



CONCOURS CENTRALE•SUPÉLEC

Sujet 05 — MPI

# Coup optimal au Scrabble




Durée : 3h

Centrale-Supélec

## Préambule

Ce sujet d'oral d'informatique est à traiter, sauf mention contraire, en respectant l'ordre du document. Votre examinatrice ou votre examinateur peut vous proposer en cours d'épreuve de traiter une autre partie, afin d'évaluer au mieux vos compétences.

Le sujet comporte plusieurs types de questions. Les questions sont différenciées par une icône au début de leur intitulé :

- les questions marquées avec  nécessitent d'écrire un programme dans le langage demandé. Le jury sera attentif à la clarté du style de programmation, à la qualité du code produit et au fait qu'il compile et s'exécute correctement ;
- les questions marquées avec  sont des questions à préparer pour présenter la réponse à l'oral lors d'un passage de l'examinatrice ou l'examinateur. Sauf indication contraire, elles ne nécessitent pas d'appeler immédiatement l'examinatrice ou l'examinateur. Une fois la réponse préparée, vous pouvez aborder les questions suivantes ;
- les questions marquées avec  sont à rédiger sur une feuille, qui sera remise au jury en fin d'épreuve.

Votre examinatrice ou votre examinateur effectuera au cours de l'épreuve des passages fréquents pour suivre votre avancement. En cas de besoin, vous pouvez signaler que vous sollicitez explicitement son passage. Cette demande sera satisfaite en tenant également compte des contraintes d'évaluation des autres candidates et candidats.

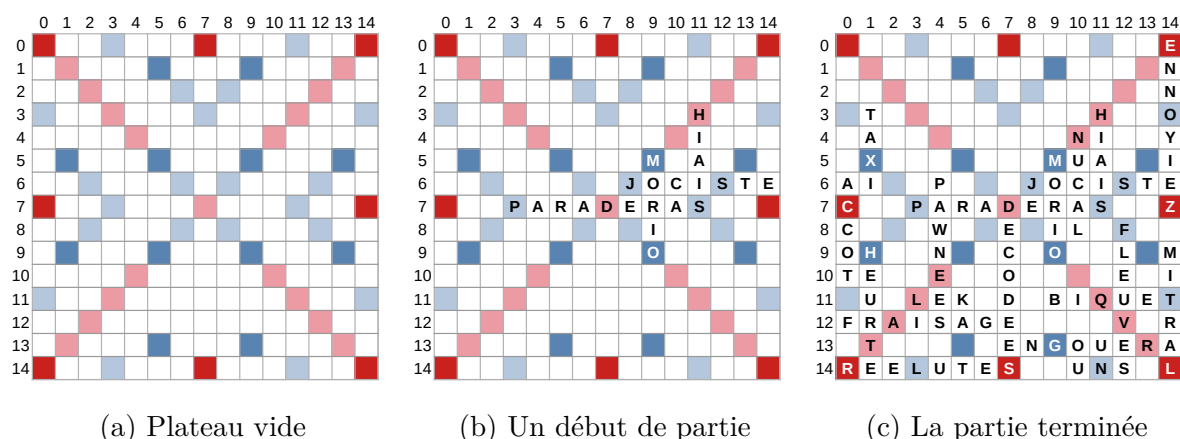


FIGURE 1 – Des plateaux de Scrabble

## 1 Introduction

Au jeu de Scrabble, on dispose d'un plateau qui est constitué d'une grille de cases. À chaque tour, un joueur tire au hasard une liste de lettres. Il doit, en utilisant toutes ces lettres ou seulement une partie, trouver un mot qui existe dans le lexique. Il doit ensuite le former sur le plateau en le plaçant à côté d'un mot déjà posé lors des coups précédents, ou bien en croisant un tel mot. Pour que le placement d'un mot soit valable, le mot lui-même et tous les autres mots adjacents, dans le sens vertical ou le sens horizontal, doivent former des mots appartenant au lexique.

