Plus court chemin dans une grille avec $A^\star$

I - Tracé de grille

Conseils généraux :

— Commencer par rappeler les consignes et les points forts du rapport du jury sur l'épreuve orale de Mines-Pont (explication du compte-rendu, interactions nécessaires avec l'examinateur, etc.).

— On peut remplacer `ocaml` par `utop` dans leur ligne de compilation. Adapter la ligne pour les élèves.

— Donner le `graphics.cma` s'il n'est pas présent.

— Si vraiment besoin, préparer des fichiers de code et les donner aux élèves pour les débloquer.

1) Fonction `construire_grille`

- Attention à garder le code lisible (demander éventuellement une ré-écriture). Par exemple, séparer des variables numérateur, dénominateur, etc.
- Attention à faire une division flottante et non entière (fonctions de conversion à connaître).
- Pointer les élèves vers la documentation du module Random (utilisé en cours). Si vraiment besoin, rappeler l'existence de `Random.float` et discuter de la façon de faire une loi de Bernoulli avec cette fonction.

2) Fonction `tracer_grille`

- Aide : Pointer les élèves vers la documentation de Graphics (https://ocaml.github.io/graphics/graphics/Graphics/index.html par exemple). Si besoin, prendre du temps à leur expliquer.
- **Discussion :** Que se passe-t-il quand tu fermes la fenêtre graphique avec la souris ? Y a-t-il une fonction qui permet de fermer proprement la fenêtre ?
  –> Ne pas fermer la fenêtre avec la souris (erreur fatale) mais avec la fonction `close_graph` dans ocaml/utop.
- Aide : Expliquer le principe d'une fenêtre graphique. On ouvre d'abord une fenêtre dans laquelle on vient tracer des choses/objets. La fenêtre est graduée pixel par pixel, et son origine se trouve en bas à gauche (donc ⚠ça ne ressemble pas du tout à une matrice mathématique qui aurait son origine en haut à gauche et l'abscisse vers le bas). Toutes les coordonnées et les longueurs sont données en pixel.
  Discussion : Y a-t-il une fonction dans le module qui permet de tracer des rectangles, par exemple ? Peux-tu commencer par tracer uniquement un grand rectancle noir à l'écran dans la fenêtre graphique ?
- **Discussion :** combien de pixels y a-t-il sur un écran d'ordinateur moderne ?
  –> Résolution 1920 x 1080 en général, d'autres formats similaires existent, ce qui importe est l'ordre de grandeur.
- Aide : Pointer les fonctions `open_graph` , `fill_rect` , et `set_color` (et les couleurs prédéfinies) si besoin, ainsi que `resize_window` qui sert au moins pour la question suivante.

3) Test demandé

- **Discussion :** Combien de carré de taille k peut-on faire rentrer dans la taille de ta fenêtre graphique ? Quelle est cette taille ? Y a-t-il une fonction qui permet de fixer cette taille (à une valeur qui sera donc connue) ?
  –> `resize_window`
- Aide : Ecrire une fonction permettant de calculer ce k à part ?

II - Dépalcements horizontaux et verticaux

1) Fonction `existe_chemin`

— **Discussion :** Connais-tu un algorithme permettant de trouver un chemin entre deux sommets dans un graphe ? De tester l'accessibilité d'un sommet depuis un autre ? Comment représenter la grille comme un graphe ? Quels sont les sommets, les arêtes ? (Lisez bien le paragraphe du sujet qui donne un indice…)
–> Parcours de graphe ! N'importe lequel fonctionne ici !

— **Discussion :** Comment parcourir les voisins d'un sommet du graphe (i.e. une case de la grille) ?
–> Aide : Ecrire une fonction `calcule_voisins` à part qui prend en argument une grille et une case et renvoie la liste des voisins de cette case dans la grille. (Par exemple, avec des références de liste, ou en filtrant la liste des voisins potentiels...)

— Aide : Lisez les questions suivantes. Quel type de parcours pourrait vous aider pour la suite ? (Bon en l'occurrence il faudra recoder le A* mais pour se mettre en jambe, un parcours en largeur peut être préférable...)

— *Demande examinateur :* Si un.e élève bloque trop, lui demander d'écrire un pseudo-code de parcours de son choix. S'iel n'y arrive pas, lui donner le pseudo-code d'un parcours générique (en annexe) et pointer vers le module `Queue` de OCaml.

