TP Algo2 – session 10 – Projet-tournoi

16 avril 2025

Objectif d'apprentissage : écrire un programme qui joue à un jeu de poursuite/évasion en utilisant différentes approches dont la recherche minimax. Trois séances préparatoires sont prévues et un temps personnel permettra de finaliser le projet-tournoi.

1 Présentation du jeu

Robin des bois et ses joyeux compagnons ont préparé un plan pour subtiliser la fortune du shérif de Nottingham en plein coeur de cette ville. Seulement celui-ci, bien préparé, leur a tendu un piège en fermant toutes les portes de la cité dès leur arrivée. Les voilà à devoir cavaler à travers la ville en évitant les soldats du shérif jusqu'à l'arrivée des renforts, Petit-jean et Frère Tuck, qui leur ouvriront un passage secret pour les exflitrer. Tiendront-ils jusqu'à temps ?

De nombreux jeux de sociétés reprennent cette mécanique. On peut citer "Scotland Yard" de 1983, et "Mister X", une modernisation de 2009. Dans le domaine scientifique, de nombreuses variantes ont été considérées en tant que jeu de poursuite-évasion (*pursuit-evasion game* ou *cops and robbers game*).

2 Règles du jeu

Dans ce jeu, on considère qu'un ou plusieurs gendarmes cherchent à capturer un ou plusieurs voleurs sur un plateau. Un plateau consiste en un graphe non orienté, les gendarmes et les voleurs pouvant se positionner sur n'importe quel sommet. Au début, les gendarmes commencent par se positionner, puis c'est au tour des voleurs. Ensuite, on réalisera les déplacements des gendarmes et des voleurs tour par tour. Un déplacement vers un nouveau sommet n'est permis que si ce sommet est adjacent au sommet actuel. Il n'est pas obligatoire d'effectuer un déplacement : chaque personnage peut rester sur place lors de son tour. L'ensemble des personnages du même type se déplacent en même temps lorsque c'est à leur tour : on déplace tous les gendarmes, puis tous les voleurs, puis tous les gendarmes, etc. Ce sont les gendarmes qui commencent. Lorsqu'un gendarme et un voleur sont sur le même sommet, le gendarme capture le voleur et ce dernier est donc éliminé du jeu. Le jeu s'arrête lorsque tous les voleurs ont été capturés ou lorsque le nombre de tours maximum est atteint. Dans le premier cas, les gendarmes gagnent. S'il reste au moins un voleur à la fin, ce sont les voleurs qui gagnent.

3 Interface du programme

La structure du programme étendra celle utilisée dans les séances de TP précédentes :

`algo.h` Ce fichier fixera quelques définitions de fonctions pour orienter certains algorithmes qui pourront être utilisés ou non dans l'implémentation finale. Il ne doit pas être modifié.

`algo.c` Ce fichier doit contenir les implémentations des algorithmes et sera à compléter.

`algo_tests.c` Ce fichier contient quelques tests, mais devra être complété pour s'assurer de la validité des fonctions implémentées.

`Makefile` Ce fichier automatise les opérations d'indentation, de compilation et de test. Il ne doit par être modifié.

`game.c` Ce dernier fichier est nouveau : c'est lui qui constituera le programme principal et qui devra respecter une interface précise en terme d'arguments et d'entrées-sorties. Il faudra l'adapter à partir de la version partielle déjà fournie.

`server.py` Ce dernier fichier est également nouveau : c'est lui qui permettra de lancer un programme contre un autre, chacun endossant soit le rôle des gendarmes, soit celui des voleurs. Il ne doit par être modifié.

À l'issue de la compilation des fichiers `algo.c` et `algo_tests.c`, on obtiendra un programme de tests unitaires similaire à ce qui était fait aux précédentes séances. Lors de la compilation des fichiers `algo.c` et `game.c`, on obtiendra un programme permettant de jouer au jeu présenté ci-dessus et devant respecter les contraintes suivantes :

Arguments le premier argument est le nom d'un fichier représentant un plateau de jeu et le second indique si le programme prend le rôle des gendarmes (0) ou des voleurs (1) ;

Entrée standard les positions du joueur adverse seront saisies ;

Sortie standard les positions du joueur piloté par le programme seront affichées.

