Algorithmique et Programmation Fonctionnelle

Projet : le jeu de Jarnac

Objectifs

Rappelons tout d'abord le principe général d'un projet. Il ne s'agit pas comme dans certains examens de traiter des questions sans rapport un peu partout pour récupérer un maximum de points, mais au contraire d'avoir un produit fini qui fonctionne, quitte à ce que certaines fonctionnalités en soient absentes. À vous donc de déterminer quelles parties de ce sujet vous semblent prioritaires, et en cas de difficultés sur un point, s'il vaut mieux insister pour le résoudre ou l'abandonner complètement.

Deux fichiers de départ vous sont fournis, auxquels seront éventuellement ajoutés en cours de projet des compléments en cas de besoin [1].

Le but de ce projet est de programmer une version informatique du jeu de Jarnac, jeu de lettres créé en 1977 par Émile Lombard.

La base du projet consiste évidemment à mémoriser et modifier les lettres tirées par les joueurs et les mots placés sur la grille. Cependant, il faudra également vérifier que les mots joués sont présents dans un dictionnaire, ce dernier pouvant être codé par des structures de données très variées. On proposera donc plusieurs dictionnaires différents, ce qu'on réalisera à l'aide du système de modules. D'autre part, il faudra sauvegarder et charger une partie entre plusieurs joueurs, ce qui met en œuvre l'analyse syntaxique. Enfin, les plus courageux pourront réaliser une intelligence artificielle afin de jouer contre la machine.

Travail demandé

Le projet en entier est à rendre par la plateforme Moodle, impérativement avant le *20 décembre 2015* sous la forme d'un fichier `noms.tar.gz`. Seuls les projets respectant les contraintes suivantes seront corrigés :

1. `noms` correspond aux noms des membres du groupe
2. la commande `tar xzf nom.tar.gz` génère un répertoire `noms` qui contiendra **obligatoirement** : les sources de votre application, un fichier `Makefile`, un fichier `README` (et rien d'autre).
3. si vous avez réussi à créer un programme interactif, la commande `make` doit générer un fichier exécutable (éventuellement plusieurs si vous souhaitez montrer par exemple une version avec IA et une sans) `main.native` ou `main`.
4. sinon, votre `README` doit expliquer comment jouer dans un interpréteur interactif avec suffisamment de détails pour qu'il ne soit pas nécessaire de regarder votre code.

Par ailleurs, le fichier `README` doit contenir :
- le principe du projet en une dizaine de lignes ;
- la liste de ce qui fonctionne et ce qui ne fonctionne pas ;
- si vous vous êtes fait aider, par qui et à quel endroit ;
- comment utiliser votre programme, quels fichiers il faut charger, des exemples de tests, etc.

Dans les fichiers d'interface `.mli`, et directement en en-tête de chaque fonction, vous préciserez :
- le type de la fonction ;
- le rôle de la fonction et à quoi correspondent ses entrées et ses sorties ;
- si besoin est, un commentaire sur les choix d'implémentation que vous avez faits, et plus généralement toute information susceptible d'aider à la bonne lecture du code.

1. `http://www-verimag.imag.fr/~wack/APF/fichiers_projet`

1 Les règles du jeu de Jarnac

Jarnac se joue à deux joueurs. Un peu comme au Scrabble, chaque joueur reçoit une main de lettres piochées au hasard avec lesquelles il essaye de former des mots. La particularité du jeu est que chacun joue sur son tableau propre, mais qu'il est possible de « voler » un mot à son adversaire si celui-ci n'a pas trouvé le meilleur coup possible.

1.1 Mise en place

La pioche est formée de 144 lettres réparties de la façon suivante :

Lettre	A	B	C	D	E	F	G	H	I	J	K	L	M
Quantité	14	4	7	5	19	2	4	2	11	1	1	6	5
Lettre	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Quantité	9	8	4	1	10	7	9	8	2	1	1	1	2

Chaque joueur reçoit un tableau de 8 lignes sur 9 colonnes.

1.2 Tour de jeu

Un joueur (appelons-le Alain) tire 6 lettres au hasard et les place face visible à côté de son tableau. Il doit ensuite former sur la première ligne de son tableau un mot DE TROIS LETTRES AU MOINS figurant dans le dictionnaire.

