

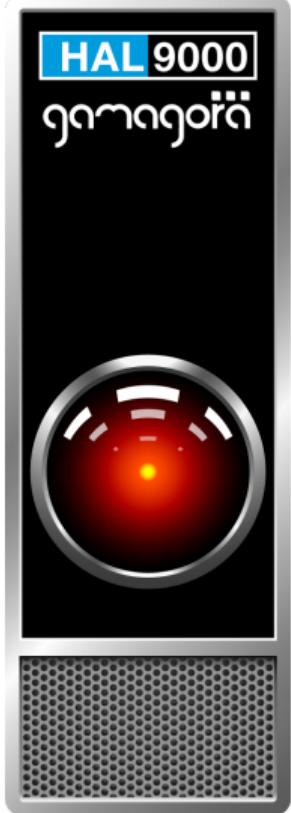
« Intelligence Artificielle pour les jeux vidéos

Didier Puzenat

didier.puzenat@univ-lyon2.fr

UE « systèmes complexes » du master 2 pro de Gamagora

UNIVERSITÉ
LUMIÈRE
LYON 2
UNIVERSITÉ DE LYON



Création d'une liste de diffusion



Merci de me faire un mail pour la création d'une liste !

didier.puzenat@univ-lyon2.fr

ou didier.puzenat@gmail.com

Pour recevoir :

- le support de cours,
- les fiches de TD-TP,
- des informations utiles, ...

Remarque : bien préciser vos **nom** et **prénom**

notamment si votre adresse mail est du genre supergeek69@gmail.com !

1

Définition de l'Intelligence Artificielle

- L'IA selon Wikipédia
- L'IA dans la littérature, le cinéma, et la recherche
- L'IA pour les jeux vidéos

2

Quelques algorithmes indispensables

- Recherche du plus court chemin : Dijkstra
- Recherche du plus court chemin : algorithme A*
- Flocking

3

Automates d'états fini

4

Réseaux de neurones artificielles

- Le neurone formel et le perceptron
- Apprentissage et Généralisation
- Le perceptron multi-couche et la rétro-propagation du gradient
- Application aux jeux vidéos, analyse du comportement, article de 2010

1

Définition de l'Intelligence Artificielle

- L'IA selon Wikipédia
- L'IA dans la littérature, le cinéma, et la recherche
- L'IA pour les jeux vidéos

2

Quelques algorithmes indispensables

- Recherche du plus court chemin : Dijkstra
- Recherche du plus court chemin : algorithme A*
- Flocking

3

Automates d'états fini

4

Réseaux de neurones artificielles

- Le neurone formel et le perceptron
- Apprentissage et Généralisation
- Le perceptron multi-couche et la rétro-propagation du gradient
- Application aux jeux vidéos, analyse du comportement, article de 2010

Définition de l'intelligence selon Wikipédia

Intelligence (« naturelle »)

L'intelligence est l'ensemble des facultés mentales permettant de comprendre les choses et les faits, de découvrir les relations entre elles et d'aboutir à la connaissance conceptuelle et rationnelle (par opposition à la sensation et à l'intuition). Elle permet de comprendre et de s'adapter à des situations nouvelles et peut en ce sens être également définie comme la faculté d'adaptation. L'intelligence peut être également perçue comme la capacité à traiter l'information pour atteindre ses objectifs.

Définition de l'intelligence selon Wikipédia

Intelligence (« naturelle »)

L'intelligence est l'ensemble des facultés mentales permettant de comprendre les choses et les faits, de découvrir les relations entre elles et d'aboutir à la connaissance conceptuelle et rationnelle (par opposition à la sensation et à l'intuition). Elle permet de comprendre et de s'adapter à des situations nouvelles et peut en ce sens être également définie comme la faculté d'adaptation. L'intelligence peut être également perçue comme la capacité à traiter l'information pour atteindre ses objectifs.

Définition de l'intelligence selon Wikipédia

Intelligence (« naturelle »)

L'intelligence est l'ensemble des facultés mentales permettant de comprendre les choses et les faits, de découvrir les relations entre elles et d'aboutir à la connaissance conceptuelle et rationnelle (par opposition à la sensation et à l'intuition). **Elle permet de comprendre et de s'adapter à des situations nouvelles et peut en ce sens être également définie comme la faculté d'adaptation.** L'intelligence peut être également perçue comme la capacité à traiter l'information pour atteindre ses objectifs.

Définition de l'intelligence selon Wikipédia

Intelligence (« naturelle »)

L'intelligence est l'ensemble des facultés mentales permettant de comprendre les choses et les faits, de découvrir les relations entre elles et d'aboutir à la connaissance conceptuelle et rationnelle (par opposition à la sensation et à l'intuition). Elle permet de comprendre et de s'adapter à des situations nouvelles et peut en ce sens être également définie comme la faculté d'adaptation. **L'intelligence peut être également perçue comme la capacité à traiter l'information pour atteindre ses objectifs.**

Définition de l'intelligence selon Wikipédia

Intelligence (« naturelle »)

L'intelligence est l'ensemble des facultés mentales permettant de comprendre les choses et les faits, de découvrir les relations entre elles et d'aboutir à la connaissance conceptuelle et rationnelle (par opposition à la sensation et à l'intuition). Elle permet de comprendre et de s'adapter à des situations nouvelles et peut en ce sens être également définie comme la faculté d'adaptation. L'intelligence peut être également perçue comme la capacité à traiter l'information pour atteindre ses objectifs.

Intelligence artificielle

L'IA est l'intelligence des machines et des logiciels. Elle se veut discipline scientifique recherchant des méthodes de création ou de simulation de l'intelligence.

1

Définition de l'Intelligence Artificielle

- L'IA selon Wikipédia
- L'IA dans la littérature, le cinéma, et la recherche
- L'IA pour les jeux vidéos

2

Quelques algorithmes indispensables

- Recherche du plus court chemin : Dijkstra
- Recherche du plus court chemin : algorithme A*
- Flocking

3

Automates d'états fini

4

Réseaux de neurones artificielles

- Le neurone formel et le perceptron
- Apprentissage et Généralisation
- Le perceptron multi-couche et la rétro-propagation du gradient
- Application aux jeux vidéos, analyse du comportement, article de 2010

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la [littérature et du cinéma](#)



Asimov : cerveau positronique, les lois de la robotique (pour le meilleur et le pire), etc.

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la [littérature et du cinéma](#)



Data : encore du positronique, guidé par la recherche d'une humanité (par exemple via les émotions)...

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la [littérature et du cinéma](#)



L'agent Smith : une IA purement logicielle, a priori sans émotion et avec un complexe de supériorité

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la [littérature et du cinéma](#)



TRON le simple programme, le MCP un programme qui a très « mal tourné » et Quorra une « IA » non programmée qui émerge de la complexité de son environnement

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la [littérature et du cinéma](#)



Jane (l'IA amie de Ender) : émerge également de la complexité, reconnue par Ender comme étant une nouvelle forme de vie «<consciente>

L'IA dans la littérature et au cinéma (et dans les séries)

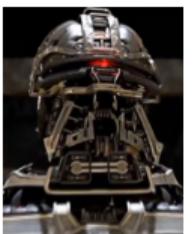
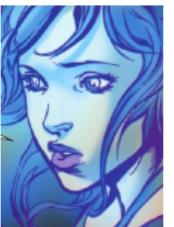
Quelques exemples d'IA dans la [littérature et du cinéma](#)



Caprica (l'ado, pas celle avec la robe rouge ;-): les premiers pas du transhumanisme

L'IA dans la littérature et au cinéma (et dans les séries)

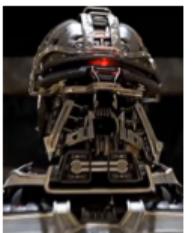
Quelques exemples d'IA dans la [littérature et du cinéma](#)



Terminator : le fantasme de l'IA qui prend le pouvoir (skynet) avec une armée de robots plus ou moins évolués (dont un T2 doué d'une certaine capacité d'apprentissage)

L'IA dans la littérature et au cinéma (et dans les séries)

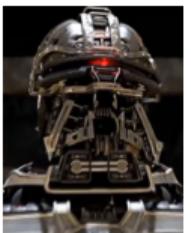
Quelques exemples d'IA dans la [littérature et du cinéma](#)



Z-6PO (aka C-3PO) : ne comprend rien aux humains mais efficace dans son domaine d'expertise (traducteur)

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la [littérature et du cinéma](#)



Hall 9000 : pour moi... le danger de l'anthropomorphisme (des humains) dans la relation IA-humain (et ça vaux pour 2016 !)

L'IA dans la littérature et au cinéma (et dans les séries)

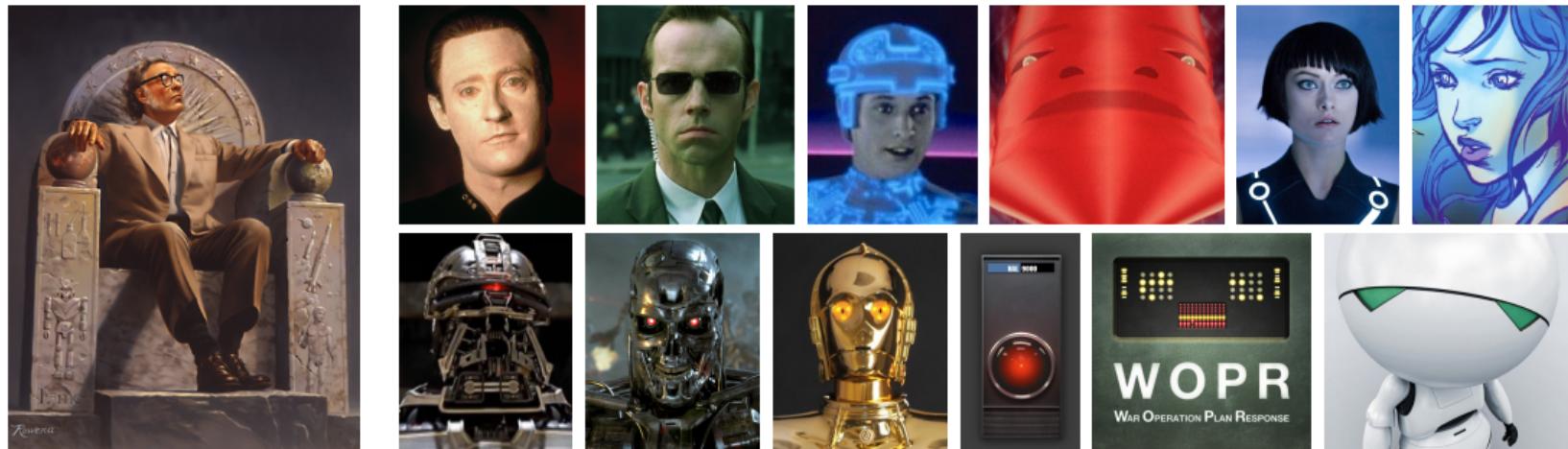
Quelques exemples d'IA dans la [littérature et du cinéma](#)



Le WOPR : il apprend très bien, pour lui le nombre de morts n'est qu'une variable, mais heureusement il préfère les échecs à la guerre thermonucléaire globale

L'IA dans la littérature et au cinéma (et dans les séries)

Quelques exemples d'IA dans la [littérature et du cinéma](#)



Marvin : l'IA fidèle mais dépressive car beaucoup trop intelligente pour être heureuse

L'IA dans le monde de la recherche

L'IA existe depuis les débuts de l'informatique, via la [cybernétique](#).



Le cerveau humain : un moyen et un modèle !

L'IA dans le monde de la recherche

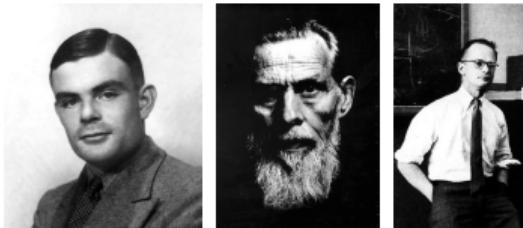
L'IA existe depuis les débuts de l'informatique, via la [cybernétique](#).



Alan Turing : un des pères de l'informatique, a contribué à l'IA par son « test de Turing »

L'IA dans le monde de la recherche

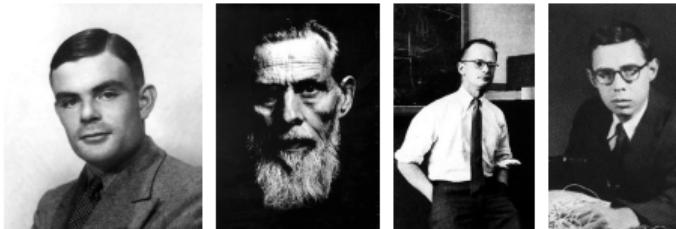
L'IA existe depuis les débuts de l'informatique, via la [cybernétique](#).



McCulloch et **Pitts** : premier modèle informatique du neurone biologique (années 40)

L'IA dans le monde de la recherche

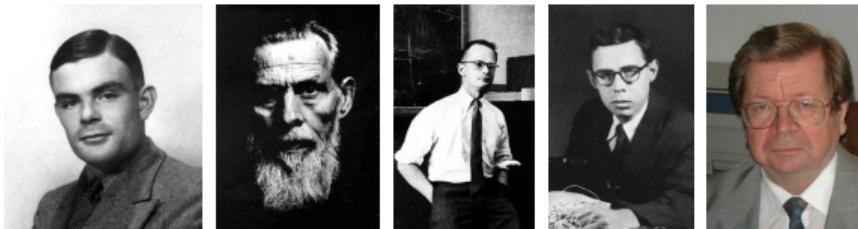
L'IA existe depuis les débuts de l'informatique, via la [cybernétique](#).



Frank Rosenblatt : perceptron (1957), modèle inspiré des théories cognitives

L'IA dans le monde de la recherche

L'IA existe depuis les débuts de l'informatique, via la [cybernétique](#).



Teuvo Kohonen : carte auto-adaptatives dites « cartes de Kohonen »

L'IA dans le monde de la recherche

L'IA existe depuis les débuts de l'informatique, via la [cybernétique](#).



Paul John Werbos : perceptron multi-couches et rétro-propagation du gradient

L'IA dans le monde de la recherche

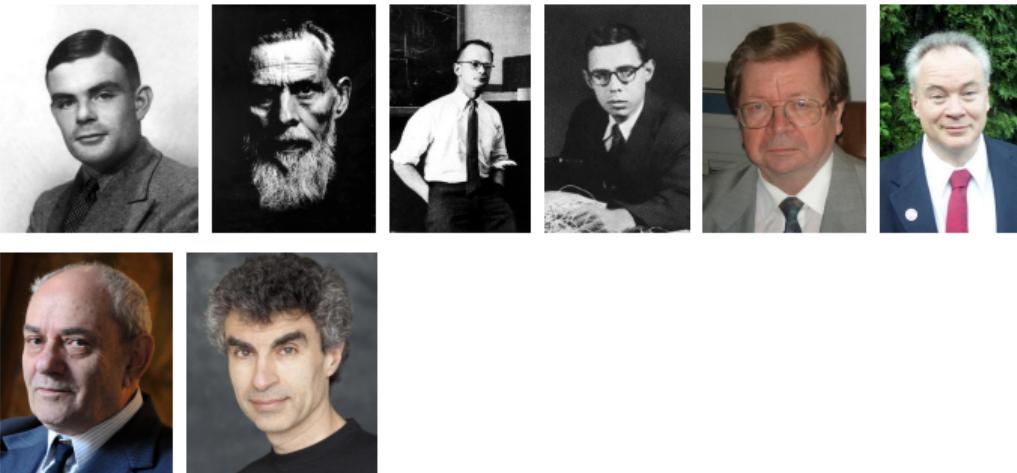
L'IA existe depuis les débuts de l'informatique, via la [cybernétique](#).



Vladimir Vapnik : théorie de l'apprentissage statistique, machines à support vectoriel

L'IA dans le monde de la recherche

L'IA existe depuis les débuts de l'informatique, via la [cybernétique](#).



Yoshua Bengio : le deep-learning ou «<apprentissage profond»

L'IA dans le monde de la recherche

L'IA existe depuis les débuts de l'informatique, via la [cybernétique](#).



Investissements massifs des géants du Net : **Google, Apple, Facebook, Amazon**, etc.

Les IA scientifiques et de la science fiction font peur

2014 : Stephen Hawking :

- « sous-estimer l'IA serait la plus grave erreur de notre histoire »
- « on peut imaginer que cette technologie déjoue les marchés financiers, dépasse les chercheurs humains, manipule les dirigeants humains et développe des armes qu'on ne peut pas même comprendre »
- « la meilleure ou la pire chose qui puisse arriver à l'humanité »

2016 : Pétition contre l'IA dans les armes signée par Stephen Hawking, Elon Musk (Tesla, Space X, etc.), Steve Wozniak (Apple), Noam Chomsky (linguiste), Demis Hassabis (fondateur de DeepMind), etc.

Mon avis : il faut relativiser, le Deep Learning ce n'est pas la révolution.

Et les politiques ?

Vladimir Poutine :



« le pays qui deviendra leader de ce secteur sera celui qui dominera le monde »

Et les politiques ?

Vladimir Poutine :



« le pays qui deviendra leader de ce secteur sera celui qui dominera le monde »



Et les politiques ?

Barack Obama :



« Historiquement, nous avons absorbé les nouvelles technologies, de nouveaux emplois ont été créés et notre niveau de vie s'est généralement amélioré. L'intelligence artificielle promet de créer une économie plus productive et efficace. Si elle est bien exploitée, cela peut générer énormément de prospérité et d'opportunités. »

1

Définition de l'Intelligence Artificielle

- L'IA selon Wikipédia
- L'IA dans la littérature, le cinéma, et la recherche
- L'IA pour les jeux vidéos

2

Quelques algorithmes indispensables

- Recherche du plus court chemin : Dijkstra
- Recherche du plus court chemin : algorithme A*
- Flocking

3

Automates d'états fini

4

Réseaux de neurones artificielles

- Le neurone formel et le perceptron
- Apprentissage et Généralisation
- Le perceptron multi-couche et la rétro-propagation du gradient
- Application aux jeux vidéos, analyse du comportement, article de 2010

Pourquoi ne pas utiliser l'état de l'art de l'IA dans les jeux vidéos ?



En 1996 Garry Kasparov perd une partie contre Deep-Blue (IBM), l'année suivante il perd un tournoi complet. Mais bon, les échecs seraient peut-être « trop facile » !

L'IA des géants du Net (GAFA) pour les jeux vidéo (?)

Mars 2016 la société DeepMind de Google lance **AlphaGo** qui gagne 4 à 1 le meilleur joueur mondial de GO : Lee Sedol.