L'ordre des positions doit toujours être le même : il faudra toujours commencer par la position du même gendarme et poursuivre en utilisant le même ordre. Lorsqu'un voleur est attrapé par un gendarme, il faudra ignorer sa position lors des tours qui suivent. Le programme doit s'arrêter si tous les voleurs ont été attrapés ou si le nombre de tours maximum est atteint.

Cette interface ne changera pas dans les séances suivantes. L'implémentation partielle fournie dans l'archive (`game.c`) la respecte avec une stratégie naïve (sur-place) mais dépend des fonctions implémentées dans `algo.c` pour fonctionner correctement. Le binaire `low` la respecte complètement en utilisant une stratégie inefficace (déplacement aléatoire).

Par contre, le fichier `algo.h` sera modifié à la seconde séance. Il est d'ailleurs possible de demander des modifications spécifiques avant cette seconde séance. Elles seront prises en compte si elles s'intègrent dans la logique du projet.

4 Exemple d'exécution

On considère un exemple avec 1 gendarme, 2 voleurs et un plateau avec 4 sommets formant un carré.
Voici le fichier `input.txt` représentant le plateau :

```
Cops: 1
Robbers: 2
Max turn: 3
Vertices: 4
0 0
0 1
1 1
1 0
Edges: 4
0 1
1 2
2 3
3 0
```

Pour le gendarme, on lancera le programme A avec `./game input.txt 0`. Pour le voleur, on lancera le programme B avec `./game input.txt 1`.

Le programme A affiche la position initiale du gendarme sur la sortie standard : 0. Le programme B lit cette position initiale puis affiche celles des voleurs : 1 et 2. On commence alors le premier tour de jeu avec le déplacement du gendarme qui va sur le sommet 3. Au second tour de jeu, ce seront les voleurs qui vont en 0 (déplacement du premier voleur) et 2 (sur-place pour le second voleur). Au troisième tour (le dernier), le gendarme se déplace en 2 et capture l'un des voleurs. Le programme se termine par la victoire de l'équipe des voleurs.

Voici les entrées-sorties :

Sortie de A → entrée de B	Sortie de B → entrée de A
0	
	1
	2
3	
	0
	2
2	

Pour faire jouer 2 programmes l'un contre l'autre, on pourra utiliser le serveur `server.py` ainsi :

```
python server.py ./low ./low input.txt 1
```

5 Objectif global du tournoi et critères d'évaluation

L'évaluation portera sur 3 points :
— la validité des algorithmes dans `algo.c` (6 points) ;
— la validité du programme `game.c` (6 points) ;
— la performance de la stratégie implémentée (8 points).

Des points bonus sont prévus pour ceux et celles qui participent au tournoi (c'est-à-dire les personnes présentes lors de la séance de restitution).

L'évaluation de la validité se fera sur la même base que pour les TP : avertissements du compilateur, validité des accès mémoires, absence de fuite mémoire, validité des tests et respect des conventions de codage. En particulier, le projet-tournoi sera évalué sur la base de certaines contraintes fortes :
— le respect de l'échéance ;
— la soumission des fichiers `algo.c`, `algo_tests.c` et `game.c` **tels quels** (non-renommés, non-archivés, non-compréssés) ;
— il ne faut pas modifier `algo.h` ;
— des fichiers qui compilent (il faut une implémentation pour chacune des fonctions du `.h`, même si c'est un squelette) ;
— l'absence de plagiat.

Des pénalités significatives seront appliquées dans les cas suivants :
— avertissements du compilateur `gcc` ;
— souci d'indentation : `indent` version 2.2.12 ou 2.2.13.

Des pénalités seront appliquées pour chaque test :
— accès mémoire valide (sinon le test est considéré comme échoué) ;
— fuite mémoire (en fonction de l'ampleur de la fuite).

6 Considération sur l'utilisation des IA

L'utilisation d'une quelconque forme d'IA est interdite.

