TP numéro 5

Intelligence artificielle, ENSIIE

Semestre 4 en avance, 2024–25

Votre code devra être déposé avant le 29 janvier 2025 (inclus) sur `exam.ensiie.fr` dans le dépôt `ia-fisa-tp5`.

L'objectif de ce TP est de proposer une intelligence artificielle pour résoudre le jeu Wordle (cf. par exemple une version française à l'adresse `https://wordle.louan.me/`). Le but de ce jeu est de deviner un mot de cinq lettres en moins de six essais. Pour chaque essai, chaque lettre proposée est coloriée par rapport à la solution :

- si la lettre proposée est présente dans la solution à la même place, elle est coloriée en vert ;
- si la lettre proposée est présente dans la solution mais à une place différente, elle est coloriée en orange ; une lettre de la solution ne peut être utilisée qu'une seule fois pour colorier en vert ou en orange les lettres de la proposition ;
- sinon, la lettre est coloriée en noir.

Par exemple, avec la proposition E L E V E et la solution P E L L E on aura la coloration E L E V E. On peut donc déduire de la coloration qu'il y a un E dans la solution en dernière position, qu'il y a un deuxième E qui n'est ni en première ni en troisième position, qu'il n'y a pas de troisième E, qu'il y a (au moins) un L qui n'est pas en deuxième position et qu'il n'y a pas de V. Bien entendu, la solution n'est a priori pas connue du joueur.

Dans tout ce travail, nous considérerons des mots de taille quelconque et pas nécessairement 5, de façon à rester générique. Les mots seront représentés par des listes de lettres, une lettre étant la constante formée de cette lettre. On aura donc par exemple `[e, l, e, v, e]`. Le prédicat `word/1` indiquera qu'un mot est valide, c'est-à-dire qu'il est contenu dans un dictionnaire. On trouvera dans le fichier `word.pl` la définition de ce prédicat pour les 7980 mots valides de cinq lettres.

Exercice 1 : Coloration

Avant d'écrire l'intelligence artificielle permettant d'essayer de trouver la solution, nous allons d'abord écrire un programme permettant de colorier une proposition, en connaissant la solution. Il s'agit donc de définir un prédicat `color(P, S, C)` tel que si `P` est un mot constituant une proposition, `S` est la solution qui est un mot supposé de même longueur, alors `C` est une liste de couleurs de même longueur que `P` et `S` qui représente les couleurs attribuées à chacune des lettres de `P`, dans le même ordre. Les couleurs seront données par les constantes `black`, `green` et `orange`. Par exemple, on aura

`color([e, l, e, v, e], [p, e, l, l, e], [orange, orange, black, black, green])`.
Pour définir ce prédicat, nous allons procéder par plusieurs étapes.

1. Définir un prédicat `availables(P, S, A)` tel que `A` contient la liste des lettres de la solution `S` qui n'ont pas été trouvé à la bonne place dans la proposition `P`. On aura par exemple `availables([e, l, e, v, e], [p, e, l, l, e], [p, e, l, l])`.
2. Définir un prédicat `mem_remove(X, L, Q)` tel que `Q` soit la liste `L` dans laquelle on a retirée la première occurrence de `X`. Le prédicat ne sera pas défini si `X` n'est pas dans `L`. On aura par exemple `mem_remove(a, [b,a,c,a,d], [b,c,a,d])`.
3. Définir un prédicat `color_with_availables(P, S, A, C)` tel que `C` est la coloration de la proposition `P` pour la solution `S`, en utilisant `A` pour savoir si on peut colorier en orange.

   On utilisera une définition récursive :

   — si `P` et `S` commencent par la même lettre, on colorie récursivement le reste des mots et on ajoute un vert devant ;

   — sinon, si `P` commence par une lettre `X` disponible dans `A`, on colorie récursivement le reste des mots en supprimant `X` de `A`, et on rajoute un orange devant ;

   — sinon on colorie en noir.
4. Définir le prédicat `color/3` en utilisant `availables/3` et `color_with_availables/4`.

Exercice 2 : Suggestion de propositions

Le but de cette partie est d'écrire un prédicat qui suggère des propositions en fonction des propositions précédentes et de leur coloration. On ne suppose donc plus qu'on connaît la solution. Ici aussi on va y aller par étapes. L'idée générale est qu'on va considérer n'importe quel mot valide (de façon non-déterministe), puis on va vérifier que ce mot est en accord avec les contraintes données par les colorations des propositions précédentes.

5. Définir un prédicat `check_colors(P, O, C)` qui est vrai si pour chaque position, les lettres de la proposition `P` sont en accord avec la coloration `C` de l'ancienne proposition `O`. Il faudra donc que ce soit la même lettre que celle de `O` si la couleur est `green`, et une lettre différente si c'est `orange` ou `black`.

Ce prédicat ne suffit pas pour exploiter complètement les contraintes de la coloration : si une lettre est coloriée en orange, elle doit apparaître quelque part dans le mot ; si elle est coloriée en noir, elle ne peut apparaître nulle part. On va donc construire deux listes : celles des lettres qui doivent apparaître dans le mot, et celles des lettres qui ne doivent pas (ou ne doivent plus dans le cas d'une lettre colorié en orange et en noir à deux positions différentes) apparaître.