Il tire alors en prime une lettre supplémentaire du sac, puis il dispose de 3 possibilités :

1. Former sur la ligne suivante un nouveau mot d'au moins 3 lettres
2. Transformer un mot déjà affiché sur son tableau par addition d'une ou plusieurs de ses lettres inutilisées
3. Passer s'il estime qu'il n'a aucune possibilité de jeu.

N.B. : On ne peut ni retirer une lettre d'un mot déjà formé, ni faire passer une lettre d'un mot à un autre. D'autre part, à la différence du Scrabble, il ne faut pas se soucier des alignements verticaux. Seule compte la signification d'un alignement horizontal.

Exemple Considérons qu'Alain a tiré les lettres : A B U Z R T.

Il choisit de former le mot RAT :

		9	16	25	36	49	64	81
R	A	T						

Il tire alors un T du sac, et dispose donc maintenant des lettres : B U Z T.

Il décide de former le mot BUT :

		9	16	25	36	49	64	81
R	A	T						
B	U	T						

Il tire cette fois-ci un R, qu'il décide d'ajouter à BUT pour former BRUT :

		9	16	25	36	49	64	81
R	A	T						
B	R	U	T					

Alain tire enfin un W, et décide de passer son tour. Il conserve les lettres W et Z pour son prochain tour, et c'est à l'autre joueur (Béatrice) de jouer.

1.3 Coup de Jarnac

Lors du changement de tour, si le joueur actif pouvait encore former ou transformer un mot mais ne l'a pas trouvé et si son adversaire s'en aperçoit, ce dernier a le droit de déclarer « JARNAC » et de s'emparer du mot qu'il a su découvrir afin de le mettre sur son propre tableau.

Ce coup ne remplace pas le tour normal d'un joueur, il s'insère « entre deux tours ».

Exemple À la fin du tour d'Alain, Béatrice se rend compte qu'Alain aurait pu former le mot TZAR en ajoutant son Z au mot RAT. Elle prend donc ces quatre lettres et les place sur une ligne vide de son tableau.

Alain n'a plus que BRUT sur son tableau, et c'est maintenant à Béatrice de former ou d'allonger un mot avec ses lettres.

1.4 Tours suivants

Qu'il y ait ou non un coup de Jarnac, le second joueur joue son premier tour selon les mêmes règles que le premier, jusqu'à ce qu'il ne trouve plus de solution. Il passe alors et s'expose à son tour à un éventuel coup de Jarnac...

Aux tours suivants, les joueurs ne tirent plus qu'une lettre avant de jouer. Cependant, au lieu de tirer cette lettre, ils peuvent choisir d'échanger 3 de leurs lettres encore non utilisées (ni plus ni moins) contre 3 lettres du sac ; et cela sans perdre leur tour.

1.5 Fin de la partie

La partie est terminée quand un joueur, ayant un mot sur chacune des 7 lignes précédentes, peut en afficher un sur la dernière et ceci quelle que soit la longueur de ce mot. Le gagnant est alors celui qui totalise à son tableau le plus grand nombre de points :

- 9 points pour un mot de 3 lettres
- 16 points pour un mot de 4 lettres
- etc.

Les nombres en haut des colonnes du tableau rappellent ce barême de score.

2 Implémentation des règles de base

Évidemment, votre implémentation devra respecter toutes les règles du jeu, et assurer que les joueurs ne peuvent pas tricher (volontairement ou par inadvertance). Le contenu de la pioche devra être géré correctement lui aussi.

Dans un premier temps, vous n'aurez pas à écrire un programme « interactif » : vous allez simplement choisir des types adaptés pour représenter les éléments du jeu, et écrire une série de fonctions permettant de manipuler ces structures.

Ensuite, à l'aide de ces structures, vous programmerez (toujours sous forme de fonctions) les coups de base permis par la règle du jeu :

- initialiser le jeu
- poser un nouveau mot
- ajouter une lettre à un mot
- passer
- jouer un coup de Jarnac
- détecter la fin de partie
- ...