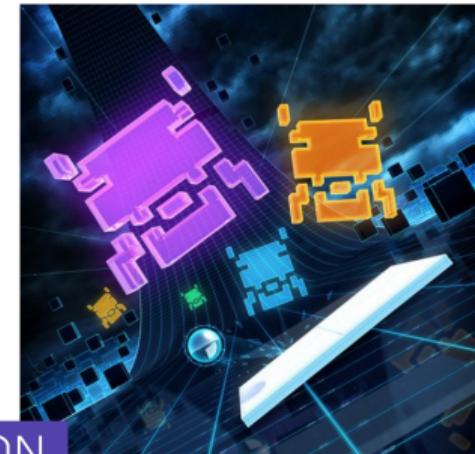
AlphaGo

The first computer program to ever beat a professional player at the game of Go.

L'IA des géants du Net (GAFA) pour les jeux vidéo (?)

Mars 2016 la société DeepMind de Google lance **AlphaGo** qui gagne 4 à 1 le meilleur joueur mondial de GO : Lee Sedol.

Projet actuel : l'algorithme « Deep Q-Network » aka **DQN**
« human-level control through Deep Reinforcement Learning »
→ jouer à la place du joueur à des jeux Atari 2600
(et avoir de supers scores)



Better than human-level control of classic Atari games through Deep Reinforcement Learning.

L'IA des géants du Net (GAFA) pour les jeux vidéo (?)

Mars 2016 la société DeepMind de Google lance **AlphaGo** qui gagne 4 à 1 le meilleur joueur mondial de GO : Lee Sedol.

Projet actuel : l'algorithme « Deep Q-Network » aka **DQN**
« human-level control through Deep Reinforcement Learning »
→ jouer à la place du joueur à des jeux Atari 2600
(et avoir de supers scores)



Rumeur : prochaine cible de DeepMind : **StarCraft !**

DeepMind comme adversaire dans les jeux du commerce ?

Alors pourquoi ne pas utiliser DeepMind dans nos jeux ?

DeepMind comme adversaire dans les jeux du commerce ?

Alors pourquoi ne pas utiliser DeepMind dans nos jeux ?

- Ça coûte cher à développer,
formation de pointe, puissance de calcul, temps de développement, risque, etc.

DeepMind comme adversaire dans les jeux du commerce ?

Alors pourquoi ne pas utiliser DeepMind dans nos jeux ?

- Ça coûte cher à développer,
formation de pointe, puissance de calcul, temps de développement, risque, etc.
(mais pas forcément gourmand en puissance une fois dans le jeu).

DeepMind comme adversaire dans les jeux du commerce ?

Alors pourquoi ne pas utiliser DeepMind dans nos jeux ?

- Ça coûte cher à développer,
formation de pointe, puissance de calcul, temps de développement, risque, etc.
(mais pas forcément gourmand en puissance une fois dans le jeu).
- Le joueur ne s'en rendrait peut-être même pas compte... .
- Ce n'est peut-être même pas ce que recherche le joueur.

DeepMind comme adversaire dans les jeux du commerce ?

Alors pourquoi ne pas utiliser DeepMind dans nos jeux ?

- Ça coûte cher à développer,
formation de pointe, puissance de calcul, temps de développement, risque, etc.
(mais pas forcément gourmand en puissance une fois dans le jeu).
- Le joueur ne s'en rendrait peut-être même pas compte... .
- Ce n'est peut-être même pas ce que recherche le joueur.

Conclusions :

- ① un jour peut-être, mais pas tout de suite
- ② en complément de techniques moins gourmandes voire simplistes
- ③ peut ouvrir de nouvelles perspectives, eg des dialogues enfin un peu moins pauvres.

Spécificité de l'IA dans les jeux vidéos

Dans un jeu il y a des contraintes fortes :

- la jouabilité,
- ressources restreintes (mémoire, temps de développement, temps de calcul, etc.)

⇒ **Il objectif de l'IA dans les jeux vidéos est de donner
l'illusion d'un comportement intelligent**

Spécificité de l'IA dans les jeux vidéos

Dans un jeu il y a des contraintes fortes :

- la jouabilité,
- ressources restreintes (mémoire, temps de développement, temps de calcul, etc.)

⇒ **I'objectif de l'IA dans les jeux vidéos est de donner
l'illusion d'un comportement intelligent**

Tous les moyens sont bons pour que cette illusion soit réaliste, y compris tricher :

- information sur l'état du jour, par exemple sa position,
- information sur l'environnement,
- ressources (eg munitions), etc.

Exemple de domaines d'application de l'IA dans les jeux vidéos

- **Le déplacement** des « personnages »,
qui doit sembler au moins naturel même dans un monde discret (pixels),
inclus : **le suivi, l'évitement, le comportement en groupe,**
voire des algos plus pointus comme la **recherche de chemin**.

Exemple de domaines d'application de l'IA dans les jeux vidéos

- Le déplacement des « personnages »,
qui doit sembler au moins naturel même dans un monde discret (pixels),
inclus : le suivi, l'évitement, le comportement en groupe,
voire des algos plus pointus comme la recherche de chemin.
- Les **interactions** personnage(s)–joueur(s) et personnage(s)–personnage(s),
inclus la **gestion des champs perceptifs** (visuel, auditif, etc.),
voire les **dialogues**...

Exemple de domaines d'application de l'IA dans les jeux vidéos

- Le déplacement des « personnages »,
qui doit sembler au moins naturel même dans un monde discret (pixels),
inclus : le suivi, l'évitement, le comportement en groupe,
voire des algos plus pointus comme la recherche de chemin.
- Les interactions personnage(s)–joueur(s) et personnage(s)–personnage(s),
inclus la gestion des champs perceptifs (visuel, auditif, etc.),
voire les dialogues... .
- **l'animation** 2D ou 3D (enfin 4D ;-).

Exemple de domaines d'application de l'IA dans les jeux vidéos

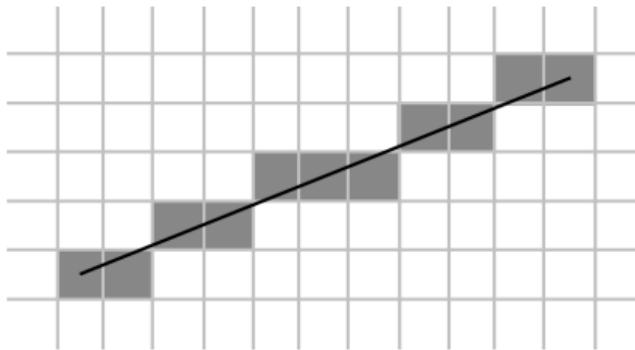
- Le déplacement des « personnages »,
qui doit sembler au moins naturel même dans un monde discret (pixels),
inclus : le suivi, l'évitement, le comportement en groupe,
voire des algos plus pointus comme la recherche de chemin.
- Les interactions personnage(s)–joueur(s) et personnage(s)–personnage(s),
inclus la gestion des champs perceptifs (visuel, auditif, etc.),
voire les dialogues... .
- l'animation 2D ou 3D (enfin 4D ;-).

Mais aussi le cas échéant : la **création de l'environnement**, une **stratégie globale**, etc.

Exemple de techniques informatiques utilisées pour l'IA dans les jeux

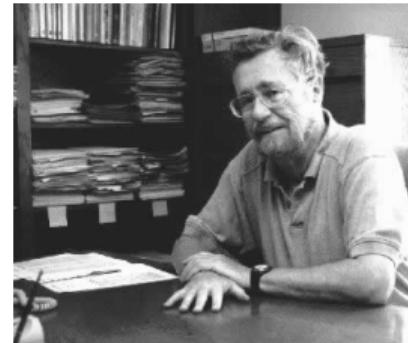
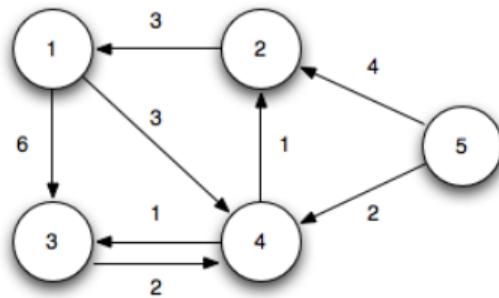
- **techniques d'image,**

par exemple l'algorithme de tracé de segment de Bresenham (publié en 1963) pour les déplacements dans un univers discret (pavés ou pixels) :



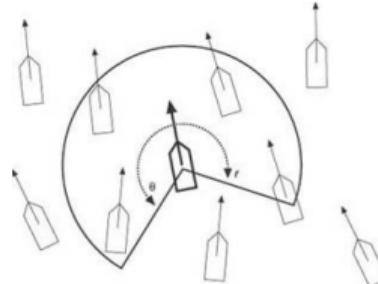
Exemple de techniques informatiques utilisées pour l'IA dans les jeux

- techniques d'image,
par exemple l'algorithme de tracé de segment de Bresenham (publié en 1963)
- **technique de recherche opérationnelle**
par exemple d'algorithme de Dijkstra (publié en 1959)



Exemple de techniques informatiques utilisées pour l'IA dans les jeux

- techniques d'image,
par exemple l'algorithme de tracé de segment de Bresenham (publié en 1963)
- technique de recherche opérationnelle
par exemple d'algorithme de Dijkstra (publié en 1959)
- **étude du comportement** animal ou humain, exemple : agrégation ou « flocking »
(simulé sur ordinateur par Craig Reynolds (1987))



Exemple de techniques informatiques utilisées pour l'IA dans les jeux

- techniques d'image,
par exemple l'algorithme de tracé de segment de Bresenham (publié en 1963)
- technique de recherche opérationnelle
par exemple d'algorithme de Dijkstra (publié en 1959)
- étude du comportement animal ou humain, exemple : agrégation ou « flocking »
(simulé sur ordinateur par Craig Reynolds (1987))
- application des lois de la physique voire de **modèles physiques**

Exemple de techniques informatiques utilisées pour l'IA dans les jeux

- techniques d'image,
par exemple l'algorithme de tracé de segment de Bresenham (publié en 1963)
- technique de recherche opérationnelle
par exemple d'algorithme de Dijkstra (publié en 1959)
- étude du comportement animal ou humain, exemple : agrégation ou « flocking »
(simulé sur ordinateur par Craig Reynolds (1987))
- application des lois de la physique voire de modèles physiques
- IA « if-then » : if-then-else, automates, scripts
- IA symbolique (rare) : systèmes experts, logique floue, systèmes multi-agents, ...
- IA numérique (très rare) : réseaux de neurones artificiels

1

Définition de l'Intelligence Artificielle

- L'IA selon Wikipédia
- L'IA dans la littérature, le cinéma, et la recherche
- L'IA pour les jeux vidéos

2

Quelques algorithmes indispensables

- Recherche du plus court chemin : Dijkstra
- Recherche du plus court chemin : algorithme A*
- Flocking

3

Automates d'états fini

4

Réseaux de neurones artificielles

- Le neurone formel et le perceptron
- Apprentissage et Généralisation
- Le perceptron multi-couche et la rétro-propagation du gradient
- Application aux jeux vidéos, analyse du comportement, article de 2010

1

Définition de l'Intelligence Artificielle

- L'IA selon Wikipédia
- L'IA dans la littérature, le cinéma, et la recherche
- L'IA pour les jeux vidéos

2

Quelques algorithmes indispensables

- Recherche du plus court chemin : Dijkstra
- Recherche du plus court chemin : algorithme A*
- Flocking

3

Automates d'états fini

4

Réseaux de neurones artificielles

- Le neurone formel et le perceptron
- Apprentissage et Généralisation
- Le perceptron multi-couche et la rétro-propagation du gradient
- Application aux jeux vidéos, analyse du comportement, article de 2010

Recherche du plus court chemin : Dijkstra

Objectif : **plus court chemin** dans un graphe orienté pondéré par des réels positifs

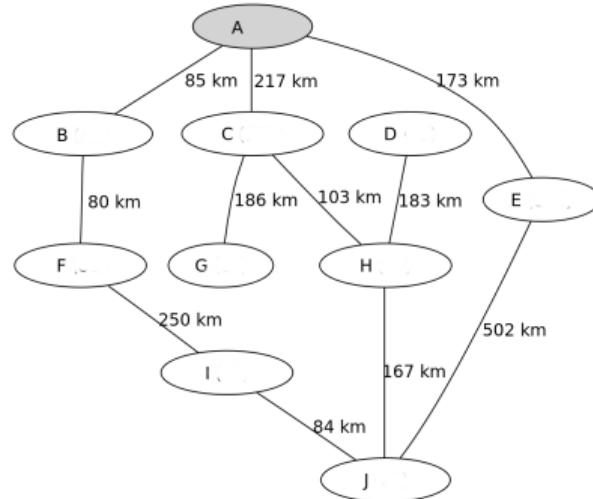
Principe : on construit progressivement un sous-graphe dans lequel sont classés les sommets par ordre croissant de leur distance minimale au sommet de départ

À chaque itération :

- ① on choisit en dehors du sous-graphe un sommet de distance minimale et on l'ajoute au sous-graphe
- ② on met à jour les distances des sommets voisins de celui ajouté
- ③ on continue jusqu'à épuisement des sommets (ou jusqu'à l'arrivée)

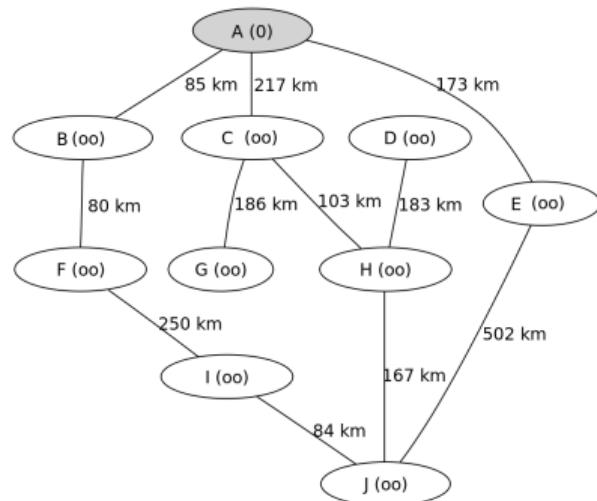
Exemple d'utilisation de Dijkstra dans un graphe non-orienté

On va calculer le **plus court trajet pour aller de la ville A à la ville J** :



Exemple d'utilisation de Dijkstra dans un graphe non-orienté

On va calculer le **plus court trajet pour aller de la ville A à la ville J** :

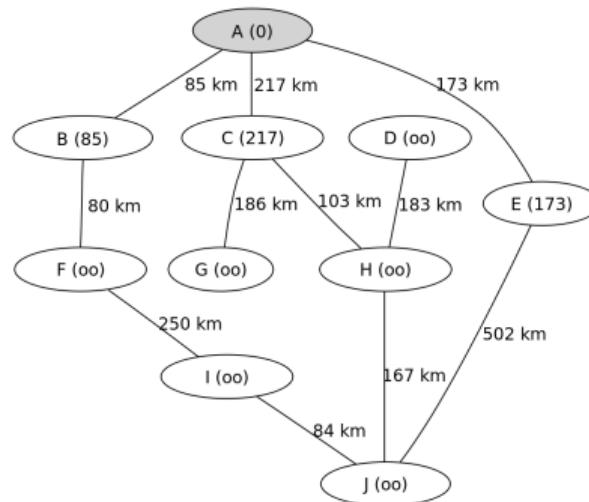


Étape 0 (initialisation) :

- on considère que les distances de chaque sommet au sommet de départ sont infinies sauf pour le sommet de départ pour lequel la distance est de 0
- le sous-graphe de départ est l'ensemble vide

Exemple d'utilisation de Dijkstra dans un graphe non-orienté

On va calculer le **plus court trajet pour aller de la ville A à la ville J** :



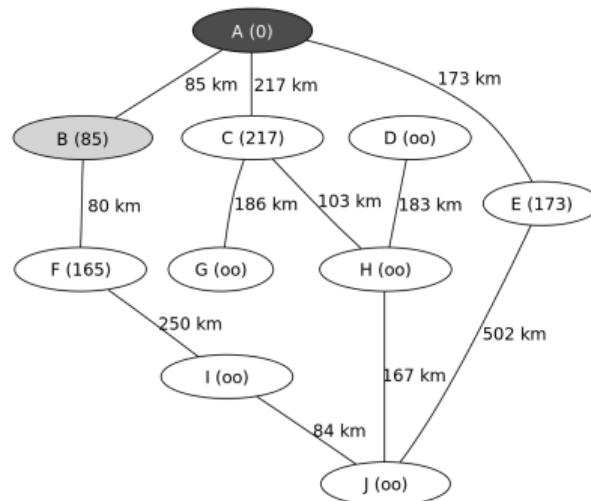
Étape 1 :

- ➊ on choisit la ville A
- ➋ on met à jour les villes voisines de A

⇒ Les distances de B, C et E deviennent 85, 217, 173

Exemple d'utilisation de Dijkstra dans un graphe non-orienté

On va calculer le **plus court trajet pour aller de la ville A à la ville J** :



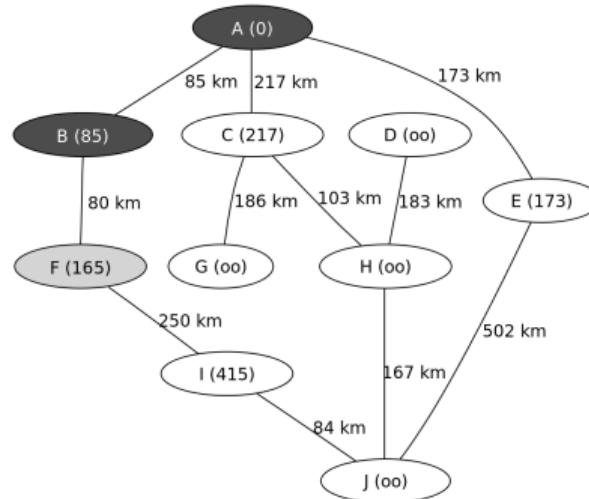
Étape 2 :

- ① on choisit la ville B car c'est la ville hors du sous-graphe qui est à la distance minimale (85)
- ② on met à jour le seul voisin (F)

⇒ La distance de F devient $85+80 = 165$

Exemple d'utilisation de Dijkstra dans un graphe non-orienté

On va calculer le **plus court trajet pour aller de la ville A à la ville J** :