6. Définir un prédicat `must_be_present(O, C, A)` tel que `A` contient la liste des lettres de `O` qui sont associées à la couleur `orange` dans `C`. Par exemple on aura `must_be_present([b,b,c,b,b],[green, orange, orange, orange, black], [b,c,b])`.

7. Définir un prédicat `forbidden(O, C, F)` tel que `F` contient la liste des lettres de `O` qui sont associées à la couleur `black` dans `C`. Par exemple on aura `forbidden([b,b,c,b,b],[green, orange, orange, orange, black], [b]).`
8. Définir un prédicat `remove_green(P, C, R)` tel que `R` est la liste des lettres de `P` qui ne sont pas à la même position que la couleur `green` dans la coloration `C`. Par exemple on aura
`remove_green([a,b,c,d,e], [black, green, orange, green, black], [a,c,e]).`
9. Définir un prédicat `check_presence_and_forbidden(P, A, F)` qui est valide quand `P` contient toutes les lettres présentes dans `A` au moins autant de fois qu'elles apparaissent dans `A`, et si `L` est présente dans `F`, alors `P` ne contient pas plus de fois `L` que le nombre de fois où `L` apparaît dans `A`. En particulier, si `A` ne contient pas une lettre `L` de `F`, alors `L` n'apparaît pas du tout dans `P`. On aura donc par exemple
`check_presence_and_forbidden([a,b,a,b,c], [a, b, b], [d, b])`
mais **pas** `check_presence_and_forbidden([a,b,a], [a], [a])`
ou `check_presence_and_forbidden([a,b,d], [c], [c])`.
10. Définir un prédicat `check_one(P, O, C)` qui est valide si la proposition `P` est compatible avec la coloration `C` d'une ancienne proposition `O`. Pour cela :
    — on vérifie l'accord des lettres de `P` avec la coloration `C` de `O` ;
    — on calcule les ensembles de lettres qui doivent être présentes, et celles qui doivent être interdites ;
    — on enlève les lettres correspondant au vert ;
    — on vérifie la présence des lettres nécessaires et l'absence des lettres interdites.
11. Définir un prédicat `check_all(P, OS, CS)` qui prend en argument une proposition `P` et deux listes de même longueur `OS` et `CS` qui contiennent respectivement les anciennes propositions et leur coloration associée. Le prédicat est valide si la proposition satisfait toutes les contraintes données par les anciennes propositions et leurs colorations. On aura par exemple

```
check_all([b,o,n,d,s],
  [[o,u,a,i,s],[t,h,o,n,s]],
  [[orange,black,black,black,green],[black,black,orange,orange,green]]
```

12. Définir un prédicat `suggest(P, OS, CS)` qui est valide si `P` est un mot valide (i.e. présent dans le dictionnaire), et s'il vérifie les contraintes données par les anciennes propositions `OS` et leurs colorations `CS`.
13. Étant données une liste de propositions et une liste de leurs colorations, quelle requête faire pour obtenir une nouvelle suggestion ?

Exercice 3 : Simulation de partie

On va maintenant écrire un prédicat qui va nous permettre de voir comment notre intelligence artificielle fonctionne par rapport à une solution donnée.

14. Définir un prédicat `play_n_steps(S, N, PS, CS)` tel que pour une solution `S`, on construit une liste de `N` propositions dans `PS`, avec la coloration par rapport à `S` associée dans la liste `CS`, tel que la proposition à la position i est obtenue en utilisant `suggest` avec les contraintes données par propositions situées après.
15. Définir un prédicat `play(S, PS)` tel que `PS` est la liste de propositions obtenue au bout de 6 essais, par rapport à la solution `S`.
16. Définir un prédicat `success(S)` qui est valide si `S` est un mot valide, et si la première liste de proposition obtenue en simulant une partie pour `S` commence bien par `S`.
17. Définir un prédicat `number_of_solutions(N)` tel que `N` est le nombre de solution pour lesquels notre intelligence artificielle gagne. On pourra utiliser le prédicat `findall(L, success(L), R)` qui met dans `R` la liste des mots qui vérifient `success(L)`.

Une façon d'améliorer notre intelligence artificielle est maintenant de modifier l'ordre dans lequel les suggestions vont être faite. Pour cela, il suffit de modifier l'ordre des faits dans le fichier `word.pl`.

18. Essayer de modifier l'ordre des mots dans `word.pl` pour améliorer le nombre de succès, qu'on vérifiera avec `number_of_solutions`.

En guise d'ouverture, on peut tenter l'amélioration suivante :

19. Écrire un prédicat `suggest_best(P, OS, CS)` qui propose un mot `P` valide dans le dictionnaire, qui ne vérifie pas forcément les colorations `CS` des anciennes propositions `CS`, mais permet d'obtenir le plus de colorations différentes si on considère toutes les solutions potentielles vérifiant les colorations des anciennes propositions. Ainsi, cette proposition aura le plus grand facteur de branchement et conduira donc, au moins en moyenne, plus rapidement à la solution.