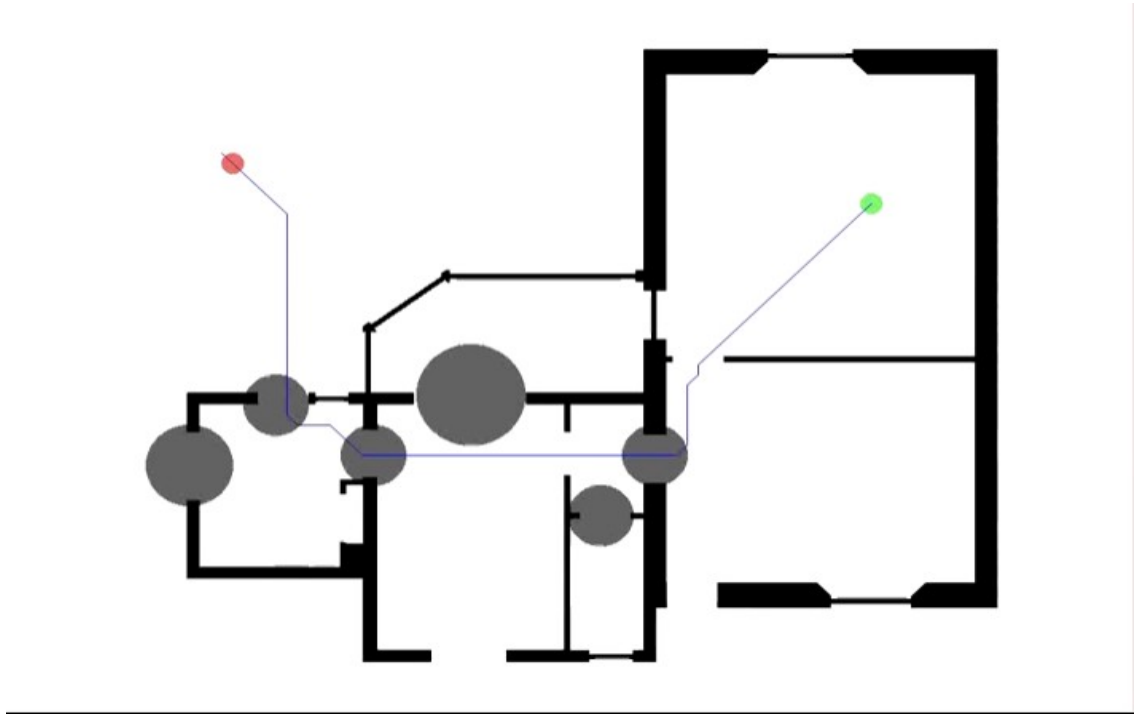






D* Lite

Exemple concret



LukeYoder

L’algorithme ARA*

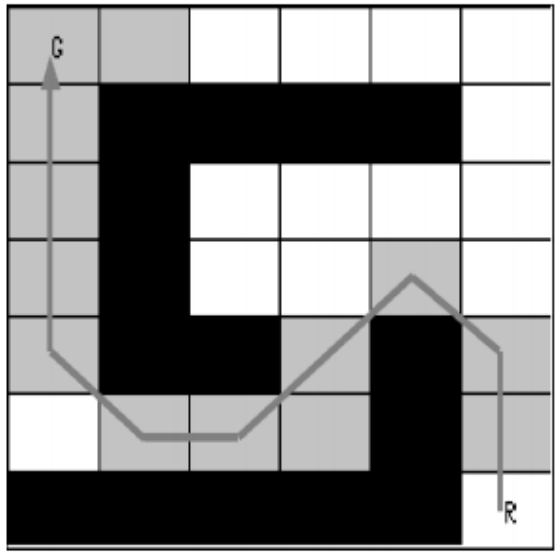
UN ALGORITHME ANYTIME

ARA*

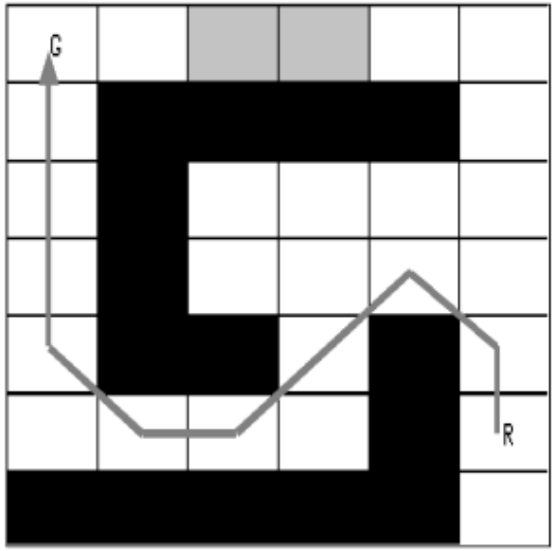
- Algorithme « anytime »
- Créé par Maxim Likhachev, Geoff Gordon et Sebastian Thrun en 2003
- Très utilisé pour les arbres de decisions complexes