Dans la figure 1c, les premiers coups ont par exemple été joués dans l'ordre suivant :

- avec son premier tirage de lettres, le joueur a placé le mot **PARADERAS** en recouvrant la case centrale,
- avec son deuxième tirage, le joueur a composé le mot **MORIO**, perpendiculairement au mot précédent en réutilisant la lettre R,
- au troisième tirage, le joueur a tiré des lettres qui lui ont permis de former le mot **HIAIS** en réutilisant le S final du mot **PARADERAS**,
- au quatrième tirage, le joueur tire des lettres qui permettent de former le mot **JOCISTE**. Le placement de ce mot est possible car, verticalement, cela crée les mots **JE** et **CA** verticaux qui sont bien dans le lexique. Si l'un d'eux n'avait pas été un mot correct, le coup n'aurait pas été permis.

Chaque mot est posé dans une direction, horizontale ou verticale.

Les lettres de l'alphabet possèdent un poids qui permet de calculer les scores, en fonction des lettres posées, des lettres adjacentes et des éventuels bonus présents sur le plateau.

Plusieurs fichiers vous sont fournis pour traiter ce sujet :

- un fichier `alphabet.ml` contient quelques fonctions utilitaires pour manipuler l'alphabet du jeu, vous n'aurez probablement pas à modifier ce fichier,
- un fichier `gaddag.ml` contient des fonctions pour manipuler la structure de GAD-DAG définie en partie 2,
- un fichier `scrabble.ml` contient des fonctions qui implémentent les règles et le déroulement du jeu,

- un fichier `lexique.txt` contient la liste des mots valides,
- un fichier `scrabble_games.sqlite` contient une base de données SQL utile pour la partie 5.

On rappelle qu'un fichier source OCaml peut être compilé en code objet avec l'appel : `ocamlc -c bar.ml`, qui produit `bar.cmo`. Ceci définit un module `Bar` qui peut être utilisé dans d'autres fichiers sources. Il est également possible d'utiliser ce module depuis `utop` en lançant par exemple `utop bar.cmo` depuis un terminal.

Vous disposez d'un script `./compile` à exécuter dans un terminal depuis le répertoire qui contient vos sources. Ce script compile les trois fichiers OCaml fournis par le sujet et les lie ensemble dans un exécutable nommé `./scrabble`.

## 2 Stockage du lexique

L'ensemble des mots que l'on peut poser sur un plateau est fixé par un lexique de mots, donné dans le fichier `lexique.txt`. Ce fichier contient un mot par ligne. Afin de stocker ce lexique d'une manière efficace pour trouver les coups autorisés sur le plateau, on implémente une structure appelée *GADDAG*, que l'on décrit dans cette partie.

▷ **Question 1.** 🚧 Rappeler pourquoi l'ensemble des mots du lexique forme un langage régulier.

On construit un premier automate déterministe (qui est un arbre préfixe, également appelé *trie*), qui est un arbre enraciné dont la racine est l'état initial, les arêtes sont étiquetées par les lettres de l'alphabet. Certains états peuvent être marqués comme étant des états terminaux (ce ne sont pas nécessairement des feuilles).

▷ **Question 2.** 📖 Si besoin en utilisant les fonctions auxiliaires déjà définies, compléter dans `gaddag.ml` la définition de la fonction `ajoute` qui modifie un automate existant en ajoutant un mot donné en paramètre.

▷ **Question 3.** 📖 On suppose que l'on a, en paramètre, une fonction `mot_suivant` qui renvoie à chaque appel le mot suivant dans le lexique. Elle lève l'exception `Lexique_fini` quand on a atteint le dernier mot du lexique. Dans le fichier `scrabble.ml`, écrire la fonction `construit_trie` qui prend en paramètre la fonction `mot_suivant` et qui renvoie le trie qui reconnaît le lexique.

▷ **Question 4.** 🧠 Pourquoi cette représentation est-elle plus intéressante qu'une simple liste de mots ? Proposer une amélioration de l'occupation mémoire (indication : on pourra par exemple considérer le lexique `{manger, bouger, ranger, debugger}`).

Étant donné un mot  $u$  dont les lettres sont  $u_1u_2\cdots u_n$ , on définit son *mot miroir*  $\bar{u}$  par  $\bar{u} = u_nu_{n-1}\cdots u_2u_1$ .

On se donne un caractère spécial, `!`, que l'on suppose distinct de toute lettre apparaissant dans un mot du lexique. Étant donné un mot  $u$ , on définit l'ensemble des *ancrés de*  $u$  comme étant l'ensemble  $\{\bar{v}!w \mid vw = u \text{ et } v \neq \epsilon\}$ .

