IA pour les Jeux
Licence Informatique et Vidéoludisme

Nicolas Jouandeau

n@up8.edu

2023

évaluation

- langages : C/C++ Racket Prolog Python
- évaluation : TPs | + DST | + PRJ (à définir)
- TPs en groupe de 2 à 4 (à définir)
  - réparti sur 2 ou 3 semaines
  - un rendu (programmes et rapport 1p) par groupe
  - objectif : comparer ou confronter les programmes
- ia de jeux de plateaux (échecs-like)
  - 1 joueur : taquin, sokoban
  - 2 joueurs en tour par tour : échecs, breakthrough
  - 2 joueurs à tours simultanés : roshambo, colonel blotto
  - `https://www.chessprogramming.org`
- ia de jeux de stratégie temps-réel (Starcraft-II)
  - 1 joueur avec une mini-mission
  - 2 joueurs opposés sur une carte
  - `https://github.com/deepmind/pysc2`

C/C++

- solution impérative avec `struct` et sans `class`
- avec conteneur C++ (`list`, `vector`, `unordered_map`)

Racket

- solution fonctionnelle

Prolog

- solution descriptive

Python

- apprentissage automatique (machine learning)
- avec NumPy, Keras et TensorFlow

Jeux à 1 joueur

- solution impérative, fonctionnelle, descriptive
- statistiques et temps de résolution

Jeux à 2 joueurs

- solution impérative, fonctionnelle, descriptive
- statistiques et victoires contre random ou mieux

Jeux de stratégie temps réel

- solution Python avec automate à états finis
- solution Python avec apprentissage automatique
- mini-mission : statistiques et temps de calcul
- deux joueurs : statistiques et victoires

Définition de l'IA

- compréhension et la construction d'entités intelligentes
- intelligence : processus de la pensée dont l'objectif est de percevoir, comprendre, prévoir, manipuler un monde plus étendu que soi-même
- objectif de l'IA : systématiser et automatiser les tâches intellectuelles

Deux définitions

- Penser et agir rationnellement (i.e. conformément à ses connaissances et à une fonction d'évaluation)
- Penser et agir comme un humain (i.e. conformément au test de Turing)

Domaines concernés

- les jeux
- et également la planification, la programmation d'agents et de systèmes autonomes, le diagnostic, la robotique, la compréhension des langages

Sociétés savantes, conférences et revues

- Association for the Advancement of Artificial Intelligence (AAAI), Special Interest Group on Artificial Intelligence (ACM-SIGAI), Association for Computational Linguistics (ACL), International Computer Games Association (ICGA)
- IJCAI, ECAI, ICML, NeurIPS, IEEE-TAAI, IEEE-CoG, TCIAIG, ACG, CG

A* : recherche de plus court chemin

- un état s = une position = un ensemble de pions sur un plateau
- p_i la position initiale
- p_f la position finale
- n itérations
- `free`(s) retourne vrai si s n'est pas en collision
- `nextMoves`(s) retourne la liste des mouvements en s
- `applyMove`(s, m) applique m sur s
- `dist`(a, b, c) estime la distance de a à c passant par b
- $\mathcal{H}$ la table de hashage des positions déjà évaluées $\mathcal{H}[s'] = s$ ssi $s' \leftarrow$ `applyMove` (s, m)
- $\mathcal{L}$ la liste des positions à évaluer

A* : recherche de plus court chemin

```
1  fonction astar ( p_i, p_f, n ) :
2      L ← { p_i } ;
3      H ← ∅ ;
4      for n times do
5          if |L| == 0 then break ;
6          s ← L.pop_front ( ) ;
7          if s == p_f then
8              return mkSolution ( H, s, p_i ) ;
9          M ← nextMoves ( s ) ;
10         for each m ∈ M do
11             s' ← applyMove ( s, m ) ;
12             if free (s') then
13                 if s' ∉ H then
14                     H[s'] ← s ;
15                     L ← L + s' ;                    // by distance order
16                 else
17                     if dist (p_i, H[s'], p_f) +1 > dist (p_i, s', p_f) then
18                         H[s'] ← s ;
19                         L ← L + s' ;                // by distance order
20     return ∅ ;
```