ARA*

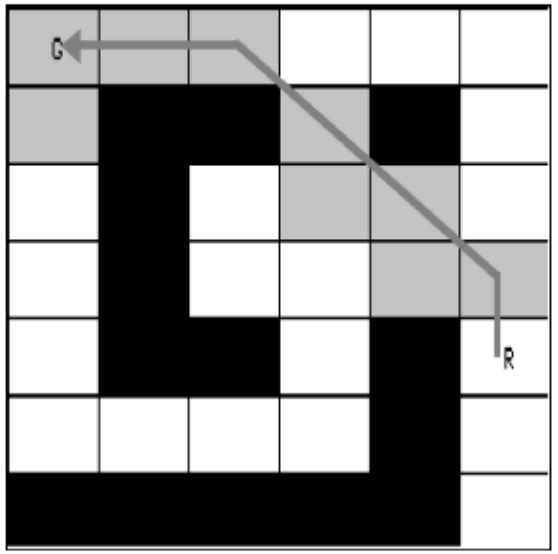
Déroulement de l'exécution :



$\epsilon = 2.5$



$\epsilon = 1.5$



$\epsilon = 1.0$

L’algorithme AD*

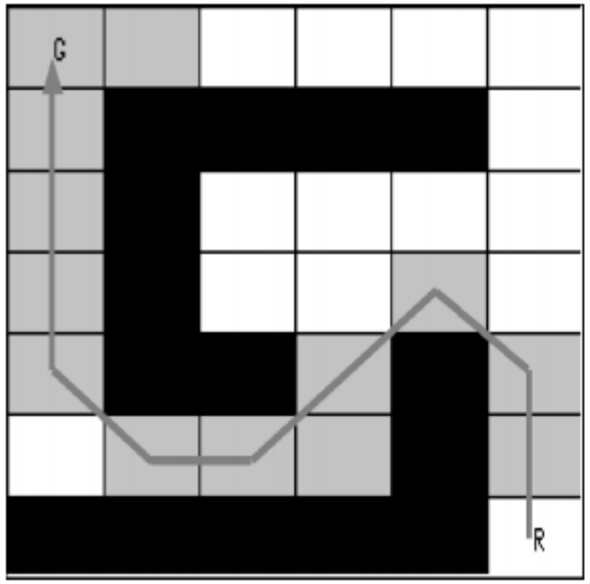
UN ANYTIME D*

AD*

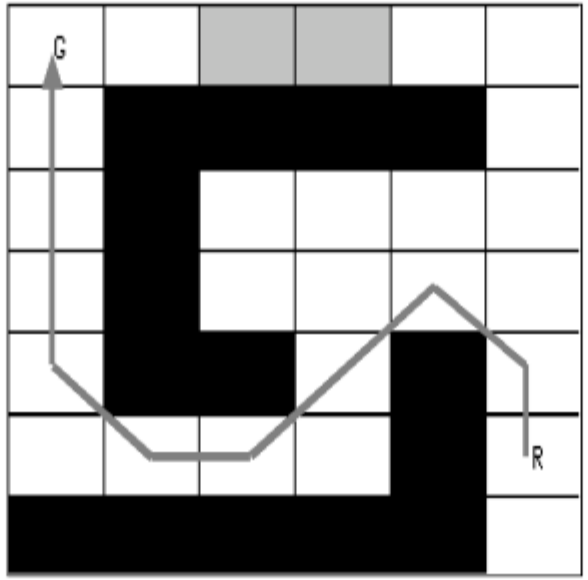
- Mélange des deux derniers algorithmes
- Créé en 2005 par Maxim Likhachev, Geoff Gordon, Sebastian Thrun, Dave Ferguson et Anthony Stentz