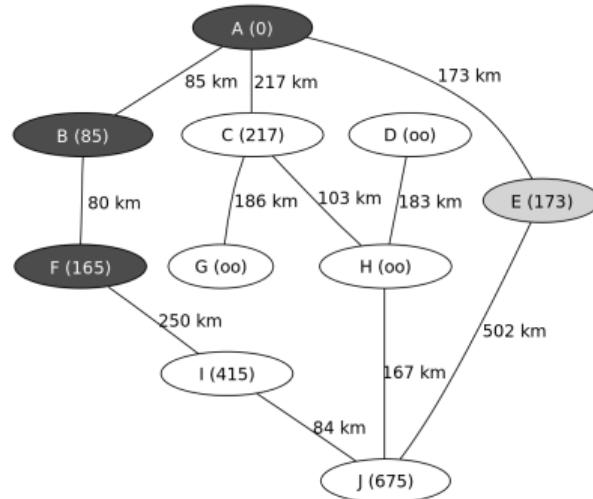


Étape 3 :

- ➊ on choisit F
- ➋ on met à jour le voisin I (415)

Exemple d'utilisation de Dijkstra dans un graphe non-orienté

On va calculer le **plus court trajet pour aller de la ville A à la ville J** :

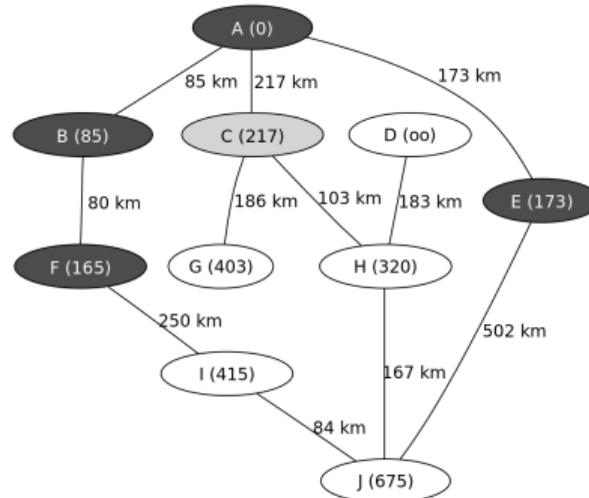


Étape 4 :

- ➊ on choisit E
- ➋ on met à jour le voisin J (675)

Exemple d'utilisation de Dijkstra dans un graphe non-orienté

On va calculer le **plus court trajet pour aller de la ville A à la ville J** :



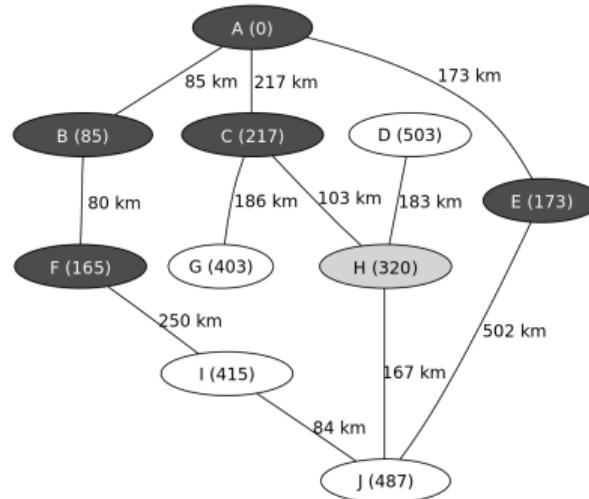
Étape 5 :

La distance la plus courte en dehors du sous-graphe est maintenant celle de la ville C.

- ➊ on choisit donc C
- ➋ on met à jour la ville G (403) et la ville H (320)

Exemple d'utilisation de Dijkstra dans un graphe non-orienté

On va calculer le **plus court trajet pour aller de la ville A à la ville J** :



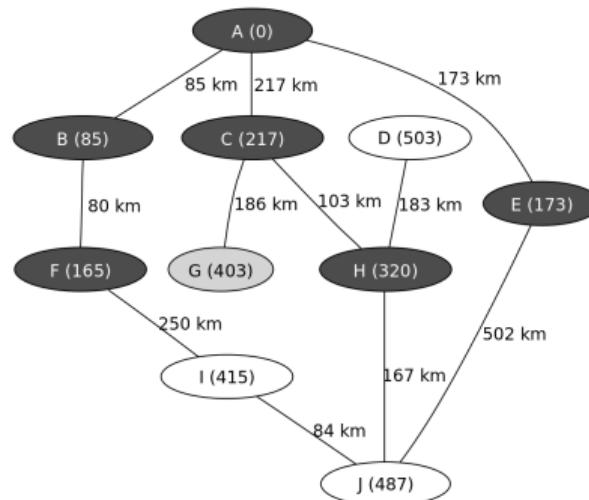
Étape 6 :

La distance la plus courte en dehors du sous-graphe est maintenant celle de H (320).

- ➊ on choisit donc H
- ➋ on met à jour la ville D (503) et la ville J ($487 < 675$)

Exemple d'utilisation de Dijkstra dans un graphe non-orienté

On va calculer le **plus court trajet pour aller de la ville A à la ville J** :



Étape 7 :

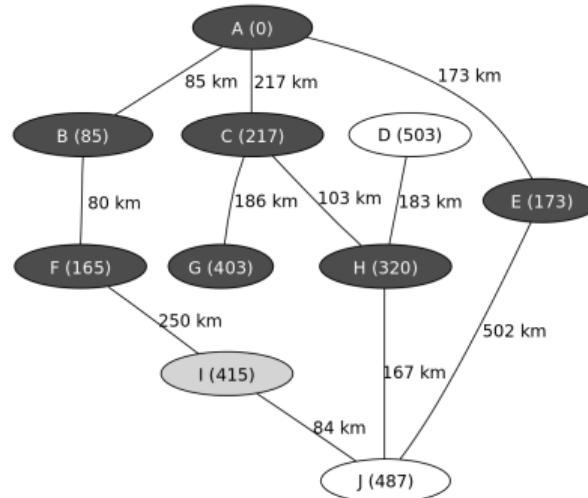
La distance la plus courte suivante est celle de la ville G.

- 1 On choisit G

La mise à jour ne change aucune autre distance.

Exemple d'utilisation de Dijkstra dans un graphe non-orienté

On va calculer le **plus court trajet pour aller de la ville A à la ville J** :



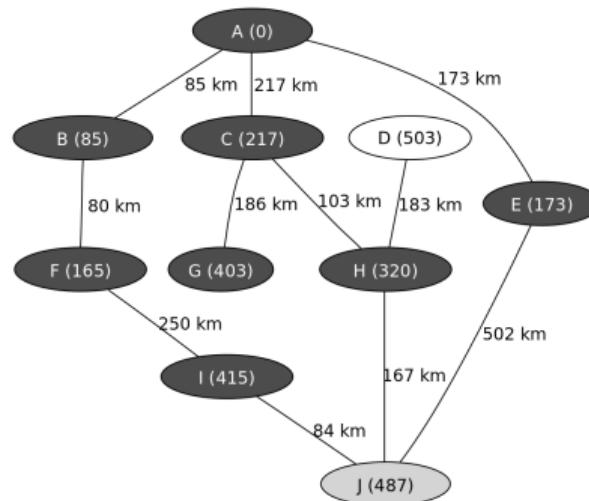
Étape 8 :

La distance la plus courte suivante est celle de la ville I.

La distance de la ville voisine J n'est pas modifiée car la distance existante est inférieure à celle que l'on obtiendrait en passant par I ($415 + 84 > 487$).

Exemple d'utilisation de Dijkstra dans un graphe non-orienté

On va calculer le **plus court trajet pour aller de la ville A à la ville J** :



Étape 9 :

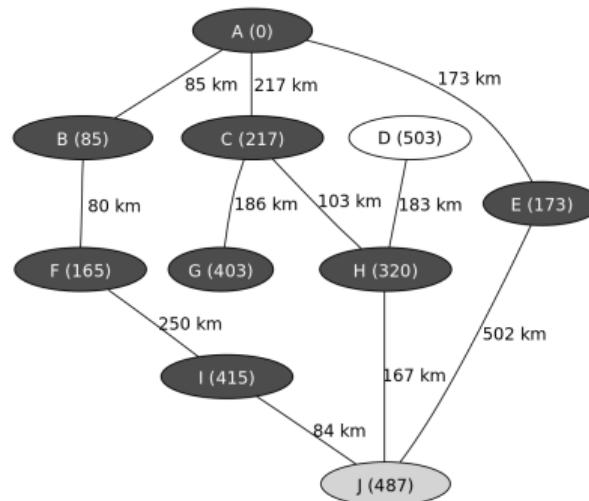
La ville dont la distance est la plus courte est J (487).

- ① on choisit J et on l'ajoute au sous-graphe
- ② on s'arrête puisque la ville d'arrivée est maintenant dans le sous-graphe

⇒ **le plus court chemin de A à J fait 487 km**

Exemple d'utilisation de Dijkstra dans un graphe non-orienté

On va calculer le **plus court trajet pour aller de la ville A à la ville J** :



Étape 10 : recherche du chemin

Pour expliciter le chemin il faut avoir mémorisé d'où on venait à chaque fois qu'on a changé la distance d'une ville,

dès lors, arrivé à J, il suffit de rebrousser chemin pour trouver H, C et enfin A !

⇒ **le plus court chemin est A → C → H → J**

Questions

Dans le cas d'un trajet en voiture, comment faire pour prendre en compte

- le **type de voie** (autoroute, route, chemin, désert, jungle, ...) ?
- le **type de véhicule**, etc.



Questions

Dans le cas d'un trajet en voiture, comment faire pour prendre en compte

- le **type de voie** (autoroute, route, chemin, désert, jungle, ...) ?
- le **type de véhicule**, etc.



Idées de solution : – valuer avec le temps plutôt que la distance,
– coefficients (type de véhicule, météo, fatigue, blessure, etc.)

À faire... et à rendre !

- ➊ Faire tourner l'algorithme en le programmant en C, C++, C#, Java, Python (au choix, pas tous !)

À faire... et à rendre !

- ① Faire tourner l'algorithme en le programmant en C, C++, C#, Java, Python (au choix, pas tous !)
- ② Essayez le graphe des transparents précédents
- ③ Essayez avec un très gros graphe
- ④ Essayez avec un graphe orienté

À faire... et à rendre!

- ① Faire tourner l'algorithme en le programmant en C, C++, C#, Java, Python (au choix, pas tous !)
- ② Essayez le graphe des transparents précédents
- ③ Essayez avec un très gros graphe
- ④ Essayez avec un graphe orienté
- ⑤ Introduire une idée personnelle dans le cadre d'un jeu

À faire... et à rendre!

- ① Faire tourner l'algorithme en le programmant en C, C++, C#, Java, Python (au choix, pas tous !)
- ② Essayez le graphe des transparents précédents
- ③ Essayez avec un très gros graphe
- ④ Essayez avec un graphe orienté
- ⑤ Introduire une idée personnelle dans le cadre d'un jeu

Remarques : – cette version de l'algorithme de fonctionne pas pour le voyage dans le temps car les valuations des arcs doivent être positives...
– dans un jeu la destination peut être un personnage qui se déplace
– dans un jeu le graphe peut changer dans le temps
– ...

1

Définition de l'Intelligence Artificielle

- L'IA selon Wikipédia
- L'IA dans la littérature, le cinéma, et la recherche
- L'IA pour les jeux vidéos

2

Quelques algorithmes indispensables

- Recherche du plus court chemin : Dijkstra
- **Recherche du plus court chemin : algorithme A***
- Flocking

3

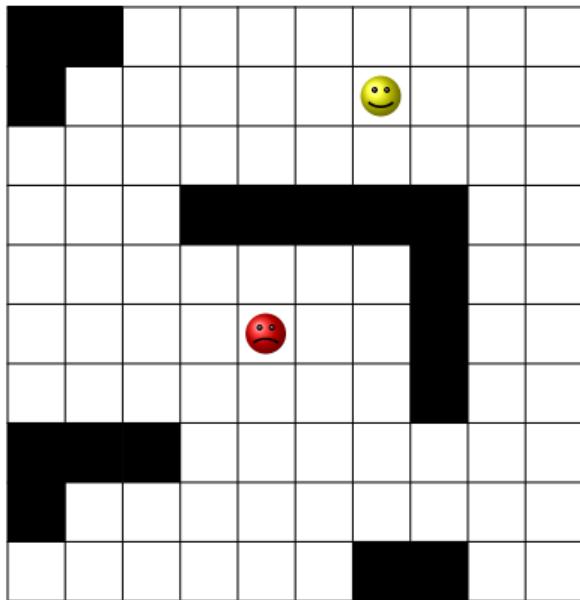
Automates d'états fini

4

Réseaux de neurones artificielles

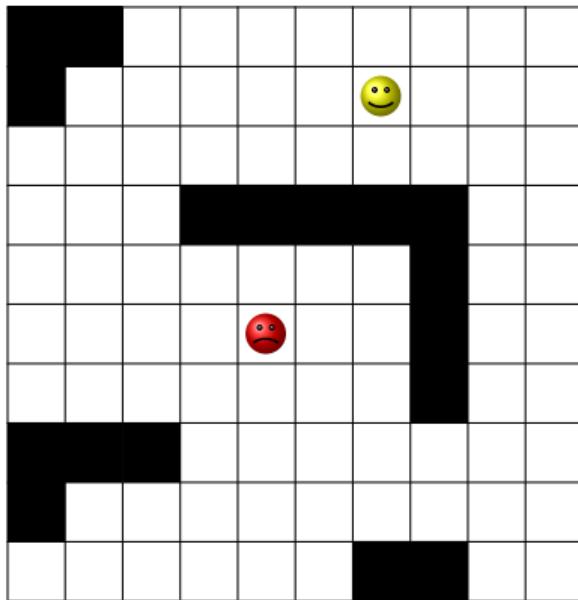
- Le neurone formel et le perceptron
- Apprentissage et Généralisation
- Le perceptron multi-couche et la rétro-propagation du gradient
- Application aux jeux vidéos, analyse du comportement, article de 2010

Algorithme A*



L'IA 😟 veut attaquer le joueur 😊
mais quel chemin 😟 doit-elle suivre ?

Algorithme A*



L'IA 😟 veut attaquer le joueur 😊
mais quel chemin 😟 doit-elle suivre ?

Problèmes :

- trouver le plus court chemin
- éviter les obstacles
- ne pas consommer trop de ressources (CPU)

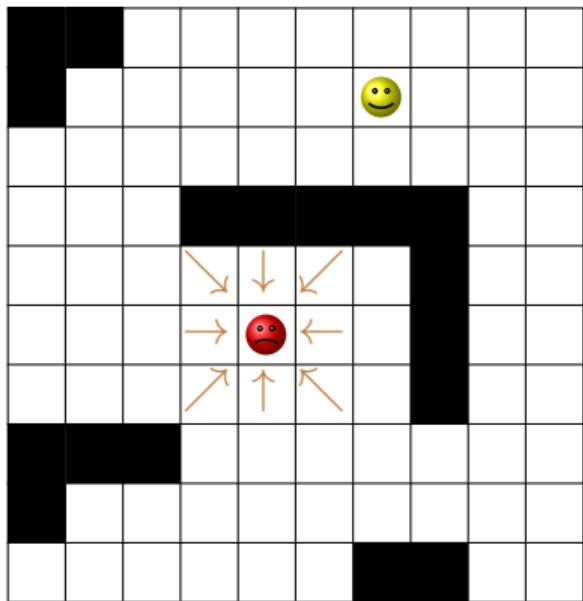
⇒ utilisation de l'algorithme « A* » !

Algorithme A* (détail)

- ① ajouter le point de départ à la « liste ouverte » en lui attribuant un « coût » nul
- ② tant que la liste ouverte n'est pas vide :
 - ① considérer la position **X** de la liste ouverte avec le coût minimum
si  ∈ **X** alors c'est terminé (sortir)
 - ② déplacer **X** de la liste ouverte à la « liste fermée »
 - ③ pour toutes les positions **Y** voisines (adjacentes) de **X**
si **Y** n'est ni dans la liste ouverte ni dans la liste fermée ni un obstacle
ajouter **Y** dans la liste ouverte et calculer son coût

Remarques : – la liste ouverte contient les positions (encore) à analyser
– la liste fermée contient les positions qui n'ont plus à être analysées
ie les positions dont toutes les voisines ont été vérifiées

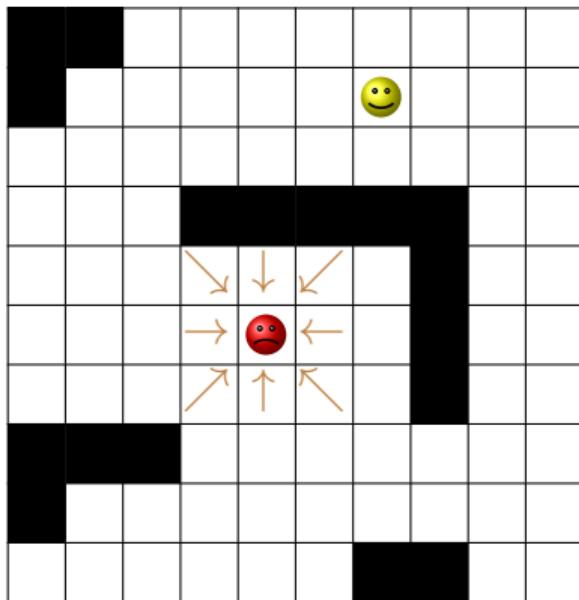
Algorithme A* (penser à semer des caillons pour retrouver son chemin)



Attention : il faut conserver la trace du trajet

- ⇒ lorsqu'on ajoute **Y** à la **liste ouverte**
on garde la trace de la position « d'où on vient » (**X**)
- ⇒ il sera possible de remonter jusqu'à **?** depuis **!**

Algorithme A* (penser à semer des caillons pour retrouver son chemin)

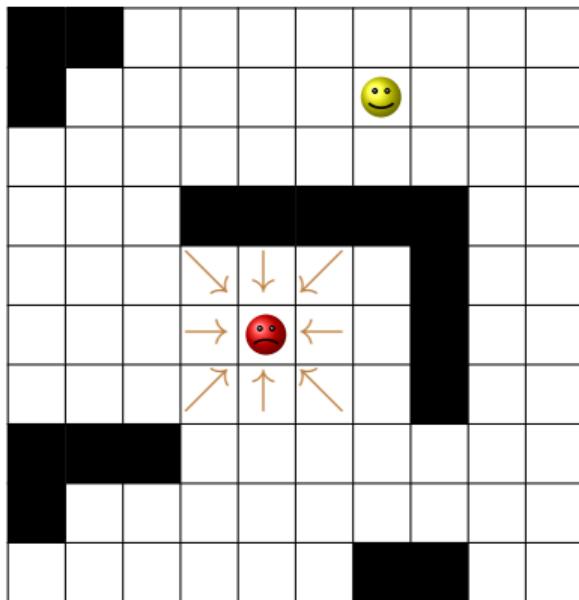


Attention : il faut conserver la trace du trajet

- ⇒ lorsqu'on ajoute **Y** à la **liste ouverte**
on garde la trace de la position « d'où on vient » (**X**)
- ⇒ il sera possible de remonter jusqu'à **?** depuis **!**

Mais comment calculer le coût d'une position ?