▷ **Question 5.** 📖 En utilisant `String.init`, dont vous pouvez consulter la documentation, écrire une fonction `ancree_mot` qui prend en paramètre un entier  $i$  et un mot  $u$ , et qui calcule l'ancré  $\bar{v}!w$  de  $u$  tel que  $|v| = i$ .

Le GADDAG est le *trie* qui reconnaît l'ensemble de tous les ancres de tous les mots du lexique.

▷ **Question 6.** 📖 Écrire dans `scrabble.ml` la fonction `construit_gaddag` qui prend en paramètre le nom de fichier contenant un lexique et qui renvoie le GADDAG associé.

▷ **Question 7.** 🧠 Déterminer la complexité de votre fonction. Celle-ci sera justifiée brièvement à l'oral.

▷ **Question 8.** 🧠 Déterminer le nombre d'états de l'automate obtenu quand on construit le GADDAG associé au lexique donné dans le fichier `lexique.txt`

### 3 Optimisation en mémoire du GADDAG

Cette partie s'intéresse à la gestion fine de la mémoire occupée par le GADDAG afin d'implémenter l'optimisation vue en question 4.

La structure de *trie* permet de limiter l'occupation mémoire du GADDAG en partageant en mémoire les préfixes des mots. On veut encore améliorer la taille de la structure en partageant les suffixes des mots.

On dira que deux états du GADDAG sont équivalents si les langages reconnus en prenant ces états comme états initiaux sont les mêmes.

▷ **Question 9.** 🧠 Exprimer l'équivalence entre deux états en fonction de l'équivalence entre les états successeurs dans l'automate. En particulier, comment peut-on caractériser l'équivalence entre deux états qui n'ont pas de transition sortante ?

On rappelle qu'un GADDAG est en particulier un graphe orienté acyclique.

▷ **Question 10.** 🧠 Proposer un algorithme efficace pour fusionner les états équivalents.

▷ **Question 11.** 📖 Proposer une implémentation de cet algorithme. En OCaml, on peut utiliser l'opérateur `==` afin de comparer en temps  $O(1)$  deux objets afin de déterminer s'il s'agit *physiquement* du même objet, stocké à la même adresse en mémoire.

▷ **Question 12.** 🧠 Évaluer la complexité de votre algorithme.

▷ **Question 13.** 🧠 Évaluer la différence d'occupation mémoire entre les structures de données suivantes :

- un arbre préfixe qui contient les mots du lexique (non ancres),
- un GADDAG sans partager les suffixes,
- le GADDAG obtenu en partageant les suffixes.

On pourra utiliser la commande `time`, dont la page de manuel est disponible.

### 4 Application au jeu de Scrabble

Les plateaux manipulés dans tout le sujet seront des grilles carrées. Une case peut contenir une lettre, correspondant à un pion déjà posé dans un tour précédent, ou bien être vide et dans ce cas on indique le bonus de points accordé par cette case.

Les cases sont représentées par des valeurs de type `case`. Un plateau est une valeur de type `plateau`, qui est une matrice que l'on suppose carrée de cases. Les définitions de ces

types sont dans le fichier `scrabble.ml`, prenez le temps de les lire et de vous familiariser avec les fonctions déjà écrites dans ce fichier.

Durant une partie de Scrabble, le premier mot doit être placé de sorte qu'une de ses lettres doit être placée sur la case centrale. En cours de partie, les mots suivants doivent avoir au moins une lettre adjacente à un mot déjà posé. La case centrale et les cases adjacentes à un mot déjà posé sont appelées des *ancres*.

▷ **Question 14.** ▢ Écrire une fonction `ancres_plateau` qui prend en paramètre un plateau et qui renvoie la liste de tous les couples  $(i, j)$  tels que la case de coordonnées  $(i, j)$  est une ancre. On notera que la case centrale n'est une ancre que si elle n'a pas encore été recouverte par une lettre.

▷ **Question 15.** ♠ Combien d'ancres possède le plateau donné en figure 1c ?

Pour déterminer un coup valide, on doit d'abord :

- choisir une ancre,
- choisir un sens (vertical ou horizontal) de placement du mot.