2) Structure de donnée pour $A^\star$

Rien à signaler. La réponse est "une file de priorité", les élèves doivent le savoir.

3) Implémentation d'une file de priorité

On attend une implémentation des files de priorité par tas, et on pénalise toute implémentation moins efficace. Citons les remarques du concours : "Les meilleurs candidats peuvent identifier une alternative pertinente pour cette partie, par exemple une liste de listes de sommets de même distance.".

— **Discussions :** Quelle implémentation utiliser pour représenter une file de priorité, concrètement ? Qu'est ce qu'un tas et comment l'implémenter (faire un schéma) ? Où sont les fils du noeud i ? Où est le parent du noeud i ?
–> Un tableau qu'on remplit au fur et à mesure, en retenant sa longueur actuel. Si besoin, redonner le type suivant :

OCaml
```
type sommet = int*int (* les cases de notre grille *)

type tas_naif = {contenu : (sommet * float) array; mutable len : int}
```

— **Discussions :** Quelles sont les opérations nécessaires sur une file de priorité, pour l'algorithme $A^\star$? Pour chacune d'entre elle, rappeler le principe de l'algorithme et donner sa complexité (faire des schémas).
  — Création de tas vide. `creer_file : int -> tas` .
  — Insertion. Par exemple `insere : tas -> sommet -> float -> unit` tel que `insere file case prio` insère `(case, prio)` dans le tas, à l'aide d'une percolation vers le haut.
  — Extraction de l'élément de plus faible priorité. Par exemple `extrait_min : tas -> sommet` tel que `extrait_mine file` retire et renvoie l'élément de plus faible priorité du tas (à la racine, donc), et rétablit la structure de tas à l'aide d'une percolation vers le bas (du dernier élément mis à la racine).
  — Diminuer la priorité d'un élément. –> garder les discussions pour la question suivante.

  Ne pas hésiter à écrire des fonctions utiles intermédiaires !!

— Aide : Attention, votre tas doit contenir des sommets et des priorités, la propriété de tas-min porte uniquement sur les priorités (ne comparez pas bêtement).

4) Implémentation de $A^\star$

- **Discussion :** Ecrire un pseudo-code de l'algorithme $A^\star$. Si l'élève en est incapable (lui laisser du temps), donner le pseudo-code de $A^\star$ en annexe.
- Aides : Ecrire une fonction `reconstruit_chemin` à part, qui prend en argument une matrice des parents (par exemple) et le sommet destination ; et qui renvoie la liste des cases à suivre pour atteindre la destination.
- **Discussion :** Peut-on améliorer la condition d'arrêt pour éviter de parcourir au delà de la destination ?

5) Tests et heuristiques

Rien à signaler sur le code. Aider au cas par cas.

5. **a.** On obtient l'algorithme de Dijkstra (c'est du cours). On parcourt exactement les sommets par distance croissante depuis le sommet de départ.

b. On pourrait penser à la distance "à vol d'oiseau" (euclidienne), mais ce serait ici une erreur, on ne pourra jamais (ou presque) espérer atteindre la distance euclidienne. Une meilleure borne serait ici la distance "Manhattan" : $|x_2 - x_1| + |y_2 - y_1|$, i.e. les sommes des différences horizontales et verticales entre les deux cases (x_1, y_1) et (x_2, y_2). Au mieux, on peut se diriger de la case 1 à la case 2 "sans détour".

c. Les indications profs disent "distance Manhattan multipliée par un flottant > 1". Il s'agit de renforcer encore davantage le biais de la distance Manhattan en leur donnant plus de poids que les distances déjà parcourues (les cases sont examinées par ordre croissant de `distance parcourue + estimation`).

On constate (notamment grâce à la question suivante) que la distance Manhattan va tenter plein de chemins "proches" en parallèle dès que l'un d'eux se heurte à un petit obstacle, et perdre beaucoup de temps en explorations parallèles. Pour éviter cela et "forcer" à explorer en piorité un chemin déjà commencé et bien avancé, on peut tenter de gonfler l'importance de la partie "chemin restant" par rapport à la partie "chemin parcouru" (le chemin en court est celui qui est le plus proche de l'arrivée). On augmente donc l'heuristique par rapport à la distance, par exemple en la multipliant par un flottant > 1.

On accepte aussi que les élèves proposent de favoriser les bords sur le centre pour rencontrer moins d'obstacles, mais de telles heuristiques ne sont pas attendues (–> ne pas perdre de temps à les implémenter).