Algorithme A* (penser à semer des caillons pour retrouver son chemin)



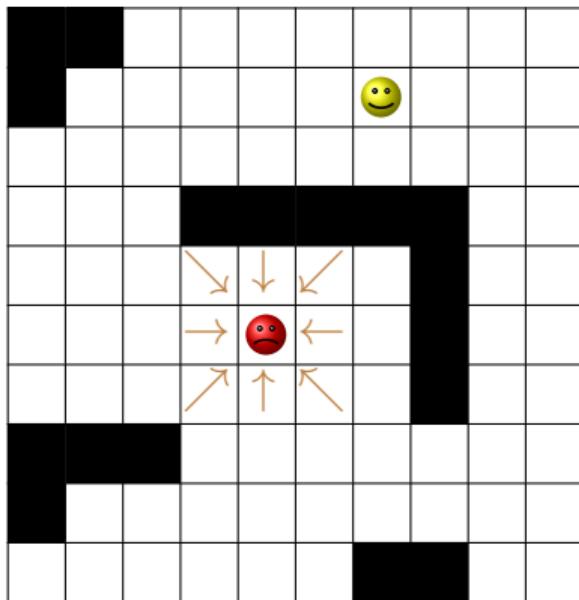
Attention : il faut conserver la trace du trajet

- ⇒ lorsqu'on ajoute **Y** à la **liste ouverte**
on garde la trace de la position « d'où on vient » (**X**)
- ⇒ il sera possible de remonter jusqu'à **?** depuis **!**

Mais comment calculer le coût d'une position ?

coût(**Y**) = **distance depuis** **?** (**déplacements**)
+ **estimation de la distance jusqu'à** **!**

Algorithme A* (penser à semer des caillons pour retrouver son chemin)



Attention : il faut conserver la trace du trajet

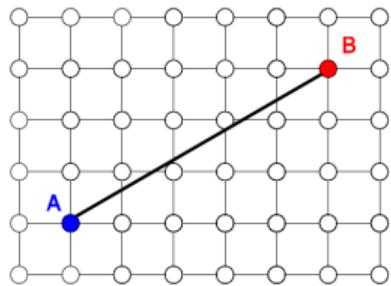
- ⇒ lorsqu'on ajoute **Y** à la **liste ouverte**
on garde la trace de la position « d'où on vient » (**X**)
- ⇒ il sera possible de remonter jusqu'à **?** depuis **!**

Mais comment calculer le coût d'une position ?

coût(**Y**) = distance depuis **?** (déplacements)
+ estimation de la distance jusqu'à **!**
→ sans prendre en compte les obstacles !

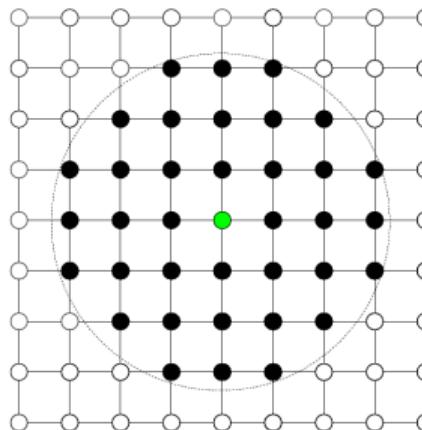
Calcul d'une distance : distance euclidienne

Facile à calculer



$$d_E(A, B) = \sqrt{5^2 + 3^2} = \sqrt{34}$$

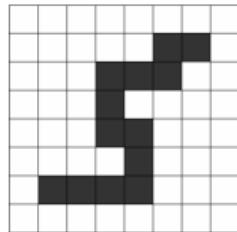
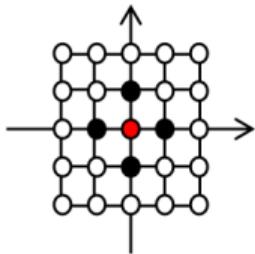
mais pas très pratique dans un monde discret :



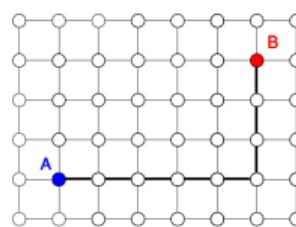
$$B_{\sqrt{10}}(C) = \left\{ z \in \mathbf{Z}^2 ; d_E(z, c) \leq \sqrt{10} \right\}$$

Calcul d'une distance : distance discrète avec 4 voisins

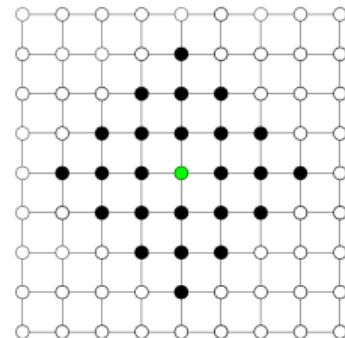
Facile à calculer
mais pas très « naturelle »



$$d_4(A, B) = |x_A - x_B| + |y_A - y_B|$$



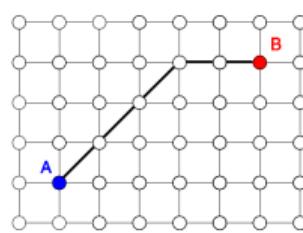
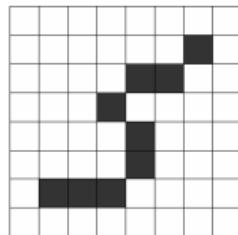
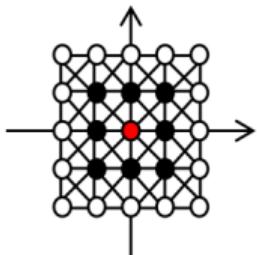
$$d_4(A, B) = 5 + 3 = 8$$



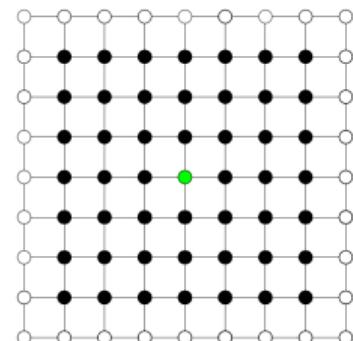
$$B_3^{d_4}(C) = \{z \in \mathbf{Z}^2; d_4(z, c) \leq 3\}$$

Calcul d'une distance : distance discrète avec 8 voisins

À peine plus lourde en calcul
et davantage « fluide »

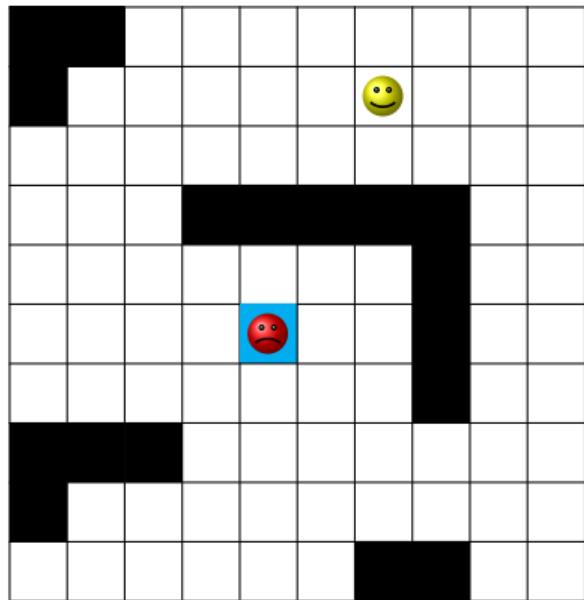


$$d_8(A, B) = \max(5, 5) = 5$$



$$B_3^{d_8}(C) = \{z \in \mathbb{Z}^2; d_8(z, c) \leq 3\}$$

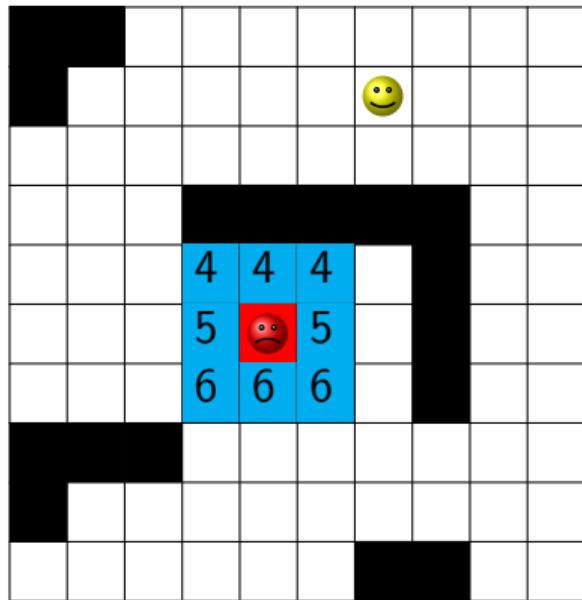
Algorithme A* (déroulé – étape 1)



Déroulons l'algorithme :

- ➊ liste ouverte = [😞]
- ➋ liste fermée = []

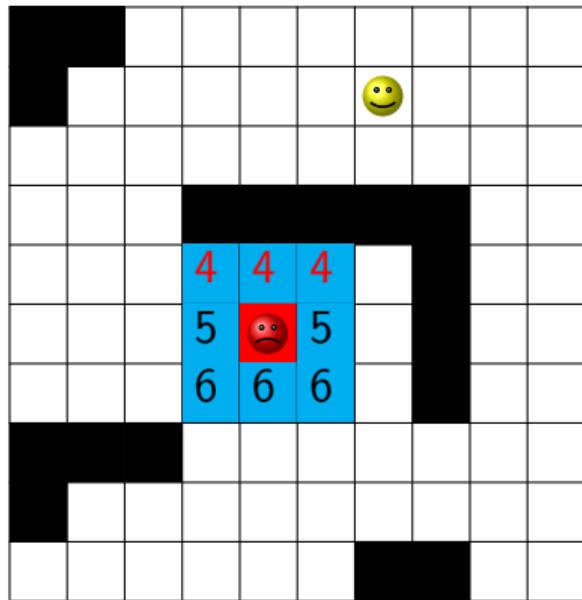
Algorithme A* (déroulé – étape 2)



Déroulons l'algorithme :

- ➊ liste ouverte = []
- ➋ liste fermée = []

Algorithme A* (déroulé – étape 2)



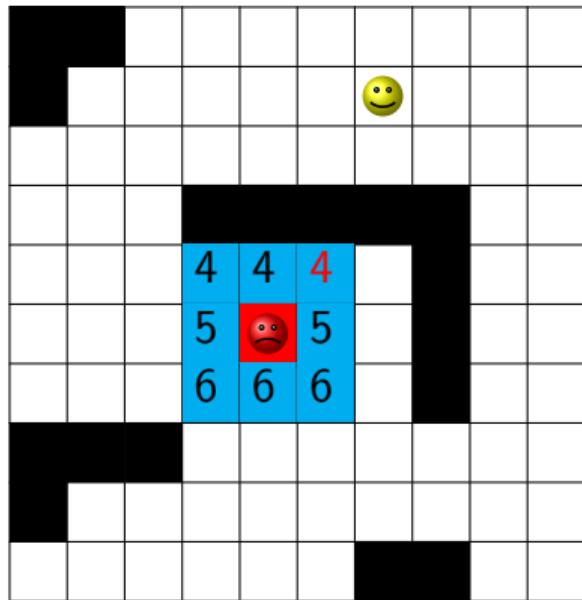
Déroulons l'algorithme :

- ➊ liste ouverte = []
- ➋ liste fermée = []

Choix du nouveau :

on a 3 positions avec le même coût

Algorithme A* (déroulé – étape 2)



Déroulons l'algorithme :

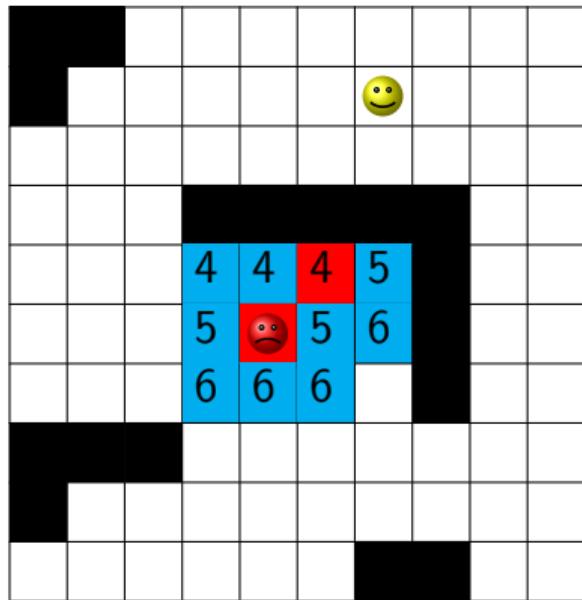
- ➊ liste ouverte = []
- ➋ liste fermée = []

Choix du nouveau :

on a 3 positions avec le même coût

⇒ on en choisit une au hasard (pas grave)

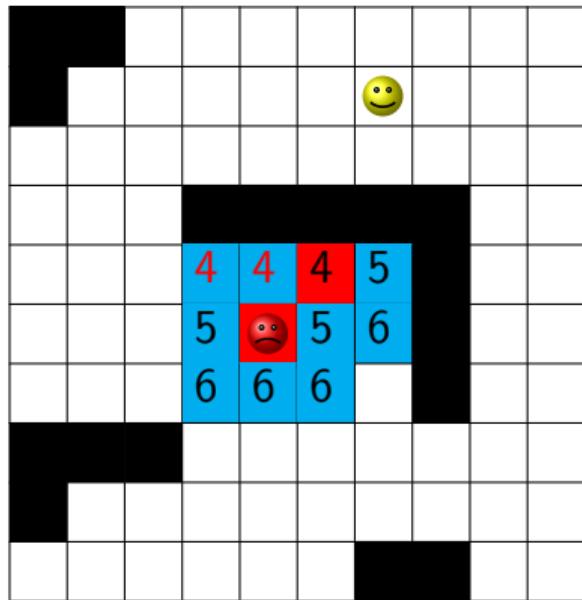
Algorithme A* (déroulé – étape 3)



Déroulons l'algorithme :

- ➊ liste ouverte = [■]
- ➋ liste fermée = [■]

Algorithme A* (déroulé – étape 3)



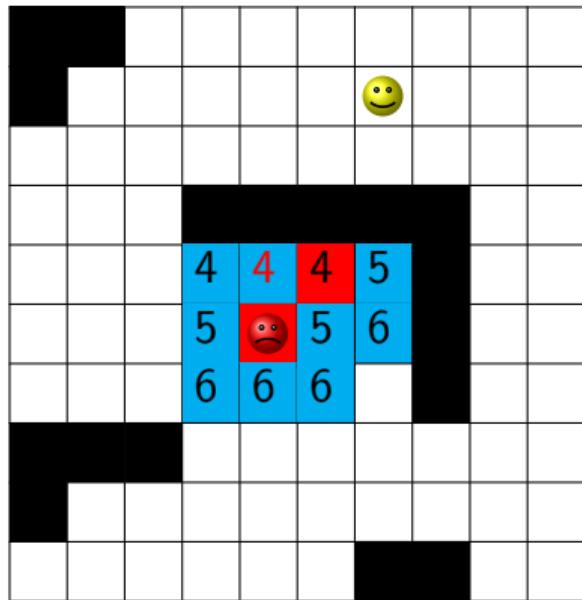
Déroulons l'algorithme :

- ① liste ouverte = [■]
- ② liste fermée = [■]

Choix du nouveau X :

on a 2 positions avec le même coût

Algorithme A* (déroulé – étape 3)



Déroulons l'algorithme :

- ➊ liste ouverte = [■]
- ➋ liste fermée = [■]

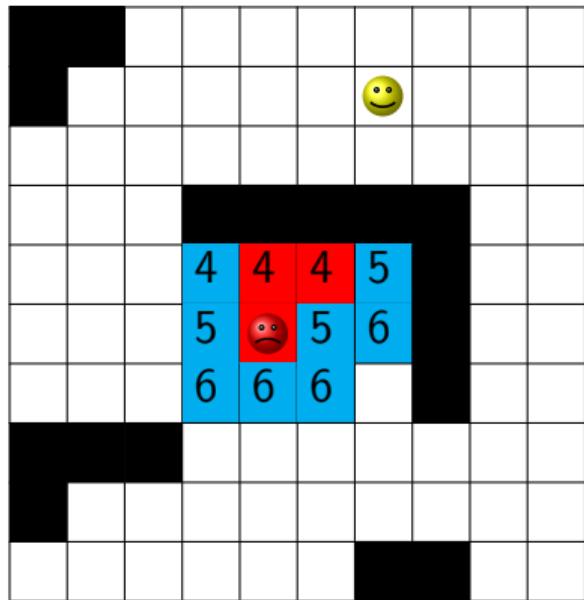
Choix du nouveau X :

on a 2 positions avec le même coût

⇒ on en choisit une au hasard

→ aucun voisin ne peut entrer dans la liste ouverte

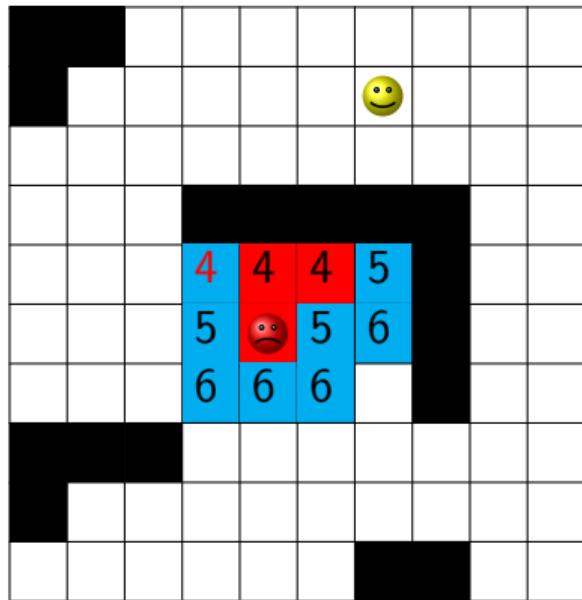
Algorithme A* (déroulé – étape 4)



Déroulons l'algorithme :

- ➊ liste ouverte = [■]
- ➋ liste fermée = [■]

Algorithme A* (déroulé – étape 4)