AD*

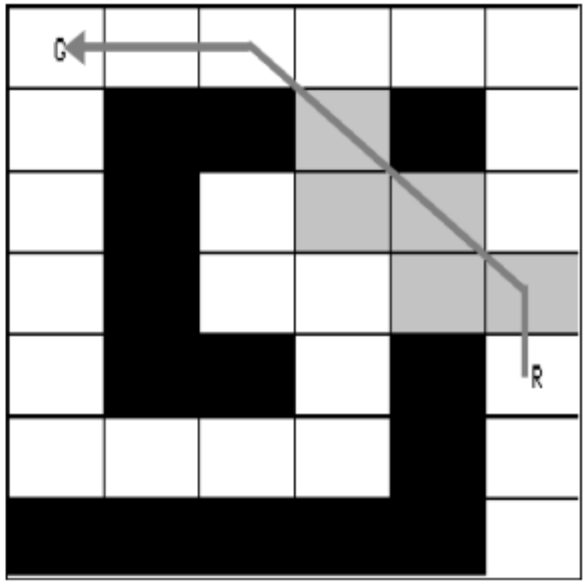
Déroulement de l'exécution :



$\epsilon = 2.5$



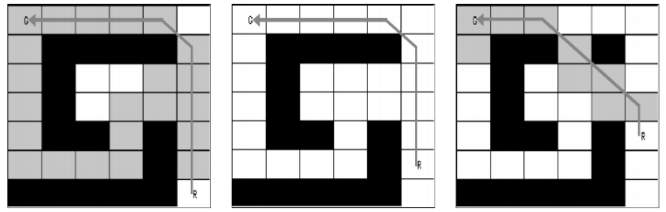
$\epsilon = 1.5$



$\epsilon = 1.0$

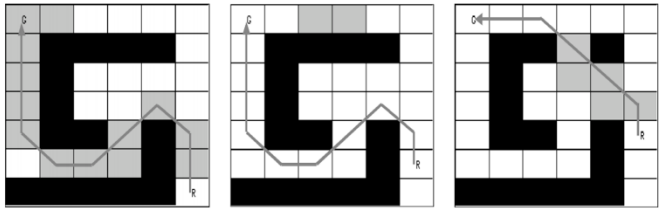
Comparaison des algorithmes

A*



= 31

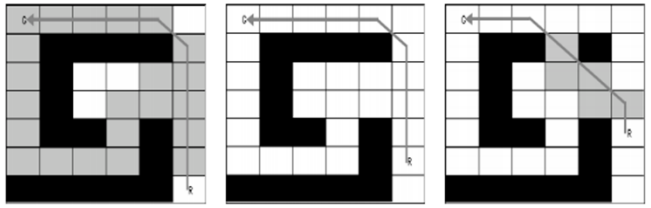
AD*



$\epsilon = 2.5$ $\epsilon = 1.5$ $\epsilon = 1.0$

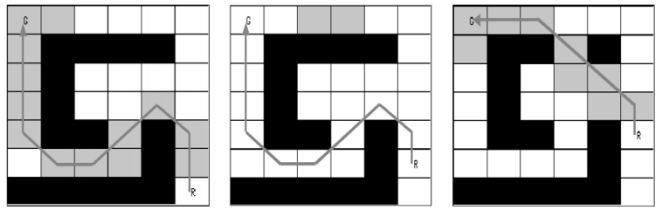
= 20

D* Lite



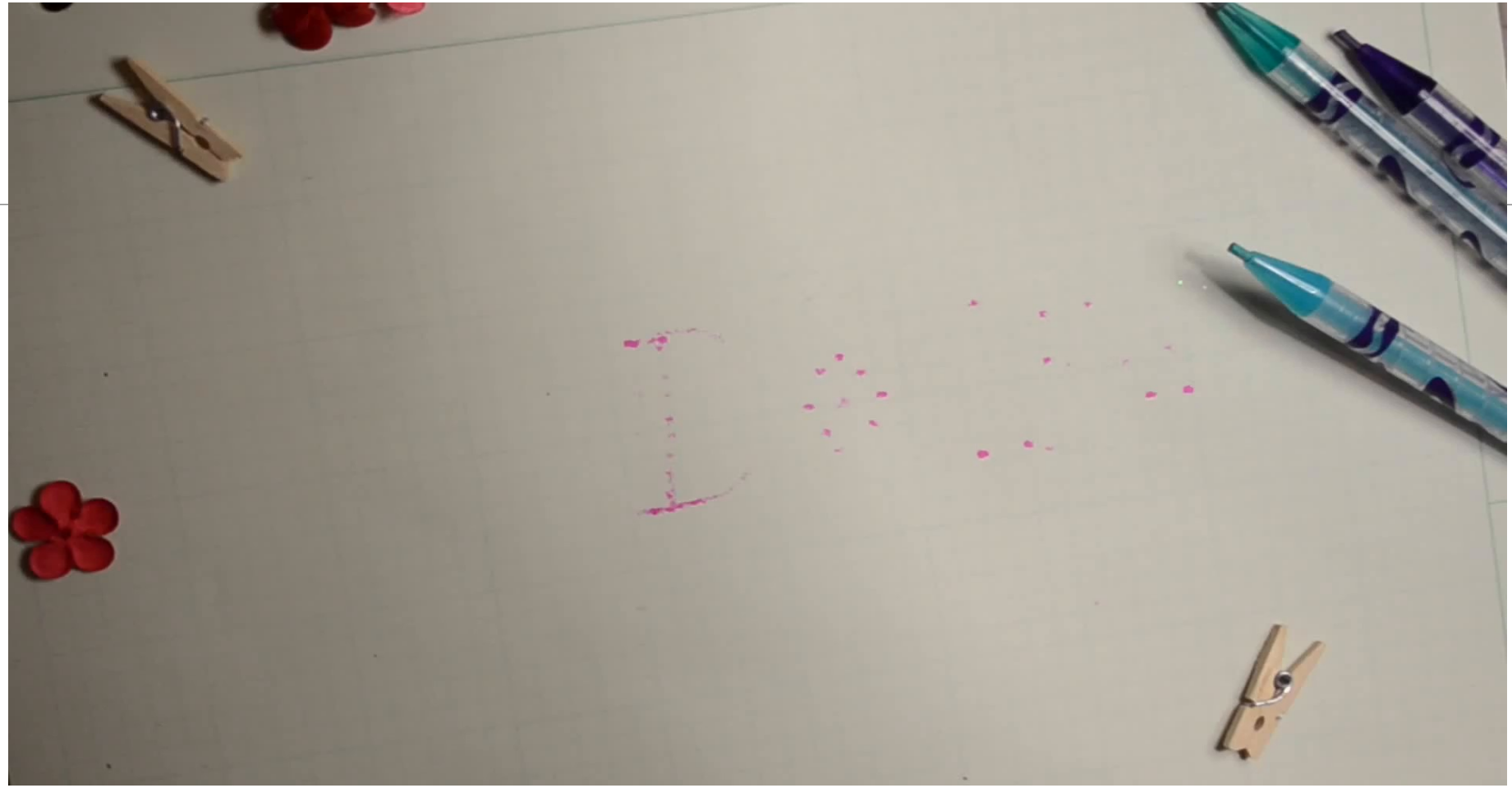
= 27

ARA*



$\epsilon = 2.5$ $\epsilon = 1.5$ $\epsilon = 1.0$

= 24



“Intro to Path Planning: D* Lite vs. A*” - CSMinute

Défauts hérités

- Moins performant que A* si trop de changement ou changement de but

expérience PerceptOR de Alonzo Kelly

- Réarrangement des nœuds trop long si trop inconsistents

Conclusion

Annexe

key(s)
01. return $[\min(g(s), rhs(s)) + h(s_{start}, s); \min(g(s), rhs(s))]$;

UpdateState(s)
02. if s was not visited before
03. $g(s) = \infty$;
04. if $(s \neq s_{goal})$ $rhs(s) = \min_{s' \in Succ(s)}(c(s, s') + g(s'))$;
05. if $(s \in OPEN)$ remove s from *OPEN*;
06. if $(g(s) \neq rhs(s))$ insert s into *OPEN* with key(s);

ComputeShortestPath()
07. while $(\min_{s \in OPEN}(\text{key}(s)) \dot{<} \text{key}(s_{start})$ OR $rhs(s_{start}) \neq g(s_{start}))$
08. remove state s with the minimum key from *OPEN*;
09. if $(g(s) > rhs(s))$
10. $g(s) = rhs(s)$;
11. for all $s' \in Pred(s)$ UpdateState(s');
12. else
13. $g(s) = \infty$;
14. for all $s' \in Pred(s) \cup \{s\}$ UpdateState(s');