A* : retourner le chemin trouvé

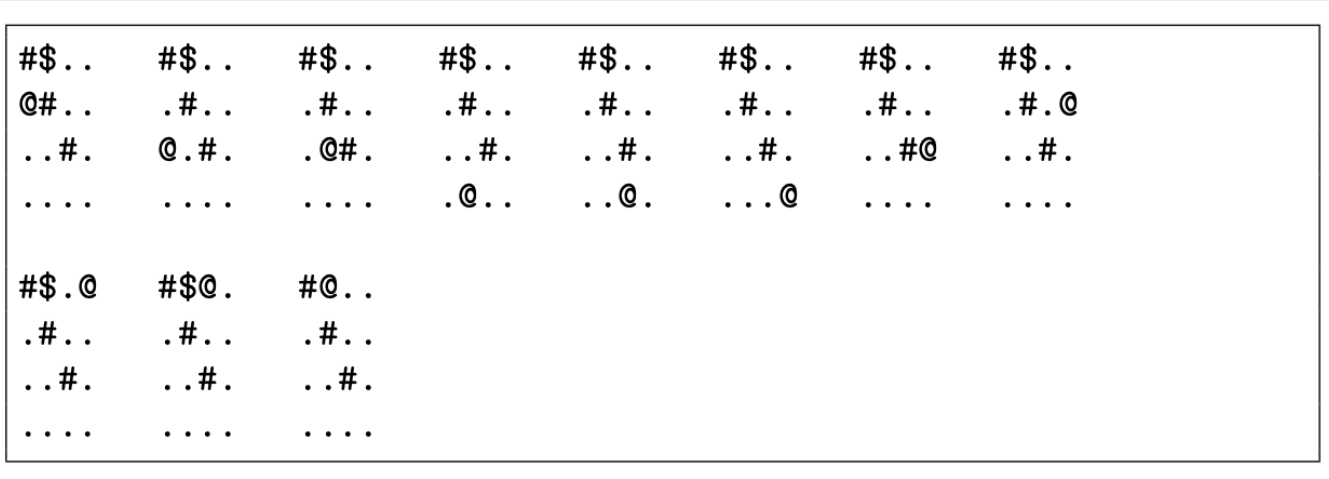
```
1 fonction mkSolution ( H, s, e ) :
2     S ← ∅ ;
3     while not-interrupted do
4         S.push_front ( s ) ;
5         if s == e then break ;
6         s ← H[s] ;
7     return S. ;
```

exemple de problème

- grille de 4x4 avec obstacles notés `#`
- un personnage `@` et un objectif `$`

```
#$..
@#..
..#.
....
```

A* : solution



- `drdrruuull`

chemins le plus long possible ?

- dans une grille de 4x4 ?
- dans une grille de 5x5 ?
- dans une grille de 6x6 ?

Depth Limited Search (DLS)

- recherche en profondeur limitée
  - jusqu'à un état *WIN*
  - interrompue sur un état *LOST*
  - interrompue à la profondeur *DLS_MAX_DEPTH*
- variables globales
  - *current_s* la position courante
  - $\mathcal{H}$ la table de hashage des profondeurs des états évaluées
  - *solved* un booléen à vrai si une solution est trouvée
- `applyMove`(*m*) joue/ajoute *m* sur *current_s*
- `undoMove`(*m*) déjoue/retire *m* sur *current_s*
- `applyMove`(*m*) et `undoMove`(*m*) modifient *current_s*

problème d'horizon de la fonction d'évaluation / profondeur

- évaluation à $DEPTH \neq$ évaluation à $(DEPTH + 1)$

DLS : recherche en profondeur limitée

```
1  fonction DLS ( d ) :
2      if solved then return ;
3      H[current_s] ← d ;
4      if h (current_s) < h (best_s) then
5          best_s ← current_s ;
6      if current_s == WIN then
7          solved ← true ;
8          return ;
9      if current_s == LOST or d == DLS_MAX_DEPTH then
10         return ;
11     M ← nextMoves ( ) ;
12     for each m ∈ M do
13         applyMove ( m ) ;
14         if current_s ∉ H or H[current_s] > d then
15             DLS ( d + 1 ) ;
16         undoMove ( m ) ;
17         if solved then return ;
```