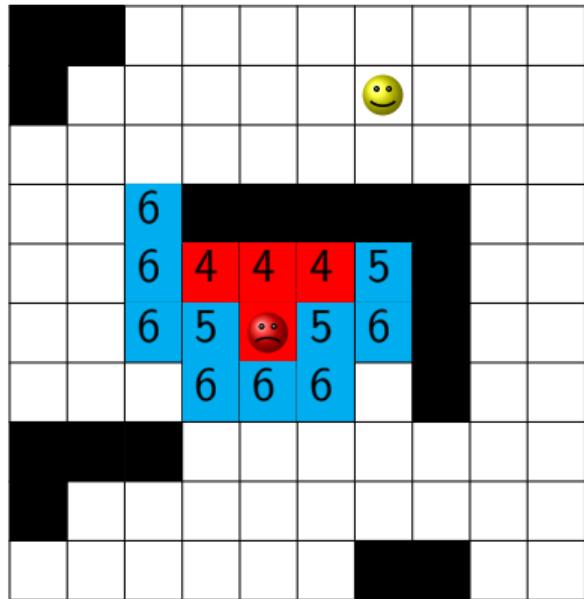


Déroulons l'algorithme :

- ① liste ouverte = [■]
- ② liste fermée = [■]

Choix du nouveau X :
on a un seul choix possible

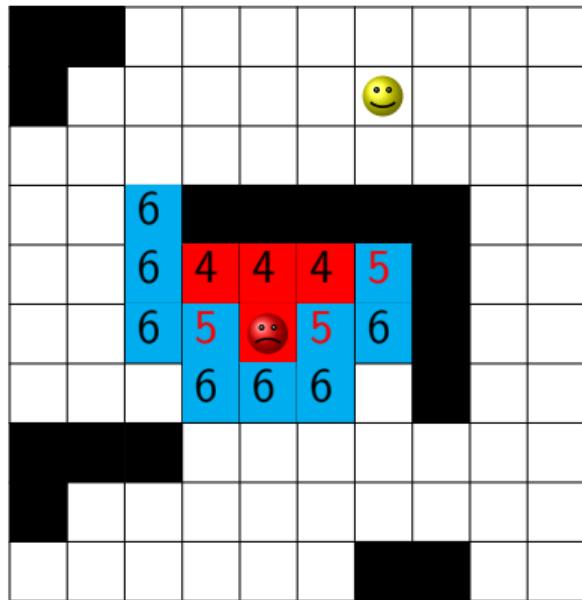
Algorithme A* (déroulé – étape 5)



Déroulons l'algorithme :

- ① liste ouverte = [■]
- ② liste fermée = [■]

Algorithme A* (déroulé – étape 5)



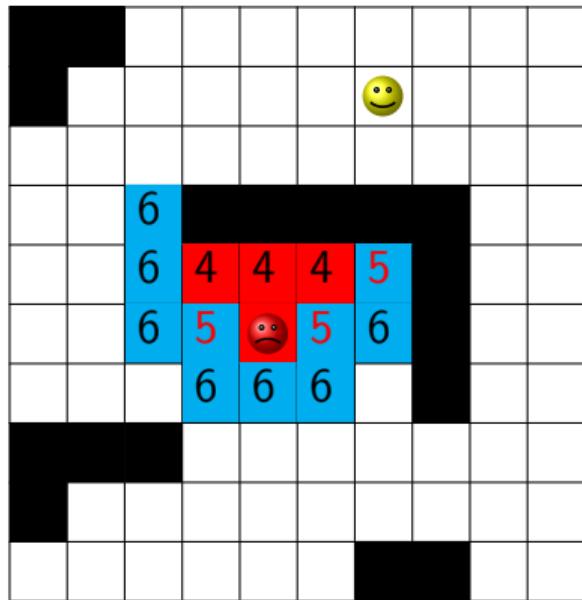
Déroulons l'algorithme :

- ① liste ouverte = [■]
- ② liste fermée = [■]

Choix du nouveau X :

on a 3 positions avec le même coût

Algorithme A* (déroulé – étape 5)



Déroulons l'algorithme :

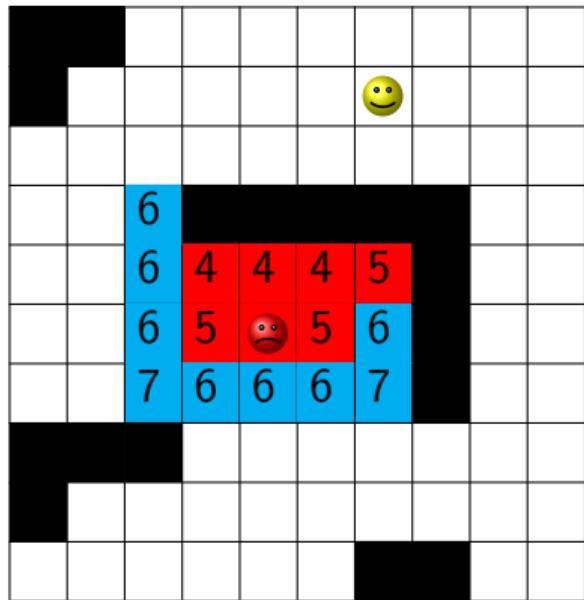
- ➊ liste ouverte = [■]
- ➋ liste fermée = [■]

Choix du nouveau X :

on a 3 positions avec le même coût

accélérerons un peu en traitant tous les 5...

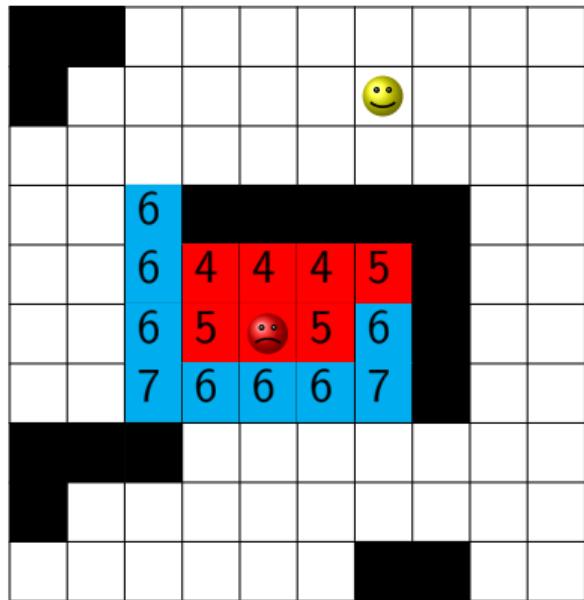
Algorithme A* (déroulé – résultat après avoir traité tous les 5)



Déroulons l'algorithme :

- ➊ liste ouverte = []
- ➋ liste fermée = []

Algorithme A* (déroulé – résultat après avoir traité tous les 5)

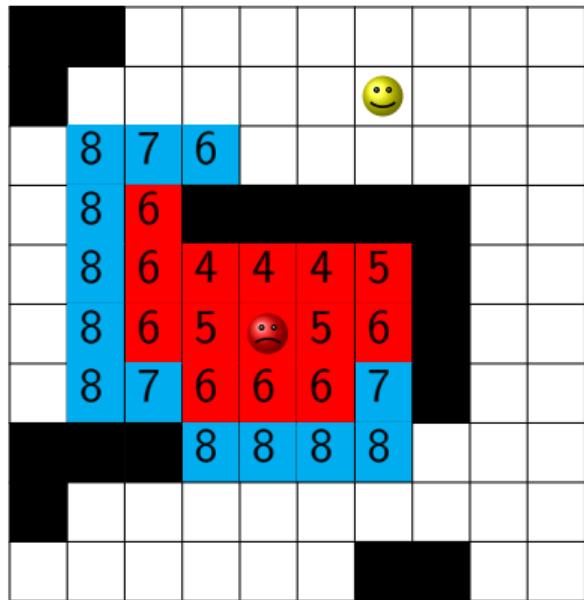


Déroulons l'algorithme :

- ➊ liste ouverte = [■]
- ➋ liste fermée = [■]

accélérerons un peu en traitant tous les 6...

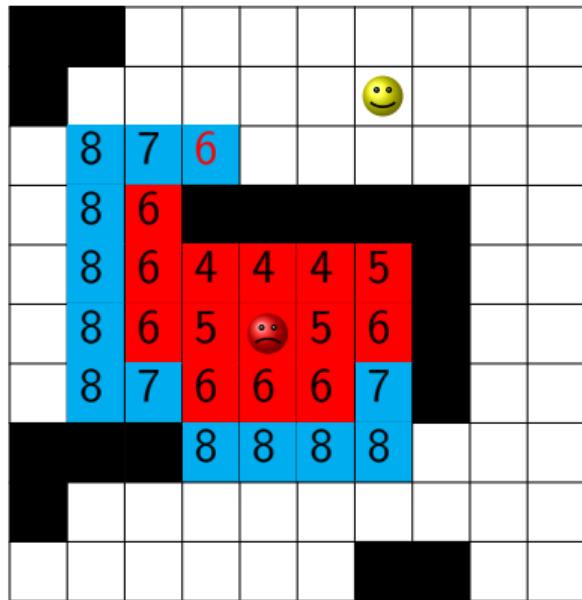
Algorithme A* (déroulé – résultat après avoir traité tous les 6)



Déroulons l'algorithme :

- ① liste ouverte = [■]
- ② liste fermée = [■]

Algorithme A* (déroulé – résultat après avoir traité tous les 6)



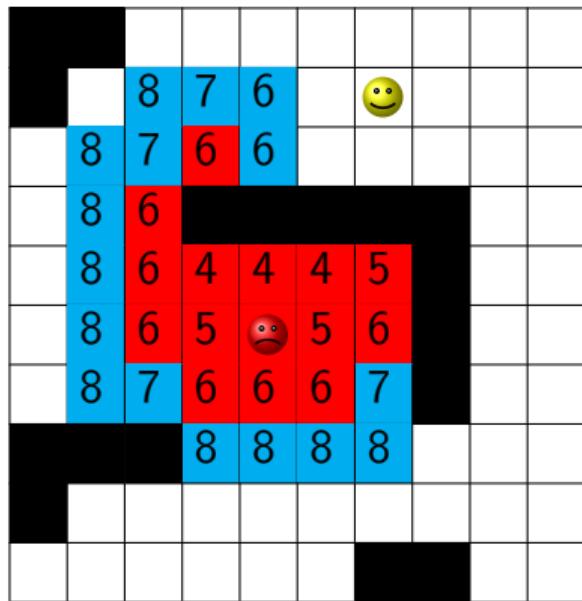
Déroulons l'algorithme :

- ➊ liste ouverte = [■]
- ➋ liste fermée = [■]

Remarques :

- un nouveau **6** apparaît et sera le prochain **X**
- les coûts commencent à être réalistes
⇒ « chemin du haut » probable
- les positions du bas sont sur la **liste ouverte**
⇒ une autre voie reste possible si besoin !

Algorithme A* (déroulé – traitement des nouveaux 6)



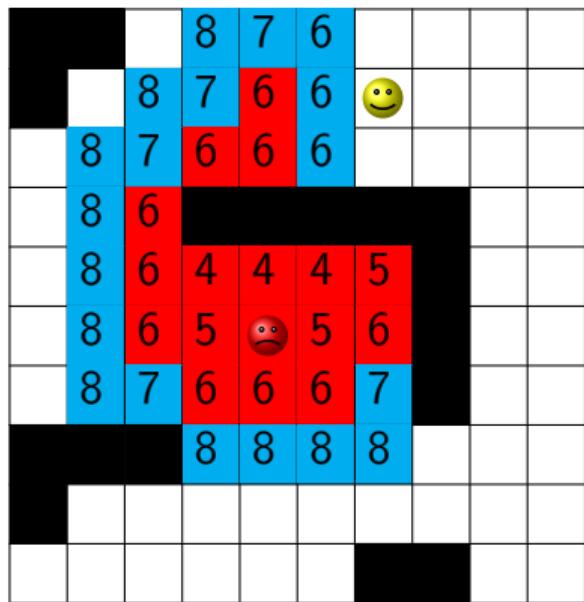
Déroulons l'algorithme :

- ➊ liste ouverte = [■]
- ➋ liste fermée = [■]

Encore des 6 !

⇒ plusieurs choix possibles
⇒ plusieurs issues possibles !

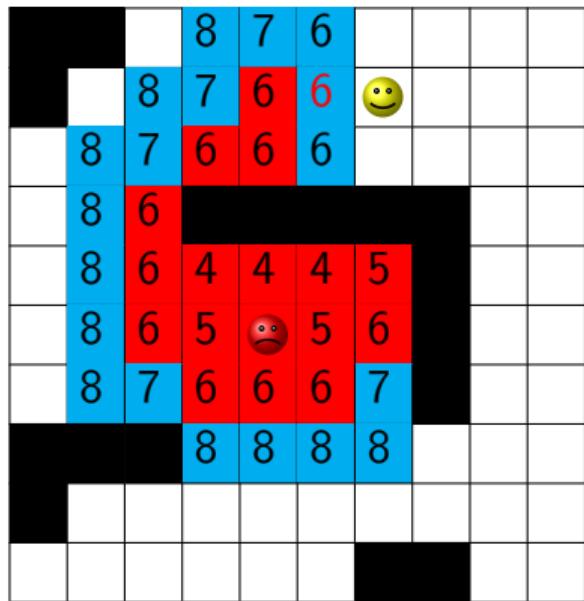
Algorithme A* (déroulé – traitement des nouveaux nouveaux 6)



Déroulons l'algorithme :

- ➊ liste ouverte = [■]
- ➋ liste fermée = [■]

Algorithme A* (déroulé – traitement des nouveaux nouveaux 6)



Déroulons l'algorithme :

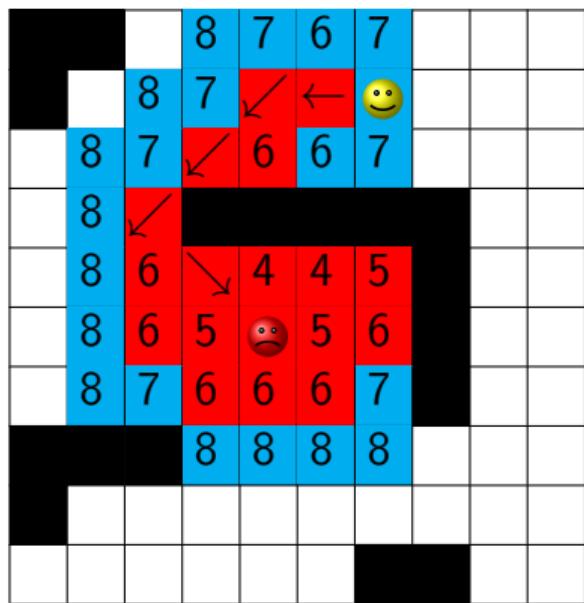
- ➊ liste ouverte = [■]
- ➋ liste fermée = [■]

On approche, prenons un 6 au hasard

Remarques :

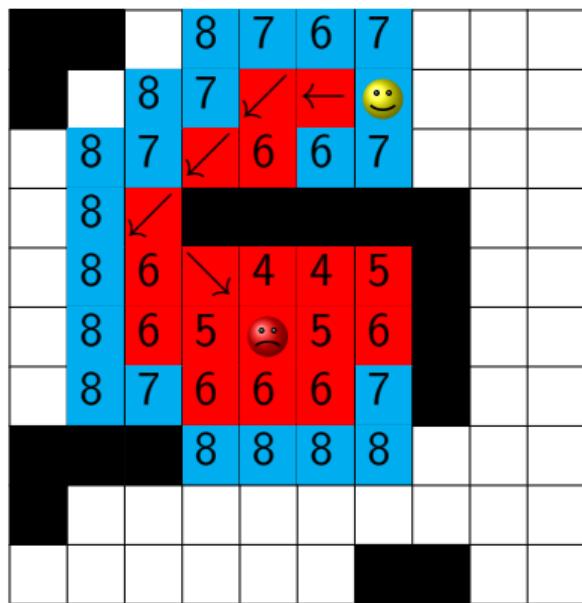
- on a plusieurs chemins possibles
mais **tous de la même longueur**
- le chemin par le 6 du haut est un peu bizarre
(petites améliorations possibles)

Algorithme A* (déroulé – retour à l'origine via la trace laissée)



Dès que 😊 entre dans la [liste ouverte](#) ou s'arrête
il n'y a plus qu'à remonter à la source
en suivant la trace laissée à l'allée

Algorithme A* (déroulé – retour à l'origine via la trace laissée)

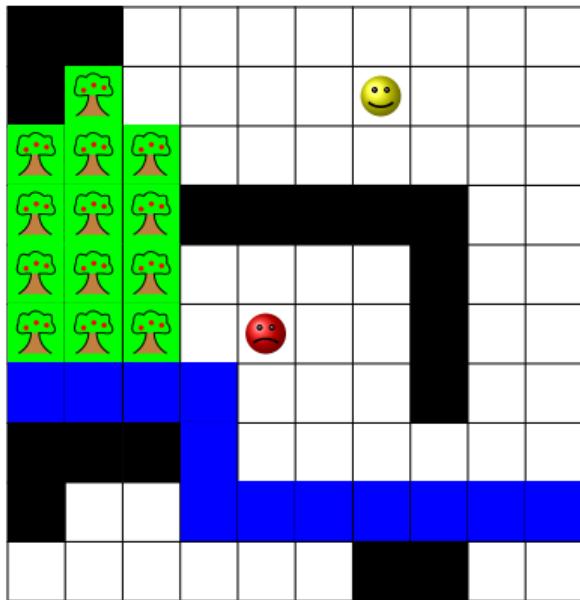


Dès que 😊 entre dans la **liste ouverte** ou s'arrête
il n'y a plus qu'à remonter à la source
en suivant la trace laissée à l'allée

Remarques :

- si on avait eu une **liste ouverte** vide
⇒ aucun chemin possible

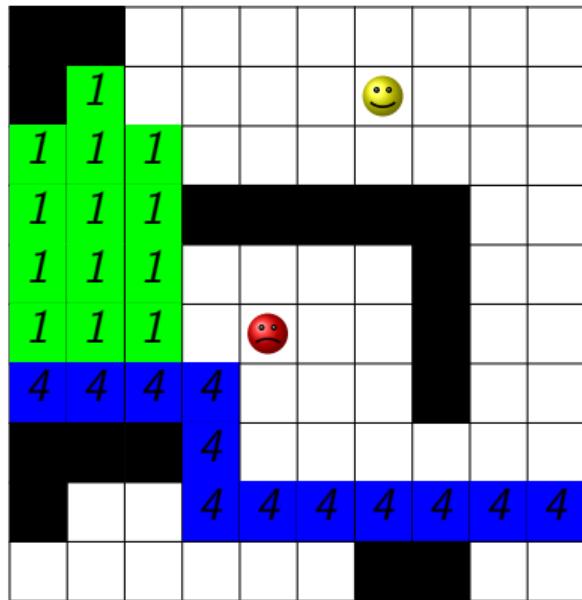
Algorithme A* – ajout d'un sur-coût selon le terrain



On peut prendre en compte le type de terrain via un sur-coûts associé à chaque position :

coût(Y) = distance depuis 😞
+ estimation de la distance jusqu'à 😊
+ coût du terrain de la position Y

Algorithme A* – ajout d'un sur-coût selon le terrain



On peut prendre en compte le type de terrain via un sur-coûts associé à chaque position :

coût(Y) = distance depuis 😞
+ estimation de la distance jusqu'à 😊
+ coût du terrain de la position Y

Par exemple : – forêt = 1,
– rivière = 4,
– falaise = 10, etc.