Même si ces fonctions ne seront pas forcément utilisées directement par les joueurs dans votre programme final, cela ne vous dispense pas de bien les choisir pour rendre le jeu agréable et robuste à utiliser.

Structures de données Vous êtes pour cette partie libres d'utiliser les structures de données de votre choix. Les types présentés en cours et travaillés en TD/TP sont pour cela suffisants, mais il peut être intéressant d'utiliser les modules String ou Array de la librairie standard d'OCaml.

D'autre part, le module Random vous permettra d'introduire de l'aléatoire dans la pioche.

Pour tous ces modules, il sera de votre responsabilité de vous documenter sur la façon de les utiliser. La référence en la matière est bien entendu le manuel en ligne de la librairie standard :

`http://caml.inria.fr/pub/docs/manual-ocaml/libref/index.html`

3 Module Dictionnaire

Le jeu de Jarnac n'autorise pas de placer n'importe quelle combinaison de lettres, les seuls mots valables sont ceux qu'on trouve dans un dictionnaire de référence.

Il existe plusieurs solutions afin de représenter un dictionnaire. Avant tout, pour pouvoir tester votre jeu, vous pouvez réaliser un « faux » dictionnaire qui accepte tous les mots quels qu'ils soient.

Ensuite, une première vraie solution, facile à mettre en œuvre mais très coûteuse en temps d'exécution, consiste simplement en une liste des mots admis.

Une seconde option est d'utiliser le module Set de la librairie standard d'OCaml. Cela ne vous demande « aucun » travail dans la mesure où ce module et les fonctions associées sont fournis dans le langage, par contre il faut bien avoir compris comment utiliser un foncteur pour s'en servir !

Une troisième solution, efficace et intéressante à programmer, est de construire un arbre 26-aire, ce qui signifie que l'on considère les mots constitués des 26 lettres de l'alphabet français sans tenir compte des accents, cédiles... Chacun des 26 liens d'un nœud à l'un de ses fils correspond à une lettre de l'alphabet, et chaque nœud de l'arbre sera étiqueté par un booléen indiquant si le mot obtenu en parcourant l'arbre de la racine jusqu'à ce nœud appartient au dictionnaire ou pas.

Enfin, vous pouvez proposer une implémentation d'un dictionnaire reposant sur une autre structure de données, inventée par vos soins ou inspirée d'une des deux références ci-dessous.

Andrew W. Appel and Guy J. Jacobson. The World's Fastest Scrabble Program. Communications of the ACM, 31(5) :572–578, 1988.

Steven A. Gordon. A Faster Scrabble Move Generation Algorithm. Software, Practice and Experience, 24(2) :219–232, 1994.

Mise en œuvre Vous regrouperez les définitions de type et de fonctions liés au dictionnaire dans un module nommé `Dictionnaire`. Le type du dictionnaire est imposé (voir fichier `.mli` fourni).

De cette façon, si vous programmez votre module de jeu comme un foncteur dépendant du dictionnaire, vous pourrez comparer les différentes versions de ce dernier en instanciant votre foncteur. Dans un premier temps, vous pourrez d'ailleurs programmer un « faux » dictionnaire qui accepte n'importe quel mot, afin de pouvoir vous focaliser sur la partie Jeu.

- la fonction `dico_vide` crée un nouveau dictionnaire vide (attention à son type !).

- la fonction `member` renvoie un booléen indiquant la présence ou non d'un mot dans le dictionnaire.
- la fonction `add` ajoute un mot dans le dictionnaire.
- la fonction `remove` retire un mot du dictionnaire s'il est présent et ne fait rien sinon.
- la fonction `of_stream` crée un dictionnaire à partir d'un flux contenant un mot par ligne.
- la fonction `to_list` convertit un dictionnaire en sa liste de mots (surtout utile pour le débogage).
- Fournir un jeu de test pour votre module dictionnaire.
- Tester cela sur un vrai dictionnaire comme par exemple celui proposé : `http://www.pallier.org/ressources/dicofr/dicofr.html`

Indications : pour créer un dictionnaire arborescent vide on écrira

```
let dico_vide () = Noeud((Array.make 26 Feuille), false)
```

Il est fortement conseillé de consulter la documentation OCaml à propos des tableaux (`Array`) et des chaînes de caractères (`String`) avant de se lancer dans cette partie.