Main()
15. $g(s_{start}) = rhs(s_{start}) = \infty$; $g(s_{goal}) = \infty$;
16. $rhs(s_{goal}) = 0$; $OPEN = \emptyset$;
17. insert s_{goal} into *OPEN*with key(s_{goal});
18. forever
19. ComputeShortestPath();
20. Wait for changes in edge costs;
21. for all directed edges (u, v) with changed edge costs
22. Update the edge cost $c(u, v)$;
23. UpdateState(u);

The D* Lite Algorithm (basic version).

key(s)
01. return $g(s) + \epsilon \cdot h(s_{start}, s)$;

ImprovePath()
02. while $(\min_{s \in OPEN}(\text{key}(s)) < \text{key}(s_{start}))$
03. remove s with the minimum key from *OPEN*;
04. $CLOSED = CLOSED \cup \{s\}$;
05. for all $s' \in Pred(s)$
06. if s' was not visited before
07. $g(s') = \infty$;
08. if $g(s') > c(s', s) + g(s)$
09. $g(s') = c(s', s) + g(s)$;
10. if $s' \notin CLOSED$
11. insert s' into *OPEN* with key(s');
12. else
13. insert s' into *INCONS*;

Main()
14. $g(s_{start}) = \infty$; $g(s_{goal}) = 0$;
15. $\epsilon = \epsilon_0$;
16. $OPEN = CLOSED = INCONS = \emptyset$;
17. insert s_{goal} into *OPEN* with key(s_{goal});
18. ImprovePath();
19. publish current ϵ-suboptimal solution;
20. while $\epsilon > 1$
21. decrease ϵ;
22. Move states from *INCONS* into *OPEN*;
23. Update the priorities for all $s \in OPEN$ according to key(s);
24. $CLOSED = \emptyset$;
25. ImprovePath();
26. publish current ϵ-suboptimal solution;

The ARA* Algorithm (backwards version).

Annexe

key(s)
01. if $(g(s) > rhs(s))$
02. return $[rhs(s) + \epsilon \cdot h(s_{start}, s); rhs(s)]$;
03. else
04. return $[g(s) + h(s_{start}, s); g(s)]$;

UpdateState(s)
05. if s was not visited before
06. $g(s) = \infty$;
07. if $(s \neq s_{goal})$ $rhs(s) = \min_{s' \in Succ(s)}(c(s, s') + g(s'))$;
08. if $(s \in OPEN)$ remove s from *OPEN*;
09. if $(g(s) \neq rhs(s))$
10. if $s \notin CLOSED$
11. insert s into *OPEN* with key(s);
12. else
13. insert s into *INCONS*;

ComputeorImprovePath()
14. while $(\min_{s \in OPEN}(\text{key}(s)) \dot{<} \text{key}(s_{start})$ OR $rhs(s_{start}) \neq g(s_{start}))$
15. remove state s with the minimum key from *OPEN*;
16. if $(g(s) > rhs(s))$
17. $g(s) = rhs(s)$;
18. $CLOSED = CLOSED \cup \{s\}$;
19. for all $s' \in Pred(s)$ UpdateState(s');
20. else
21. $g(s) = \infty$;
22. for all $s' \in Pred(s) \cup \{s\}$ UpdateState(s');

Main()
01. $g(s_{start}) = rhs(s_{start}) = \infty$; $g(s_{goal}) = \infty$;
02. $rhs(s_{goal}) = 0$; $\epsilon = \epsilon_0$;
03. $OPEN = CLOSED = INCONS = \emptyset$;
04. insert s_{goal} into *OPEN* with key(s_{goal});
05. ComputeorImprovePath();
06. publish current ϵ-suboptimal solution;
07. forever
08. if changes in edge costs are detected
09. for all directed edges (u, v) with changed edge costs
10. Update the edge cost $c(u, v)$;
11. UpdateState(u);
12. if significant edge cost changes were observed
13. increase ϵ or replan from scratch;
14. else if $\epsilon > 1$
15. decrease ϵ;
16. Move states from *INCONS* into *OPEN*;
17. Update the priorities for all $s \in OPEN$ according to key(s);
18. $CLOSED = \emptyset$;
19. ComputeorImprovePath();
20. publish current ϵ-suboptimal solution;
21. if $\epsilon = 1$
22. wait for changes in edge costs;

Anytime Dynamic A*: Main function.