Algorithme A* – cartes dynamiques

On peut aussi **modifier la carte des sur-coûts en fonction du game play**, par exemple :

- ajout d'un gros sur-coût sur la ligne de mire d'un char
⇒ on peut traverser mais c'est risqué !
- augmentation du coût de la position où le personnage contrôlé par l'IA se fait tuer par le joueur
⇒ évite les guet-apens
⇒ peut faire passer l'IA de « méga stupide » à « rusée » car l'IA semble avoir compris la tactique du joueur !
- etc.

À faire... et à rendre !

- ① Implémenter l'algorithme A* (langage au choix)
avec un affichage au choix dans la console ou graphique
- ② Essayer sur l'exemple du cours
- ③ Donner au joueur la possibilité de déplacer 😊
(par exemple via les touches du pavé numérique)
et lancer un ou plusieurs 😟 à sa poursuite !

Facultatif : si 😊 croque un gâteau il peut durant 10 cycles tuer 😟 !
⇒ il faut modifier le comportement de 😟

Remarque : rester simple (structure de données, graphismes, etc.)

1

Définition de l'Intelligence Artificielle

- L'IA selon Wikipédia
- L'IA dans la littérature, le cinéma, et la recherche
- L'IA pour les jeux vidéos

2

Quelques algorithmes indispensables

- Recherche du plus court chemin : Dijkstra
- Recherche du plus court chemin : algorithme A*
- **Flocking**

3

Automates d'états fini

4

Réseaux de neurones artificielles

- Le neurone formel et le perceptron
- Apprentissage et Généralisation
- Le perceptron multi-couche et la rétro-propagation du gradient
- Application aux jeux vidéos, analyse du comportement, article de 2010

Définition du Flocking dans le règne animal

Définition de Wikipédia

Le flocking – ou agrégation en français – désigne un regroupement plus ou moins temporaire d'animaux grégaires.

Il y a plusieurs synonymes selon l'espèce :

- une **nuée** (ou une ronde) d'oiseaux
- un **banc** de poissons
- un **essaim** d'insectes
- un **troupeau** de mammifères

Définition du Flocking dans le règne animal

Définition de Wikipédia

Le flocking – ou agrégation en français – désigne un regroupement plus ou moins temporaire d'animaux grégaires.

Il y a plusieurs synonymes selon l'espèce :

- une **nuée** (ou une ronde) d'oiseaux \equiv *a flock of birds* *foraging*
- un **banc** de poissons \equiv *school of fish* *shoaling*
- un **essaims** d'insectes \equiv *swarm of insects* *swarming*
- un **troupeau** de mammifères \equiv *herd of mammals* *herd behavior*

Remarque : il y a évidemment l'équivalent en [en anglais](#) !

Modèle de Craig Reynolds

Idée : – chaque individu peut observer ses voisins (à une distance limitée ou non)
– chaque individu du groupe est identique (pas de leader)
– le comporte de groupe « émerge »

Craig Reynolds propose 3 règles :

- se déplacer dans la même direction que ses voisins
- rester suffisamment proche de ses voisins
- éviter les collisions avec ses voisins



Remarque : « boid(s) » = *bird android(s)*

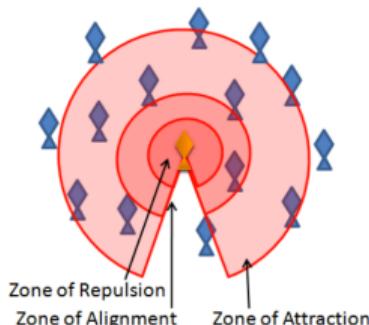
Modèle de Craig Reynolds

On calcule :

- ① la direction moyenne des voisins situés dans la zone d'**alignement**
- ② la direction pour s'éloigner des voisins les plus proches (**répulsion**)
- ③ la direction pour se rapprocher des voisins les plus éloignés (**attraction**)
- ④ la nouvelle direction que l'individu doit prendre
= moyenne des directions calculées en ① ② ③

Remarque : le choix des zones influe sur le comportement
⇒ à régler à la main

Exemple : zone dirigée vers l'avant
⇒ déplacement en file indienne



Exemple avec des « particules » en python

Exemple avec des particules (pas orienté, pas de champ de vision) et en Python

```
#!/usr/bin/env python
# Boid implementation in Python using PyGame -- Ben Dowling

import sys, pygame, random, math

pygame.init()

size = width, height = 800, 600
black = 0, 0, 0

maxVelocity = 10
numBoids = 50

boids = []
```

Exemple avec des « particules » en python

Définition de la classe Boid :

```
class Boid:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
        self.velocityX = random.randint(1, 10) / 10.0  
        self.velocityY = random.randint(1, 10) / 10.0  
  
    "Return the distance from another boid"  
    def distance(self, boid):  
        distX = self.x - boid.x  
        distY = self.y - boid.y  
        return math.sqrt(distX * distX + distY * distY)
```

... suite de la classe Boid, méthode d'attraction :

```
def moveCloser(self, boids):
    if len(boids) < 1: return
    # calculate the average distances from the other boids
    avgX = 0
    avgY = 0
    for boid in boids:
        if boid.x == self.x and boid.y == self.y:
            continue
        avgX += (self.x - boid.x)
        avgY += (self.y - boid.y)
    avgX /= len(boids)
    avgY /= len(boids)
    # set our velocity towards the others
    self.velocityX -= (avgX / 100)
    self.velocityY -= (avgY / 100)
```

... suite de la classe Boid, méthode d'alignement :

```
def moveWith(self, boids):
    if len(boids) < 1: return
    # calculate the average velocities of the other boids
    avgX = 0
    avgY = 0
    for boid in boids:
        avgX += boid.velocityX
        avgY += boid.velocityY
    avgX /= len(boids)
    avgY /= len(boids)
    # set our velocity towards the others
    self.velocityX += (avgX / 40)
    self.velocityY += (avgY / 40)
```

... suite de la classe Boid, début de la méthode **de répulsion** :

```
def moveAway(self, boids, minDistance):
    if len(boids) < 1: return
    distanceX = 0
    distanceY = 0
    numClose = 0
    for boid in boids:
        distance = self.distance(boid)
        if distance < minDistance:
            numClose += 1
            xdiff = (self.x - boid.x)
            ydiff = (self.y - boid.y)
            if xdiff >= 0: xdiff = math.sqrt(minDistance) - xdiff
            elif xdiff < 0: xdiff = -math.sqrt(minDistance) - xdiff
            if ydiff >= 0: ydiff = math.sqrt(minDistance) - ydiff
            elif ydiff < 0: ydiff = -math.sqrt(minDistance) - ydiff
            distanceX += xdiff
            distanceY += ydiff
```

... suite de la classe Boid, suite de la méthode de répulsion :

```
if numClose == 0:  
    return  
self.velocityX -= distanceX / 5  
self.velocityY -= distanceY / 5
```

... suite de la classe Boid, méthode déplacement proprement dit :

```
def move(self):  
    if abs(self.velocityX) > maxVelocity or \  
        abs(self.velocityY) > maxVelocity:  
  
        scaleFactor = maxVelocity / max(abs(self.velocityX), \  
                                         abs(self.velocityY))  
        self.velocityX *= scaleFactor  
        self.velocityY *= scaleFactor  
    self.x += self.velocityX  
    self.y += self.velocityY
```

Début du corps du programme, initialisation de l'environnement :

```
screen = pygame.display.set_mode(size)
ball = pygame.image.load("ball.png")
ballrect = ball.get_rect()
```

et création des boids :

```
# create boids at random positions

for i in range(numBoids):
    boids.append(Boid(random.randint(0, width), random.randint(0, height)))
```

Début de la boucle principale :

```
while 1:  
    for event in pygame.event.get():  
        if event.type == pygame.QUIT: sys.exit()  
  
    for boid in boids:  
        closeBoids = []  
        for otherBoid in boids:  
            if otherBoid == boid: continue  
            distance = boid.distance(otherBoid)  
            if distance < 200:  
                closeBoids.append(otherBoid)  
  
        boid.moveCloser(closeBoids)  
        boid.moveWith(closeBoids)  
        boid.moveAway(closeBoids, 20)
```

... suite de la boucle principale, on est toujours dans le << for boid in boids >>,
on s'assure qu'on est toujours dans l'espace de jeux puis on se déplace :

```
# if we roubound we can lose some of our velocity
border = 25
if boid.x < border and boid.velocityX < 0:
    boid.velocityX = -boid.velocityX * random.random()
if boid.x > width - border and boid.velocityX > 0:
    boid.velocityX = -boid.velocityX * random.random()
if boid.y < border and boid.velocityY < 0:
    boid.velocityY = -boid.velocityY * random.random()
if boid.y > height - border and boid.velocityY > 0:
    boid.velocityY = -boid.velocityY * random.random()

boid.move()
```

... fin du code de la boucle principale (le << while 1 >>) et du programme,
on nettoie l'écran et on redessine tous les boids,
on attend un peu puis on boucle :

```
screen.fill(black)

for boid in boids:
    boidRect = pygame.Rect(ballrect)
    boidRect.x = boid.x
    boidRect.y = boid.y
    screen.blit(ball, boidRect)

pygame.display.flip()
pygame.time.delay(10)

# THE END
```

Le Flocking en action !

À faire [mais pas à rendre](#) :

- ① testez le programme qu'on vient de voir (reçu par mail) : [boids_balls.py](#)
...jouez avec les paramètres...
- ② testez les programmes [curseboid.py](#) (console)
et [glboid.py](#) (graphique)
qui implémentent des **boids orientés**
(depuis l'archive [optboid-master.zip](#) (reçue par mail))

Remarque : sous Ubuntu j'ai dû installer quelques packages :

- « `sudo apt-get install python-pygame` » pour [boids_balls.py](#)
 - « `sudo pip install pyglet` » pour [glboid.py](#)
- mais ce sera probablement différent selon vos environnements respectifs !

Le Flocking en action !

À faire **et à rendre :**

- ① ajouter des obstacles voire des murs dans le programme `boids_balls.py`
- ② ajouter une méthode pour que les boids aillent vers une cible
- ③ ajouter un participant contrôlé au clavier qui sera la cible !

Le Flocking en action !

À faire **et à rendre :**

- ① ajouter des obstacles voire des murs dans le programme `boids_balls.py`
- ② ajouter une méthode pour que les boids aillent vers une cible
- ③ ajouter un participant contrôlé au clavier qui sera la cible !

Remarque : – **faire simple !!!!**

- les boids doivent (au moins essayer) d'éviter les obstacles,
pas les percuter de plein fouet !
- vous pouvez partir de `curseboid.py` ou `glboid.py`
mais seulement si vous êtes sûrs de vous !
- vous pouvez partir d'un autre programme si vous n'aimez pas Python
mais ça risque de vous prendre beaucoup (trop) de temps... .

Exemple de programme python pour lire le clavier (1/2)

Pour vous aider : fonction Python pour déplacer une balle au pavé numérique :

```
# script clavier.py

from Tkinter import *

def touche(event):
    global X,Y
    k = event.keysym
    decX = 20 if k in [ 'KP_1', 'KP_4', 'KP_7'] else 0
    incX = 20 if k in [ 'KP_3', 'KP_6', 'KP_9'] else 0
    decY = 20 if k in [ 'KP_7', 'KP_8', 'KP_9'] else 0
    incY = 20 if k in [ 'KP_1', 'KP_2', 'KP_3'] else 0
    X += incX - decX
    Y += incY - decY
    Canevas.coords(balle, X-10, Y-10, X+10, Y+10) # on redessine la balle
```

Exemple de programme python pour lire le clavier (1/2)

... et le programme principal qui va avec :

```
Mafenetre = Tk()
Mafenetre.title('Lecture du clavier pour bouger une balle')
L = 800
H = 600
X = L / 2
Y = H / 2
Canevas = Canvas(Mafenetre, width = L, height = H, bg ='white')
balle = Canevas.create_oval(X-10,Y-10,X+10,Y+10,width=2,fill='blue')
Canevas.focus_set()
Canevas.bind('<Key>', touche)
Canevas.pack()

Mafenetre.mainloop()
```

Exemple de programme python pour lire le clavier... avec pygame

Hélas c'est différent avec pygame (les joies du Python), voilà :

```
import pygame
from pygame import *

pygame.init()
pygame.display.set_mode((800, 600))
boucle = True
while True :
    for event in pygame.event.get():
        if event.type==KEYDOWN :
            print event.key
            if event.key==K_RETURN :
                boucle = False
```

→ les touches 1 à 9 du pavé numérique renvoient les valeurs 257 à 265.

1

Définition de l'Intelligence Artificielle

- L'IA selon Wikipédia
- L'IA dans la littérature, le cinéma, et la recherche
- L'IA pour les jeux vidéos

2

Quelques algorithmes indispensables

- Recherche du plus court chemin : Dijkstra
- Recherche du plus court chemin : algorithme A*
- Flocking

3

Automates d'états fini

4

Réseaux de neurones artificielles

- Le neurone formel et le perceptron
- Apprentissage et Généralisation
- Le perceptron multi-couche et la rétro-propagation du gradient
- Application aux jeux vidéos, analyse du comportement, article de 2010

Définition d'un automates d'états fini

Définition de Wikipédia

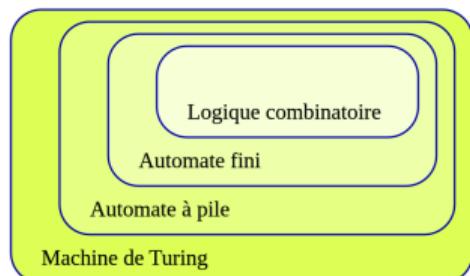
Un automate fini ou **automate avec un nombre fini d'états** (en anglais finite-state automaton ou finite state machine) est un modèle mathématique de calcul, utilisé dans de nombreuses circonstances : conception de programmes informatiques et de circuits en logique séquentielle, applications dans des protocoles de communication, contrôle des processus, linguistique, biologie, ...

Définition d'un automates d'états fini

Définition de Wikipédia

Un automate fini ou **automate avec un nombre fini d'états** (en anglais finite-state automaton ou finite state machine) est un modèle mathématique de calcul, utilisé dans de nombreuses circonstances : conception de programmes informatiques et de circuits en logique séquentielle, applications dans des protocoles de communication, contrôle des processus, linguistique, biologie, ...

Hiérarchie des modèles de calcul classiques :



Remarque : automates finis = concept abstrait de calcul,
mais **puissance faible** (< machine de Turing) : nombre d'états fini !

Description d'un automate fini

Description :

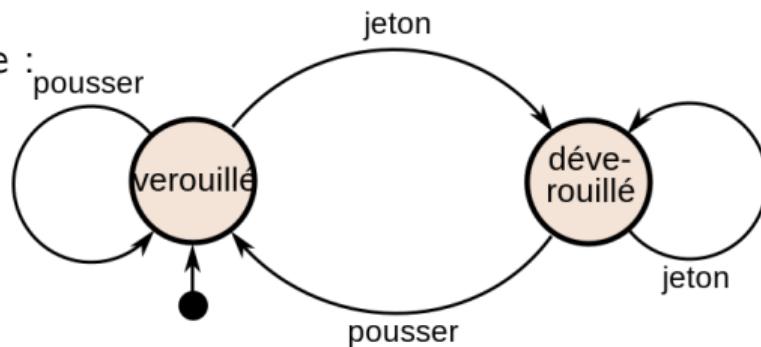
- **nombre fini d'états**,
- **un seul état actif à la fois** : « **état courant** »,
- passage d'un état à un autre dirigé par un **événement** ou une **condition** : on parle de « **transition** ».

Description d'un automate fini

Description :

- **nombre fini d'états**,
- **un seul état actif à la fois** : « **état courant** »,
- passage d'un état à un autre dirigé par un **événement** ou une **condition** : on parle de « **transition** ».