On considère d'abord le cas d'un mot placé horizontalement. On tente de former sur l'ancre une lettre du tirage actuel. On note  $q_0$  l'état initial du GADDAG et  $q_1$  l'état obtenu après avoir lu la lettre que l'on vient de placer. Si la case à gauche de l'ancre est vide, on place une nouvelle lettre du tirage parmi les transitions du GADDAG de source  $q_1$ . Si la case à gauche de l'ancre contient déjà une lettre, on ne place aucune lettre du tirage et on suit la transition étiquetée par cette lettre depuis  $q_1$ . On itère ce processus jusqu'à ce que le GADDAG se trouve dans un état  $q_n$  tel qu'il existe une transition étiquetée par ! depuis  $q_n$ . On se replace alors sur le plateau à droite de l'ancre et à chaque étape :

- si la case est libre, on ajoute une lettre du tirage parmi celles autorisées dans l'état actuel par le GADDAG,
- si la case est occupée, on suit la transition correspondante dans le GADDAG.

Un mot est placé correctement si le GADDAG est dans un état final et que la case suivante à droite est vide ou bien se trouve sur le bord du plateau.

À chaque étape, il peut y avoir de nombreux choix possibles :

- plusieurs lettres peuvent être valablement placées sur la case en cours d'examen,
- lors du déplacement vers la gauche, on peut soit placer une lettre de plus soit lire une transition ! et repartir à droite de l'ancre.

De plus, il est possible que certains choix ne mènent pas à un mot valide (le GADDAG n'a aucune transition valide, le tirage est épuisé et le GADDAG n'est pas dans un état final).

▷ **Question 16.** ▢ Écrire une fonction `trouver_coup` qui prend en paramètres :

- un plateau,
- la liste des lettres du tirage,
- un GADDAG,
- les coordonnées d'une ancre,
- un booléen qui indique le sens de placement souhaité,

et qui renvoie, s'il existe, un mot pouvant être placé autour de cette ancre.

## 5 Statistiques des tournois

Le fichier `scrabble_games.sqlite` contient une base de données SQL qui possède une table `games` qui décrit les résultats de parties qui ont été jouées en tournois. En voici les principales colonnes :

- `gameid` est un identifiant numérique unique à chaque partie,
- `tourneyid` est l'identifiant du tournoi dans lequel la partie a eu lieu,
- `winnerid` est l'identifiant du joueur qui a gagné la partie,
- `loserid` est l'identifiant du joueur qui a perdu la partie,
- `date` est la date à laquelle s'est jouée la partie,
- `winneroldrating` est le rang de classement international, avant la partie, du gagnant de la partie,
- `winnernewrating` est son rang mis à jour après la partie,
- `loseroldrating` est le rang du perdant avant la partie,
- `losernewrating` est son rang mis à jour après la partie.

Dans un terminal, la base de données peut être manipulée grâce à la commande `sqlite3 scrabble_games.sqlite`, qui permet en suite de saisir directement des requêtes SQL.

▷ **Question 17.** 📖 ⓘ La table des joueurs ayant été égarée, indiquer comment reconstituer la liste des identifiants des joueurs à partir des parties.

▷ **Question 18.** 📖 ⓘ Écrire une requête qui renvoie la liste des tous joueurs avec, pour chacun, son nombre de victoires total.

▷ **Question 19.** 📖 ⓘ Quel est l'identifiant du joueur qui totalise le plus grand nombre de victoires ?

▷ **Question 20.** 📖 ⓘ Écrire une requête qui permet d'obtenir, pour chaque joueur, son rang de classement obtenu suite à la dernière partie qu'il a jouée.

▷ **Question 21.** 📖 ⓘ Proposer une requête qui détermine, pour un joueur fixé, le plus grand nombre de victoires obtenues sans aucune défaite entre elles. (Indication : cette requête est difficile, il peut être pertinent d'y réfléchir pendant que vous traitez la suite du sujet et d'y revenir plus tard. On pourra commencer par déterminer la durée entre ces deux victoires.)

## 6 Jeu en mode duplicaté

En mode duplicaté, tous les joueurs possèdent le même tirage et ils doivent trouver le mot qui réalise le score maximal pour un plateau donné.

▷ **Question 22.** ⓘ Formaliser l'algorithme proposé pour résoudre ce problème.

▷ **Question 23.** 📖 ⓘ Écrire une fonction `meilleur_coup` qui détermine le meilleur coup pour un plateau donné. Cette fonction prend en paramètres un plateau, un tirage et un GADDAG. Elle renvoie la position d'une ancre, le mot placé, un booléen qui donne le sens de placement et le score de ce coup.