DLS : recherche en profondeur limitée

- sans état global *current_s*
- avec un état local *s*

```
1  fonction DLS ( s, d ) :
2      if solved then return ;
3      H[s] ← d ;
4      if h (best_s) > h (s) then
5          best_s ← s ;
6      if s == WIN then
7          solved ← true ;
8          return ;
9      if s == LOST or d == DLS_MAX_DEPTH then
10         return ;
11     M ← nextMoves ( s ) ;
12     for each m ∈ M do
13         s' ← applyMove ( s, m) ;
14         if s' ∉ H or H[s'] > d then
15             DLS ( s', d + 1 ) ;
16         if solved then break ;
```

DLS : recherche en profondeur limitée

- avec la solution : *solution_size* et *best_s*

```
1  fonction DLS ( s, d ) :
2      if solution_size ≠ 0 then return ;
3      H[s] ← d ;
4      if h (best_s) > h (s) then
5          best_s ← s ;
6      if s == WIN then
7          solution_size ← d ;
8          return ;
9      if s == LOST or d == DLS_MAX_DEPTH then
10         return ;
11     M ← nextMoves ( s ) ;
12     for each m ∈ M do
13         s' ← applyMove ( s, m) ;
14         if s' ∉ H or H[s'] > d then
15             solution [ d ] ← m ;
16             DLS ( s', d + 1 ) ;
17         if solution_size ≠ 0 then break ;
```

DLS : initialiser la recherche

- fixer la profondeur max
- vider la table des états déjà évalués
- initialiser la taille de la solution
- définir *best_s* avec l'état courant
- lancer la recherche en profondeur limitée

```
1 DLS_MAX_DEPTH ← 10 ;
2 H ← ∅ ;
3 solution_size ← 0 ;
4 best_s ← p ;
5 DLS ( p, 0 ) ;
```

Breadth First Search (BFS)

- recherche en largeur d'abord (limitée en profondeur)
  - jusqu'à un état *WIN*
  - interrompue sur un état *LOST*
  - interrompue à la profondeur *BFS_MAX_DEPTH*
- variables globales
  - *solution_state* l'état correspondant à la victoire trouvée
  - *best_s* la meilleure solution trouvée
- variables locales
  - $\mathcal{L}$ la liste des états à évaluer
  - d la profondeur courante
  - $\mathcal{L}'$ la liste des fils de $\mathcal{L}$

Breadth First Search (BFS)

```
1  fonction BFS ( L, d ) :
2      if solution_state ≠ 0 then return ;
3      S ← ∅ ;
4      for each s ∈ L do
5          M ← nextMoves ( s ) ;
6          for each m ∈ M do
7              s' ← applyMove ( s, m ) ;
8              S ← S + (s', m) ;
9      L' ← ∅ ;
10     for each (s, m) ∈ S do
11         if s ∉ H then
12             H[s] ← m ;
13             if h (best_s) > h (s) then best_s ← s ;
14             if s == WIN then
15                 solution_state ← s ;
16                 return ;
17             if s == LOST or d == BFS_MAX_DEPTH then
18                 return ;
19             L' ← L' + s ;
20     BFS ( L', d + 1 ) ;
```

défaut de DLS

- première solution possiblement sous-optimale
- rechercher toutes les solutions pour obtenir la solution optimale

défaut de BFS

- première solution = solution optimale
- coût du calcul et du stockage de $\mathcal{L}$

Iterative Deepening Search (IDS)

- recherche DLS à profondeur itérative
- première solution = solution optimale
- pas de calcul et de stockage de $\mathcal{L}$

```
1 fonction IDS ( p ) :
2     solution_size ← 0 ;
3     best_s ← p ;
4     for depth ∈ [1; IDS_MAX_DEPTH ] do
5         H ← ∅ ;
6         DLS_MAX_DEPTH ← depth ;
7         DLS ( p , 0 ) ;
8         if solution_size ≠ 0 then break ;
```