Exemple :



Notation formelle

$$\mathcal{A} = (\Sigma, S, s_0, \delta, F)$$

avec :

- Σ : un ensemble fini, non vide, de lettres qui est l'alphabet d'entrée
- S : un ensemble fini, non vide, d'états
- s_0 : l'état initial ($s_0 \in S$)
- δ : fonction de transition d'états, on a donc $\delta : S \times \Sigma \rightarrow S$
- F : ensemble d'états états terminaux ($F \subseteq S$)

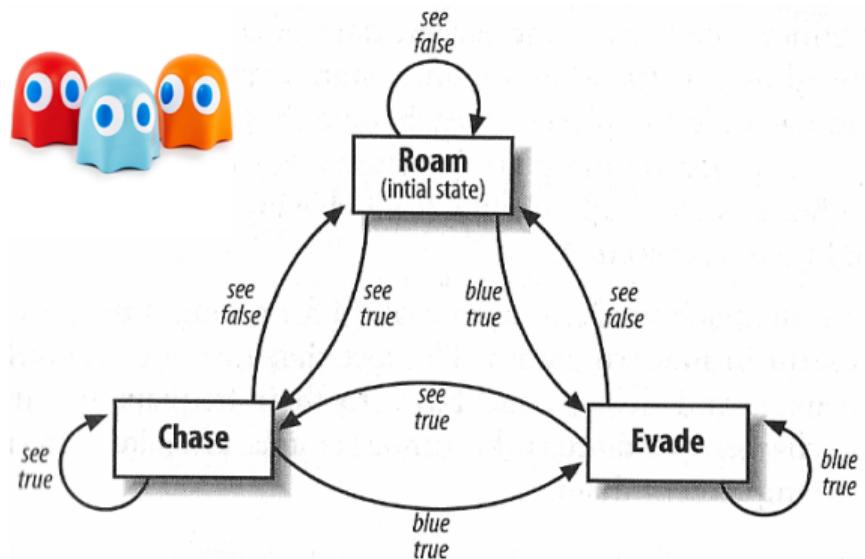
Intérêt pour l'IA des jeux vidéos

- Bien plus lisible que des if, then else
- Programmation dirigée par les événements
- Très peu de ressources nécessaires
- Pas compliqué à programmer

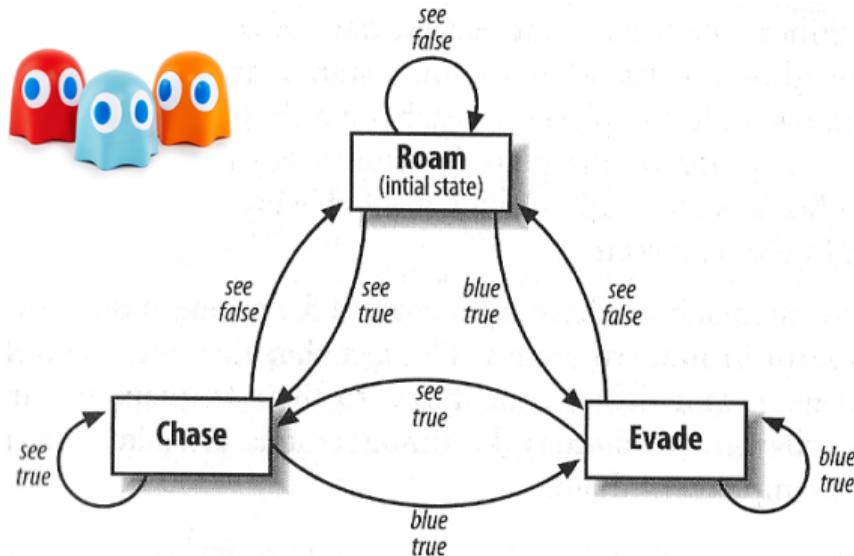
Problèmes :

- jeu complexe : risque de multiplication du nombre d'état
→ plusieurs automates (concurrents) nécessaires
- pas de mémoire (état précédent)
→ mise en œuvre d'automates « à pile »

Exemple classique : les fantômes de PacMan



Exemple classique : les fantômes de PacMan



```
switch (currentState)
{
    case kRoam:
        if (imBlue==true) currentState=kEvade;
        else if (canSeePlayer==true) currentState=kChase;
        else if (canSeePlayer==false) currentState=kRoam;
        break;

    case kChase:
        if (imBlue==true) currentState=kEvade;
        else if (canSeePlayer==false) currentState=kRoam;
        else if (canSeePlayer==true) currentState=kChase;
        break;

    case kEvade:
        if (imBlue==true) currentState=kEvade;
        else if (canSeePlayer==true) currentState=kChase;
        else if (canSeePlayer==false) currentState=kRoam;
        break;
}
```

Exemple plus complexe avec un jeu de « fight »

Voyons ensemble : <http://gameprogrammingpatterns.com/state.html>

- intérêt des automates à états fini dans un jeu de combat (genre *street fighter*)
- exemple d'une **machine avec pile**
- intérêt et exemple de **machines concurrentes** (pour minimiser le nombre d'états)
- détails d'implémentation en C++ (on verra pas mais *up to you !*)

À faire !

Intégrer un mécanisme à la PacMan (pastille pour manger la méchante IA), au choix :

- dans le jeu de l'algorithme A*
- dans le jeu du flocking
- dans un nouveau jeu

Attention : bonus de points si rendu avant 18 heures!!!!

⇒ faire simple !



1

Définition de l'Intelligence Artificielle

- L'IA selon Wikipédia
- L'IA dans la littérature, le cinéma, et la recherche
- L'IA pour les jeux vidéos

2

Quelques algorithmes indispensables

- Recherche du plus court chemin : Dijkstra
- Recherche du plus court chemin : algorithme A*
- Flocking

3

Automates d'états fini

4

Réseaux de neurones artificielles

- Le neurone formel et le perceptron
- Apprentissage et Généralisation
- Le perceptron multi-couche et la rétro-propagation du gradient
- Application aux jeux vidéos, analyse du comportement, article de 2010

1

Définition de l'Intelligence Artificielle

- L'IA selon Wikipédia
- L'IA dans la littérature, le cinéma, et la recherche
- L'IA pour les jeux vidéos

2

Quelques algorithmes indispensables

- Recherche du plus court chemin : Dijkstra
- Recherche du plus court chemin : algorithme A*
- Flocking

3

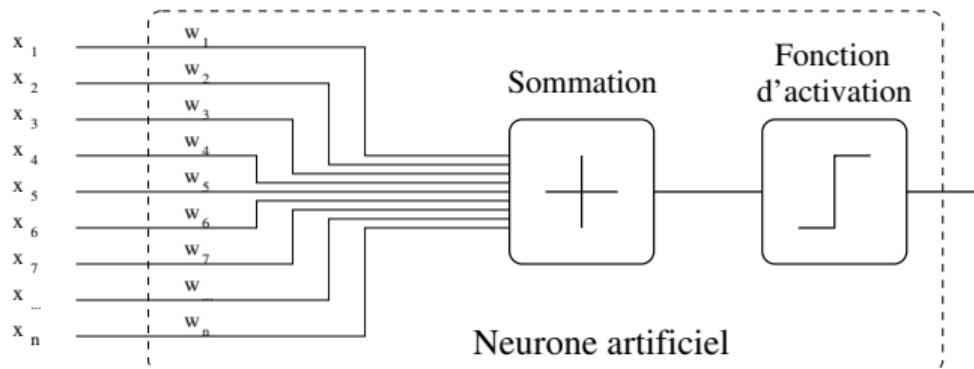
Automates d'états fini

4

Réseaux de neurones artificielles

- **Le neurone formel et le perceptron**
- Apprentissage et Généralisation
- Le perceptron multi-couche et la rétro-propagation du gradient
- Application aux jeux vidéos, analyse du comportement, article de 2010

Le neurone formel de Mc Culloch et Pitts (1943)



- vecteur d'entrée
 $X = (x_1, x_2, \dots, x_n)$
- vecteur poids
 $W = (w_1, w_2, \dots, w_n)$
- la sortie dépend de la fonction d'activation

Remarque : codage « en amplitude » et non « impulsionnel » pour un neurone réel.

Exemple de « l'automate booléen à seuil »

Description de l'automate booléen à seuil :

- vecteur d'entrée booléen
- fonction d'activation de Heaviside (notée \mathcal{H}) \Rightarrow sortie aussi booléenne
- poids et le seuil réels

Fonctionnement :

- ❶ calcul de la somme pondérée des entrées : $\sum_{i=1}^n w_i x_i$
- ❷ calcul de la sortie de la cellule $s = \mathcal{H}(\sum_{i=1}^n w_i x_i - \theta) = \begin{cases} 1 & si \sum_{i=1}^n w_i x_i > \theta \\ 0 & si \sum_{i=1}^n w_i x_i \leq \theta \end{cases}$

Présentation du perceptron de Rosenblatt (1958)

Description du perceptron :

- entrées réelles
- fonction d'activation de Heaviside (notée \mathcal{H}) \Rightarrow sortie encore booléenne
- poids et le seuil réels

Principe : un perceptron, avec 1 neurone, divise l'espace d'entrée \mathbb{R}^n en deux (et seulement deux) sous-espaces séparés par un hyperplan d'équation :

$$w_1x_1 + \dots + x_iw_i + \dots + x_nw_n - \theta = 0$$

\Rightarrow on va pouvoir faire un « choix » \Rightarrow de la classification

Interprétation géométrique du perceptron

Si on se limite à \mathbb{R}^2 :

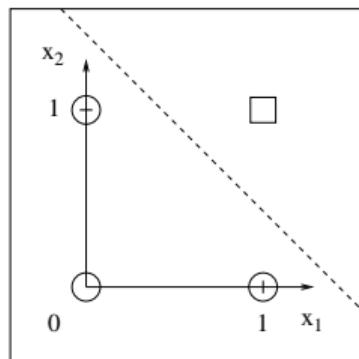
le perceptron divise l'espace d'entrée en deux demi-plans par une droite d'équation :

$$w_1x_1 + x_2w_2 - \theta = 0$$

- ⇒ selon l'intensité en entrée de la cellule, on considère un demi-plan ou l'autre
- ⇒ le perceptron est capable d'effectuer une discrimination de l'espace d'entrée en deux classes **linéairement séparables**.

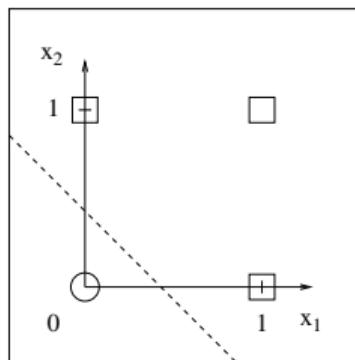
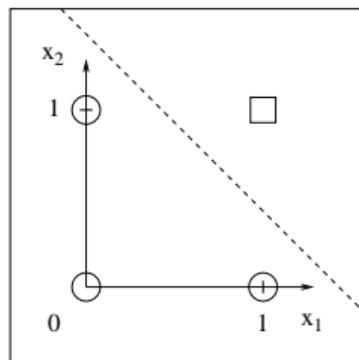
Simulation des fonctions *ET*, *OU*, et *XOR*

x_1	x_2	<i>ET</i>	<i>OU</i>	<i>OU exclusif</i>
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0



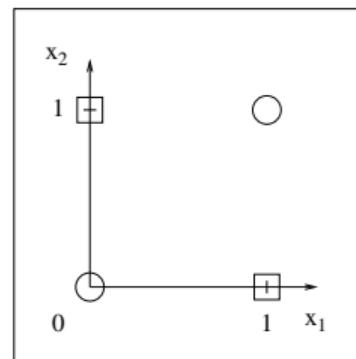
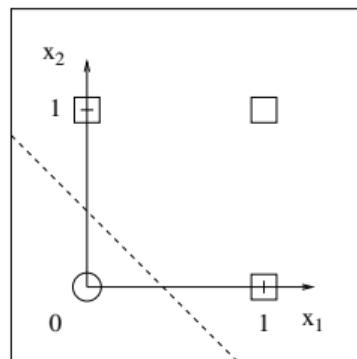
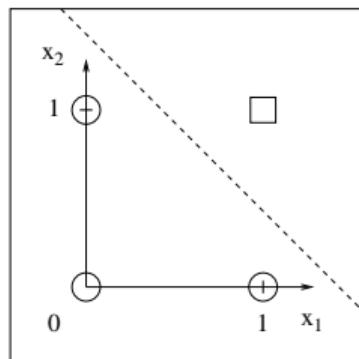
Simulation des fonctions *ET*, *OU*, et *XOR*

x_1	x_2	<i>ET</i>	<i>OU</i>	<i>OU exclusif</i>
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0



Simulation des fonctions *ET*, *OU*, et *XOR*

x_1	x_2	<i>ET</i>	<i>OU</i>	<i>OU exclusif</i>
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0



Le problème du « ou exclusif » (XOR)

⇒ Impossible de discriminer les valeurs de vérité du *OU exclusif!*

La limitation est significative car le pourcentage de fonctions booléennes linéairement séparables diminue très rapidement si on augmente la dimension de l'espace d'entrée :

n	séparable	total
2	14	16
3	104	256
4	1882	65536
5	94572	$4,3 \cdot 10^9$
6	5028134	$1,8 \cdot 10^{19}$

Théorème de McCulloch et Pitts

Remarque : on peut simuler un NON logique

$$\begin{cases} 0 \times w_1 + 0 - \theta > 0 \\ 1 \times w_1 + 0 - \theta \leq 0 \end{cases} \quad \begin{cases} \theta < 0 \\ w_1 \leq \theta \end{cases} \quad \begin{cases} \theta = -0,5 \\ w_1 = -1 \end{cases}$$

Théorème de McCulloch et Pitts

Remarque : on peut simuler un NON logique

$$\begin{cases} 0 \times w_1 + 0 - \theta > 0 \\ 1 \times w_1 + 0 - \theta \leq 0 \end{cases} \quad \begin{cases} \theta < 0 \\ w_1 \leq \theta \end{cases} \quad \begin{cases} \theta = -0,5 \\ w_1 = -1 \end{cases}$$

⇒ on a les 3 opérateurs de la logique booléenne

⇒ on peut synthétiser toutes les fonctions booléennes

Théorème de McCulloch et Pitts

Remarque : on peut simuler un NON logique

$$\begin{cases} 0 \times w_1 + 0 - \theta > 0 \\ 1 \times w_1 + 0 - \theta \leq 0 \end{cases} \quad \begin{cases} \theta < 0 \\ w_1 \leq \theta \end{cases} \quad \begin{cases} \theta = -0,5 \\ w_1 = -1 \end{cases}$$

⇒ on a les 3 opérateurs de la logique booléenne

⇒ on peut synthétiser toutes les fonctions booléennes

Théorème : Toute fonction booléenne de n variables peut être synthétisée par un **réseau** d'automates à seuil possédant n entrées

Théorème de McCulloch et Pitts

Remarque : on peut simuler un NON logique

$$\begin{cases} 0 \times w_1 + 0 - \theta > 0 \\ 1 \times w_1 + 0 - \theta \leq 0 \end{cases} \quad \begin{cases} \theta < 0 \\ w_1 \leq \theta \end{cases} \quad \begin{cases} \theta = -0,5 \\ w_1 = -1 \end{cases}$$

⇒ on a les 3 opérateurs de la logique booléenne

⇒ on peut synthétiser toutes les fonctions booléennes

Théorème : Toute fonction booléenne de n variables peut être synthétisée par un **réseau** d'automates à seuil possédant n entrées
... mais encore faut-il savoir le construire !

1

Définition de l'Intelligence Artificielle

- L'IA selon Wikipédia
- L'IA dans la littérature, le cinéma, et la recherche
- L'IA pour les jeux vidéos

2

Quelques algorithmes indispensables

- Recherche du plus court chemin : Dijkstra
- Recherche du plus court chemin : algorithme A*
- Flocking

3

Automates d'états fini

4

Réseaux de neurones artificielles

- Le neurone formel et le perceptron
- Apprentissage et Généralisation**
- Le perceptron multi-couche et la rétro-propagation du gradient
- Application aux jeux vidéos, analyse du comportement, article de 2010

Apprentissage et Généralisation

Objectif : trouver les w_i et θ automatiquement
en présentant une base d'exemples au réseau

Apprentissage et Généralisation

Objectif : trouver les w_i et θ automatiquement
en présentant une base d'exemples au réseau

Il faut :

- un algorithme **d'apprentissage**
- une base d'exemples
= couples (vecteur d'entrée X, sortie désirée pour X)

Exemple : algorithme d'apprentissage du perceptron

Algorithme :

- ① On présente un exemple = on calcule la sortie du réseau
- ② Si la sortie ne correspond pas à la sortie désirée
= si on est pas du bon côté de l'hyperplan
⇒ on modifie les poids et le seuil pour déplacer l'hyperplan « au bon endroit »
- ③ On boucle...

Remarques : l'algorithme converge toujours . . . si c'est linéairement séparable !

La généralisation

Définition : la généralisation est la **réponse du réseau sur des exemples**
qui n'étaient pas dans la base d'apprentissage

Objectifs :

- connaître la réponse pour des exemples « inconnus »
couples (vecteur d'entrée X , ?)
- évaluer la qualité de l'apprentissage
couples (vecteur d'entrée X , sortie connue)

Performances en généralisation

Les performances en généralisation dépendent :

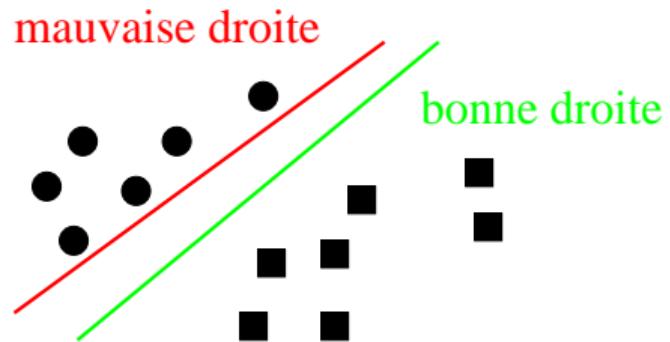
- de la qualité de l'algorithme d'apprentissage
- de l'adéquation du réseau (topologie, taille, dynamique...)
- de la qualité des exemples utilisés pour l'apprentissage, il faut notamment une bonne distribution des exemples

Performances en généralisation

Les performances en généralisation dépendent :

- de la qualité de l'algorithme d'apprentissage
- de l'adéquation du réseau (topologie, taille, dynamique...)
- de la qualité des exemples utilisés pour l'apprentissage, il faut notamment une bonne distribution des exemples

Exemple d'un **mauvais apprentissage** typique de l'algorithme classique du perceptron →



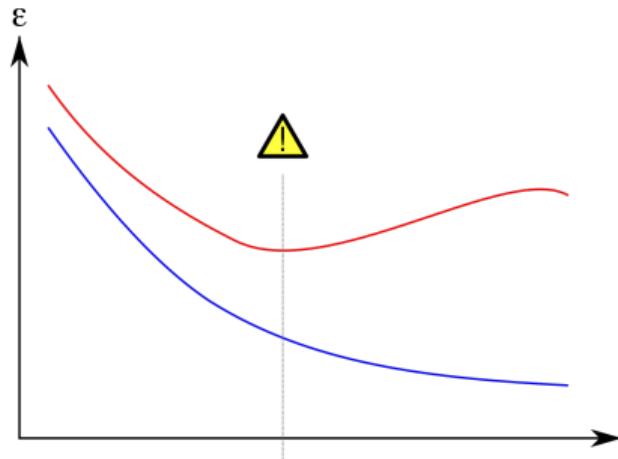
Le danger du sur-apprentissage

Dans une base d'apprentissage « réelle », on a souvent :

- un **bruit** important
- quelques **exemples non représentatifs**
- éventuellement des **erreurs** d'étiquetage...