4 Jouer

Une fois que vous disposez d'un implémentation des règles et d'un dictionnaire (même basique), il est temps de les faire fonctionner ensemble.

Pour cela, vous intégrerez votre mise en œuvre des règles dans un foncteur `Jeu` qui prend en paramètre un module de type `Dictionnaire`.

Vous pouvez également profiter de cette étape pour construire un vrai programme interactif. Pour cela, vous pourrez utiliser à bon escient les traits impératifs de OCaml. Attention, il ne s'agit pas de réaliser tout le projet en impératif : ces fonctionnalités doivent être réservées à la partie interaction avec l'utilisateur.

5 Sauvegarde et lecture depuis un fichier

Dans cette partie, il est demandé de pouvoir sauvegarder l'état d'une partie dans un fichier, et de pouvoir la relire ensuite pour la reprendre. L'état d'une partie est constituée des informations suivantes :

- les noms des joueurs, leur main (lettres encore non utilisées) ;
- le tableau de chaque joueur ;
- la pioche restante ;
- le numéro du joueur dont c'est le tour.

On demande une représentation de la forme suivante :

```
(joueurs
 (Pascal (S P O I N E L))
 (Laurent (N S A V U L G I O)))
(joueur1
 (F A C I L E)
 (E X A M E N)
 (C A R T O N))
(joueur2
 (E L E V E)
 (B R I L L A N T))
(pioche
 C S N O H I N U A E L T M X)
(tour 2)
```

Dans cette partie le prochain joueur à jouer est Laurent. Le nombre de blancs (espaces, retours à la ligne) entre chaque token est arbitraire, l'exemple est ici aéré pour plus de lisibilité.

Remarque : pour utiliser les constructions `parser` sur les flots, il faut rajouter l'option `-pp "camlp4o"` à `ocamlc` pour la compilation.

N'oubliez pas de modifier le foncteur `Jeu` écrit à la section 4 pour qu'il permette (interactivement ou sous forme de fonctions) de sauvegarder et de charger une partie.

6 Pour aller plus loin

Les consignes ci-dessous concernent des parties totalement secondaires et facultatives du projet ; elles seront valorisées dans la notation, mais il vous est demandé de ne pas les aborder avant d'avoir réussi tout le reste.

6.1 Variantes

Voici trois variantes pouvant être jouées indépendemment ou simultanément.

1. Mots à 100 points. On compte simplement 100 points au lieu de 81 pour un mot de neuf lettres.
2. Jouer la dernière ligne. La partie est terminée non pas lorsque la dernière ligne est atteinte, mais lorsqu'un joueur, ayant des mots sur toutes les lignes, décide de passer et ne subit aucun Jarnac.
3. Double coup de Jarnac. Si un joueur a raté la possibilité d'agrandir deux mots distincts, son adversaire peut lui subtiliser les deux mots simultanément.

Si vous choisissez d'implanter une ou plusieurs variantes, votre objectif sera de réutiliser au maximum les fonctions déjà implantées. Vous ferez donc bon usage du système de modules et foncteurs d'OCaml, et vous illustrerez par des exemples vos variantes.

6.2 Intelligence Artificielle

Le dernier point est de définir un ou plusieurs niveaux d'intelligence artificielle, afin de joueur contre la machine. Il est clair qu'il n'est pas raisonnable de se lancer dans cette partie avant d'avoir un dictionnaire efficace, et il peut même être bénéfique de revoir le fonctionnement du dictionnaire pour faciliter la recherche de coup valide par l'IA.

Conseils stratégiques Comme on récupère une nouvelle lettre à chaque mot placé ou agrandi, on a intérêt à procéder par paliers pour poser le moins de lettres possible à la fois.

Par exemple, si on dispose des lettres ACEERT, on peut placer d'emblée le mot TRACEE et on aura le droit alors à une prime d'une lettre ; mais si on affiche successivement ARE, RACE, TRACE, TRACEE, on tirera une lettre après chaque affichage soit 4 lettres au total, ce qui donne plus de chances pour pouvoir continuer à jouer...