Elle n'est pas particulièrement efficace pour faciliter l'apprentissage comme le confirme les résultats au DS 1 : les personnes suspectées de s'en être servie sont celles qui ont eu les plus mauvais scores à l'exercice qui portait sur le mini-projet.

Par ailleurs, la technologie n'est pas suffisamment mature pour s'y appuyer aveuglement. Ce qu'elle produit n'est pas toujours pertinent et nécessite un regard expert pour vérification.

Espérons que cela n'évolue pas trop vite car sinon, on pourra fermer la formation et il vous faudra trouver un nouveau travail. Ce qui pourrait arriver d'ici quelques années, c'est qu'un informaticien dotée d'une IA puisse en remplacer plusieurs, ce qui rendrait le marché de l'emploi plus concurrentiel. Mais on n'est pas à l'abri d'un effet rebond.

Une technologie cool mais avec des conséquences pas si positives que ça (bilan qui s'aggrave si on prend le coût environnemental et les autres coûts cachés).

7 Première séance

À l'issue de la première semaine, l'incrément suivant est attendu :
— découvrir le sujet et l'archive fournie ;
— implémenter les algorithmes élémentaires associés à `algo.h`, notamment la lecture de fichier et l'algorithme de Floyd-Warshall ;
— bien tester ces algorithmes.

Le parcours en largeur est un parcours qui permet d'obtenir le plus petit nombre d'arêtes qui séparent un sommet donné de tous les autres. Cependant, il existe un autre algorithme qui permet d'obtenir plus efficacement l'ensemble des distances qui séparent chaque paire de sommets : l'algorithme de Floyd-Warshall. Celui-ci considère que les sommets sont indicés de 1 à $|V|$ et il calcule la distance et le prochain sommet à emprunter pour aller du sommet i au sommet j.

FLOYD-WARSHALL(V, E)

pour $u = 1$ **à** $|V|$ **faire**
  pour $v = 1$ **à** $|V|$ **faire**
    $D[u, v] \leftarrow \infty$
    $N[u, v] \leftarrow 0$
pour $(u, v) \in E$ **faire**
  $D[u, v] \leftarrow 1$
  $N[u, v] \leftarrow v$
pour $v = 1$ **à** $|V|$ **faire**
  $D[v, v] \leftarrow 0$
  $N[v, v] \leftarrow v$
pour $w = 1$ **à** $|V|$ **faire**
  pour $u = 1$ **à** $|V|$ **faire**
    pour $v = 1$ **à** $|V|$ **faire**
      si $D[u, v] > D[u, w] + D[w, v]$ **alors**
        $D[u, v] \leftarrow D[u, w] + D[w, v]$
        $N[u, v] \leftarrow N[u, w]$
retourner D, N

Cet algorithme pourra être utilisé ou non pour déterminer les placements initiaux, puis les déplacements.

Une évaluation facultative blanche vérifiera ces premiers éléments. Elle est vivement recommandée quelque soit l'état d'avancement. Il faudra soumettre trois fichiers : `algo.c` contenant les implémentations, `algo_tests.c` contenant les tests unitaires (à compléter pour chaque méthode) et `game.c`. Si une fonction auxiliaire paraît utile pour les tests, c'est dans le fichier de tests qu'il faudra la rajouter.

Les trois fichiers doivent être soumis **tels quels** pour permettre le traitement et l'évaluation automatique (non compressés, non archivés, non renommés).

8 Séances suivantes

La seconde séance sera consacrée à l'imlémentation de la stratégie minimax. Celle-ci aussi pourra être ou non utilisée dans le programme final.

La dernière séance sera laissée libre. Il est cependant recommandé de commencer à travailler sur le fichier `game.c` sur son temps libre dès la fin de la première séance (il restera alors 6 semaines avant l'échéance) et de ne pas attendre la troisième séance qui sera à une semaine de l'échéance. Un bon point de départ consiste à simplement réaliser un programme où les gendarmes se déplacent vers les voleurs et où les voleurs se déplacent en évitant les gendarmes. Cette stratégie permettra de battre le binaire `low` qui devrait être l'objectif entre la première séance et la seconde séance.