⇒ pour certains problèmes, il est **inutile de chercher à apprendre tous les exemples**
⇒ arrêt de l'apprentissage une fois un certain pourcentage d'exemples appris
⇒ évite un « sur-apprentissage »
= *en apprenant trop « par cœur » on généralise moins bien*

Surapprentissage dans un apprentissage supervisé



Erreur en fonction des époques
(ie des présentations complètes de la base
d'apprentissage)

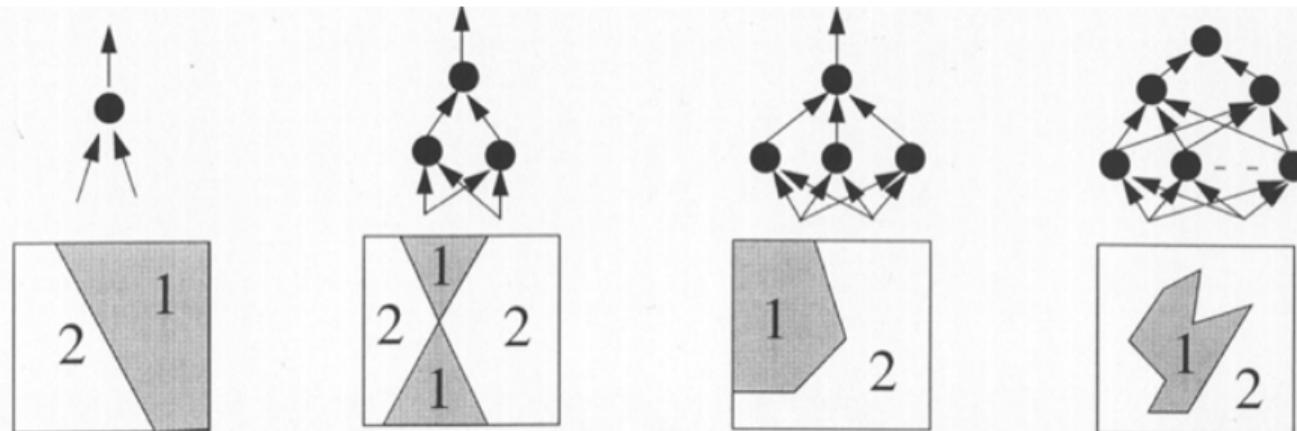
Rouge : l'erreur sur l'ensemble de
validation (généralisation)

Bleu : l'erreur d'apprentissage

Si l'erreur de validation augmente alors que l'erreur d'apprentissage continue à diminuer alors il y a un risque de surapprentissage.

The credit assignment problem

On a vu que l'on peut synthétiser toute fonction booléenne en utilisant **plusieurs couches de perceptrons** :



The credit assignment problem

On a vu que l'on peut synthétiser toute fonction booléenne en utilisant **plusieurs couches de perceptrons** :

mais on n'a pas la « sortie désirée » des couches non terminales
⇒ comment apprendre ? = « *credit assignment problem* »

The credit assignment problem

On a vu que l'on peut synthétiser toute fonction booléenne en utilisant **plusieurs couches de perceptrons** :

mais on n'a pas la « sortie désirée » des couches non terminales
⇒ comment apprendre ? = « ***credit assignment problem*** »

Problème mis en évidence par Minsky et Papert en 1969 et resté ouvert 20 ans !

The credit assignment problem

On a vu que l'on peut synthétiser toute fonction booléenne en utilisant plusieurs couches de perceptrons :

mais on n'a pas la « sortie désirée » des couches non terminales
⇒ comment apprendre ? = « *credit assignment problem* »

Problème mis en évidence par Minsky et Papert en 1969 et resté ouvert 20 ans !

Solution : algorithme de la rétro-propagation du gradient

1

Définition de l'Intelligence Artificielle

- L'IA selon Wikipédia
- L'IA dans la littérature, le cinéma, et la recherche
- L'IA pour les jeux vidéos

2

Quelques algorithmes indispensables

- Recherche du plus court chemin : Dijkstra
- Recherche du plus court chemin : algorithme A*
- Flocking

3

Automates d'états fini

4

Réseaux de neurones artificielles

- Le neurone formel et le perceptron
- Apprentissage et Généralisation
- Le perceptron multi-couche et la rétro-propagation du gradient**
- Application aux jeux vidéos, analyse du comportement, article de 2010

Comment apprendre avec un perceptron multi-couche ?

Comment apprendre avec un perceptron multi-couche ?

Idée : **rétro-propager l'erreur de la sortie vers l'entrée,**
en modifiant les poids des connexions en fonction de leur contribution à l'erreur

Comment apprendre avec un perceptron multi-couche ?

Idée : **rétro-propager l'erreur de la sortie vers l'entrée,**
en modifiant les poids des connexions en fonction de leur contribution à l'erreur

Rappel : apprentissage = modifier les poids

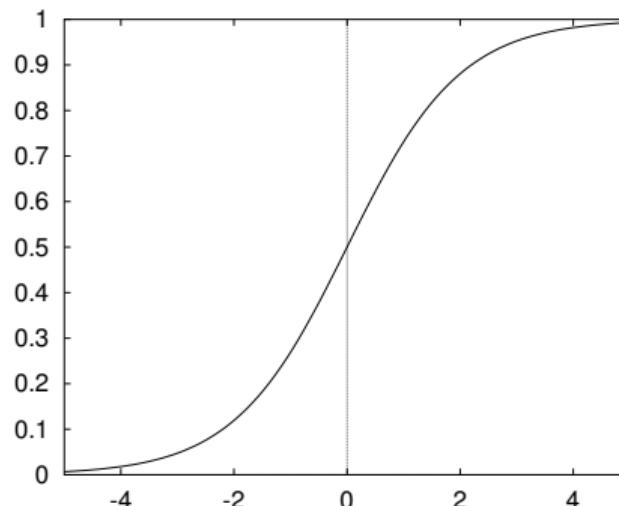
- ⇒ on doit expliciter l'erreur globale en fonction de tous les poids puis dériver cette fonction par rapport aux poids dont on veut connaître la contribution à l'erreur globale
- ⇒ il faut utiliser des fonctions d'activation dérивables !

La fonction sigmoïde

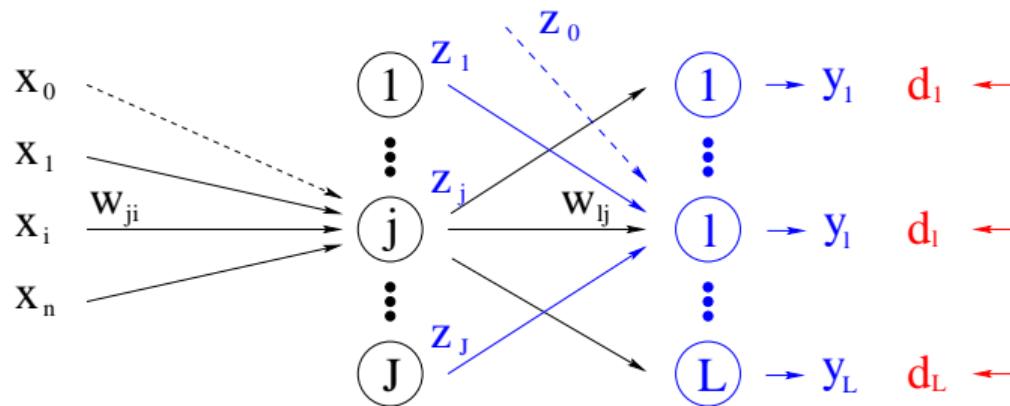
On veut toujours un « effet seuil » mais avec une fonction dérivable

On peut par exemple utiliser la fonction sigmoïde :

$$f(a) = \frac{1}{(1 + e^{-a})}$$



Topologie détaillée d'un perceptron multi-couche



- | | | | | |
|------------------|---|-----------------------------------|------------------------|-------------------------------|
| Couche d'entrée | : | $X = (x_0, x_1, x_2, \dots, x_n)$ | $\in \mathbb{R}^{n+1}$ | $(x_0 = 1)$ |
| Couche cachée | : | $Z = (z_0, z_1, z_2, \dots, z_J)$ | $\in \mathbb{R}^{J+1}$ | $(z_0 = 1)$ |
| Couche de sortie | : | $Y = (y_1, y_2, \dots, y_L)$ | $\in \mathbb{R}^L$ | |
| Sortie désiré | : | $D = (d_1, d_2, \dots, d_L)$ | $\in \mathbb{R}^L$ | (apprentissage « supervisé ») |

Apprentissage d'un perceptron multi-couche

Tant qu'on n'a pas atteint le taux d'apprentissage désiré :

- ① on présente un exemple X
- ② on active la couche cachée \equiv on calcul Z
- ③ on active la couche de sortie \equiv on calcul Y
- ④ on détermine l'erreur, entre Y et D
- ⑤ on modifie les poids entre la couche cachée et la couche de sortie, en fonction de leur contribution à l'erreur
- ⑥ on modifie les poids entre la couche d'entrée et la couche cachée, en fonction de leur contribution à l'erreur

Choix de la fonction d'activation des couches cachées

Pour la couche cachée, par exemple :

- sigmoïde :

$$f_h(\text{net}) = \frac{1}{(1 + e^{-\lambda \text{net}})}$$

- tangente hyperbolique :

$$f_h(\text{net}) = \tanh(\beta \text{net})$$

avec :

- $\text{net} = \text{somme pondérée des entrées de la cellules} = \sum_{i=0}^n x_i w_{ji}$
- λ et β = paramètres de l'apprentissage (proches de 1)

Choix de la fonction d'activation de la couche de sortie

Pour la couche de sortie → **dépend de l'application** :

- **approximation, prévision...** ⇒ sortie réelle et proportionnelle, par exemple :

$$f_o(\text{net}) = \lambda \text{net}$$

- **classification, décision...** ⇒ sortie binaire, par exemple :

$$f_o(\text{net}) = \frac{1}{(1 + e^{-\lambda \text{net}})}$$

Remarque : il faut que la sortie désirée (D) soit dans l'intervalle des valeurs que peut prendre f_o ⇒ le cas échéant il faut normaliser

Fonction d'erreur

On veut apprendre m paires d'exemples : $\{X^k, D^k\} \ k = 1, \dots, m$

≡ on veut adapter les $J(n+1) + L(J+1)$ poids du réseau

afin de minimiser l'erreur sur l'ensemble des exemples

⇒ on a besoin d'une **fonction d'erreur** !

≡ une mesure de la distance entre D et Y ⇒ très dépendant de l'application

Exemple, erreur quadratique : $d(W) = \frac{1}{2} \sum_{l=1}^L (d_l - y_l)^2$

avec W représentant l'ensemble des poids du réseau

Modification des poids

- couche de sortie, on connaît la sortie désirée :

$$\Delta w_{lj} = w_{lj}^{new} - w_{lj}^{old} = -\rho_o \frac{\partial E}{\partial w_{lj}} = -\rho_o (d_l - y_l) f'_o(net_l) z_j$$

- couche(s) cachés(s) :

$$\Delta w_{ji} = -\rho_h \frac{\partial E}{\partial w_{ji}} = \rho_h \left[\sum_{l=1}^L (d_l - y_l) f'_o(net_l) w_{lj} \right] f'_h(net_j) x_i$$

Utilisation de la bibliothèque FANN

FANN is FUN !

Fast Artificial Neural Network (FANN) library is a **free open source neural network library**, which implements multilayer artificial neural networks in C with support for both fully connected and sparsely connected networks.

FANN includes a framework for easy handling of training data sets.

FANN is easy to use, versatile, well documented, and fast. Bindings to more than **20 programming languages** are available.

Quelques caractéristiques de FANN

- Multilayer Artificial Neural Network Library in C
- Backpropagation training
- Easy to use (create, train and run with just three functions)
- Cross-platform (linux and unix, windows)
- Open source, but can still be used in commercial applications
- Graphical Interfaces
- Bindings to a large number of programming languages
- Widely used (approximately 100 downloads a day)

La licence LGPL de FANN

Rappel sur la **GPL** : **GNU General Public License**

L'objectif est de garantir à l'utilisateur les droits suivants :

- ① liberté d'exécuter le logiciel, pour n'importe quel usage ;
- ② liberté d'étudier le programme et de l'adapter à ses besoins, ce qui passe par l'accès aux codes sources ;
- ③ liberté de redistribuer des copies ;
- ④ obligation de faire bénéficier à tous des modifications.

La LesserGPL (ou licence GPL « faible ») de FANN

La licence libre LGPL autorise à lier le programme à du code non GPL, sans pour autant révoquer la licence. Cette Licence LGPL permet donc de s'affranchir du caractère héréditaire de la GPL. C'est donc la clause de copyleft que n'a pas la LGPL.



- Permet à un logiciel propriétaire d'utiliser une librairie libre, sans devenir un logiciel libre ;
- toute modification de code source de la bibliothèque LGPL devra être également publiée sous la licence LGPL ;
- passage sous GPL par simple mise à jour des notifications.

FANN à partir de Python pour apprendre un XOR

On va apprendre la fonction XOR avec un MLP,

il nous faut une base d'apprentissage : fichier texte « xor.data » →
avec la première ligne comprenant :

- le nombre d'exemples (4)
- le nombre d'entrées (2)
- le nombre de sortie(s) (1)

puis les exemples : les entrées et à la ligne la sortie

Remarque : on a pris les entrées et sorties dans $[-1, 1]$
mais on aurait pu les prendre dans $[0, 1]$

xor.data
4 2 1
-1 -1
-1
-1 1
1
1 -1
1
1 1
-1

Paramètres du réseau

```
1 #!/usr/bin/python
2
3 from fann2 import libfann
4
5 connection_rate = 1
6 learning_rate = 0.7
7
8 input = 2    # nombre de cellules sur la couche d'entrée
9 hidden = 4   # nombre de cellules sur la couche cachée
10 output = 1  # nombre de cellules sur la couche de sortie
11
12 error = 0.0001      # erreur cible
13 iterations = 100000 # nombre d'interation maximum
14 reports = 10        # nombre d'interation entre deux affichages
15
```



Création du réseau et apprentissage

On crée le réseau (lignes 16 et 17), on fixe le taux d'apprentissage (18),
on choisit une fonction sigmoïde dans [-1,1] pour la cellule de la couche de sortie (19),
puis on apprend (ligne 21) et **on sauve l'architecture et les poids** (22) :

```
15
16 ann = libfann.neural_net()
17 ann.create_sparse_array(connection_rate, (input, hidden, output))
18 ann.set_learning_rate(learning_rate)
19 ann.set_activation_function_output(libfann.SIGMOID_SYMMETRIC_STEPWISE)
20
21 ann.train_on_file("xor.data", iterations, reports, error)
22 ann.save("xor.net")
```

Mise en œuvre du réseau et résultat

Dans un autre programme (ou le même) on peut charger le réseau et l'utiliser :

```
1 #!/usr/bin/python
2
3 from fann2 import libfann
4
5 ann = libfann.neural_net()
6 ann.create_from_file("xor.net")
7
8 print |ann.run([1, -1])|
```

```
dpuzenat@X220T:~/tmp$ python FANN-1.py
Max epochs 100000. Desired error: 0.0000100000.
Epochs      1. Current error: 0.2503619194. Bit fail 4.
Epochs      10. Current error: 0.2487698793. Bit fail 4.
Epochs      20. Current error: 0.1971800178. Bit fail 3.
Epochs      30. Current error: 0.1376865059. Bit fail 2.
Epochs      40. Current error: 0.0189153515. Bit fail 0.
Epochs      50. Current error: 0.0000819563. Bit fail 0.
Epochs      51. Current error: 0.0000000000. Bit fail 0.
dpuzenat@X220T:~/tmp$ python FANN-2.py
```

Et à présent en C :-)

```
1 #include "fann.h"
2
3 int main()
4 {
5     const unsigned int num_input = 2;
6     const unsigned int num_output = 1;
7     const unsigned int num_layers = 3;
8     const unsigned int num_neurons_hidden = 3;
9     const float desired_error = (const float) 0.001;
10    const unsigned int max_epochs = 500000;
11    const unsigned int epochs_between_reports = 1000;
12
13    struct fann *ann = fann_create_standard(num_layers, num_input,
14        num_neurons_hidden, num_output);
15
16    fann_set_activation_function_hidden(ann, FANN_SIGMOID_SYMMETRIC);
17    fann_set_activation_function_output(ann, FANN_SIGMOID_SYMMETRIC);
18
19    fann_train_on_file(ann, "xor.data", max_epochs,
20        epochs_between_reports, desired_error);
21
22    fann_save(ann, "xor_float.net");
23
24    fann_destroy(ann);
25
26    return 0;
27 }

1 #include <stdio.h>
2 #include "floatfann.h"
```



Et à présent en C :-)

Pour la compilation il faut lier aux bibliothèques **lfann** et **lm** :

```
dpuzenat@X220T:~/tmp$ gcc FANN-1.c -lfann -lm -o FANN-1
dpuzenat@X220T:~/tmp$ ./FANN-1
Max epochs      500000. Desired error: 0.0010000000.
Epochs           1. Current error: 0.2505110204. Bit fail 4.
Epochs           27. Current error: 0.0007874089. Bit fail 0.
dpuzenat@X220T:~/tmp$ gcc FANN-2.c -lfann -lm -o FANN-2
dpuzenat@X220T:~/tmp$ ./FANN-2
xor test (-1.000000,1.000000) -> 0.946073
dpuzenat@X220T:~/tmp$
```

Remarques : cette fois j'ai demandé moins de précision (et un affichage plus léger)

1

Définition de l'Intelligence Artificielle

- L'IA selon Wikipédia
- L'IA dans la littérature, le cinéma, et la recherche
- L'IA pour les jeux vidéos

2

Quelques algorithmes indispensables

- Recherche du plus court chemin : Dijkstra
- Recherche du plus court chemin : algorithme A*
- Flocking

3

Automates d'états fini

4

Réseaux de neurones artificielles

- Le neurone formel et le perceptron
- Apprentissage et Généralisation
- Le perceptron multi-couche et la rétro-propagation du gradient
- Application aux jeux vidéos, analyse du comportement, article de 2010

slides_ACHI.pdf — Behavior Analysis Through Games Using Artificial Neural Networks

1 sur 50 | 138,23% | 🔍 | ⌂

Sommaire

- ▼ Introduction
 - Instrumenting video games 3
 - Why video games ? 7
 - Getting valuable information on the player 12
- ▼ Artificial neural networks
 - How to go from the measurements to a conclusion 15
 - From game to reality 19
- ▼ Predicting children ages
 - Instrumenting games for children aged 2 to 10 23
 - Using a lowcost all-in-one touchscreen computer 28
 - Building the data base to train and test the ANN 31
 - Predicting children ages 32
 - Estimating the mental age of patients 38
- ▼ Instrumentation of an action game
 - ``SuperTuxKart'' car simulation 41
 - ANN dedicated to ``SuperTuxKart'' instrumentation 43
 - Experimental protocol 44
 - Results 46
 - Conclusion and future work 48

Introduction
Artificial neural networks
Predicting children ages
Instrumentation of an action game
Conclusion and future work

Behavior Analysis Through Games Using Artificial Neural Networks

Didier Puzenat
didier.puzenat@univ-ag.fr
GRIMAAg, Université Antilles Guyane, France

Isabelle Verlut
isabelle.verlut@chu-guadeloupe.fr
CHU de la Guadeloupe, France

11 février 2010

Didier Puzenat (UAG), Isabelle Verlut (CHU Guadeloupe) | Behavior Analysis Through Games Using ANN