6) Affichage graphique pas à pas

- **Discussions :** Quand faut-il colorer une case en cyan ? En bleu ? En jaune ? (dans le code)
- Aide : Tracer le chemin en bleu, attendre, puis retracer le même chemin en jaune pour l'"effacer".
- **Discussions :** Que constatez-vous ? Quelles différences font les heuristiques ? Expliquer.

7) Complexité spatiale linéaire

- **Discussion :** Quel est l'espace utilisé par nos files de priorité ? Comment faire pour ne réserver que l'espace qu'on aura besoin d'utiliser ?

  –> tableau dynamique, ajouter un doublement à l'insertion si le tableau est plein (rendre le contenu mutable).
- Aide : Utiliser des tables de hachage pour les distances et les parents au lieu de pré-remplir des matrices/tableaux de taille $n \times m$.

III - Déplacements de direction quelconque

Partie très avancée, aider au cas par cas. Cf fichier `a_star.ml` en cas de besoin (commentaires détaillés).

IV - Coloriage de composantes connexes

Partie non traitée par ce guide-examinateur.

Annexe : à donner aux élèves en difficulté (à découper)

Pseudo-code d'un parcours générique (itératif) :

Pseudo-code

```
Entrées : G = (S, A) un graphe, s_0 ∈ S un sommet de G //ici : G est la grille
Effet : parcours de G en partant de s_0
(traitement des sommets dans la composante connexe de s_0)

PARCOURS(G, s_0) :
  a_traiter ← {s_0} //ici, a_traiter peut être une file donnée par le module Queue
  deja_vu ← [| Faux; ...; Faux |] //une case par sommet
  deja_vu[s_0] ← Vrai
  Tant que a_traiter ≠ ∅ Faire :
    s ← un élément de a_traiter //que l'on retire de a_traiter
    Visiter(s) //dans notre cas : vérifier si s est le sommet qu'on cherche à atteindre
    Pour chaque voisin v de s tel que deja_vu[v] = Faux Faire :
      a_traiter ← a_traiter ∪{v}
      deja_vu[v] ← Vrai
```

Pseudo-code de $A^\star$:

Pseudo-code

```
Entrées : G = (S, A, ρ) un graphe pondéré, une heuristique h,
un sommet source et un sommet destination.
Sortie : Un chemin (de poids minimal ?) de source à but, si il en existe un.
A*(G, s):
    dist[t] ← [∞, ..., ∞]
    dist[source] ← 0
    parents ← (Nil, Nil, ... Nil)
    ouverts ← FilePrioVide()
    Insérer source dans ouverts avec la priorité h(source)
    Tantque ouverts ≠ ∅ Faire :
        (u, _) ← ExtraireMin(ouverts)
        Si u = destination alors :
            Renvoyer le chemin trouvé en remontant les pères de destination à source.
        Pour chaque v successeur de u Faire :
            d ← dist[u] + ρ(u → v)
            Si d < dist[v] Alors :
                parents[v] ← u
                dist[v] ← d
                Si v ∈ ouverts Alors :
                    Remplacer la priorité de v dans ouverts par d + h(v)
                Sinon :
                    Insérer v dans ouverts avec la priorité d + h(v)
    Renvoyer ∅
```

Pseudo-code

```
Entrées : G = (S, A, ρ) un graphe pondéré, une heuristique h,
un sommet source et un sommet destination.
Sortie : Un chemin (de poids minimal ?) de source à but, si il en existe un.
A*(G, s):
    dist[t] ← [∞, ..., ∞]
    dist[source] ← 0
    parents ← (Nil, Nil, ... Nil)
    ouverts ← FilePrioVide()
    Insérer source dans ouverts avec la priorité h(source)
    Tantque ouverts ≠ ∅ Faire :
        (u, _) ← ExtraireMin(ouverts)
        Si u = destination alors :
            Renvoyer le chemin trouvé en remontant les pères de destination à source.
        Pour chaque v successeur de u Faire :
            d ← dist[u] + ρ(u → v)
            Si d < dist[v] Alors :
                parents[v] ← u
                dist[v] ← d
                Si v ∈ ouverts Alors :
                    Remplacer la priorité de v dans ouverts par d + h(v)
                Sinon :
                    Insérer v dans ouverts avec la priorité d + h(v)
    Renvoyer ∅
```