INF101 : Projet - Le Scrabble

Ce projet a pour but de développer un jeu de Scrabble. Les étapes sont plus ou moins guidées. Les premières sections conduisent à un programme basique permettant à des joueurs humains de s'affronter avec une interface textuelle. Les sections suivantes suggèrent des améliorations (interface graphique, aide de jeu, joueur informatisé...). Vous pouvez ajouter autant d'améliorations que vous le souhaitez pour vous rapprocher des règles complètes[a] du jeu, dont seule une version simplifiée est considérée ici.
Le projet se fait en **binôme** du même groupe de TP, ou exceptionnellement en individuel. Il ne **peut pas** se faire à plus de 2 étudiants. Vous rendrez votre code à votre enseignant-e de TP avant la dernière séance de TP de votre groupe. Une note de code sera attribuée selon votre avancement : le code doit être rendu à temps, être commenté, et s'exécuter sans erreur. Vous ferez une démonstration de votre projet et devrez répondre à des questions lors du dernier TP. Des fonctions issues du projet seront posées en quicks (contrôle continu) et une fonction adaptée du projet sera posée en examen final.

[a] `https://fisf.net/scrabble/decouverte/formules-de-jeu/`

1 Le plateau de jeu

Le Scrabble se joue sur un plateau de 15x15 cases. On représente le plateau de jeu par des listes de listes (une liste de cellules pour chaque ligne du plateau):

- Une liste *bonus* (liste de listes de chaînes de caractères) contient les bonus des cases, tels qu'indiqués sur l'image 1. La valeur "MT" (cases rouges) indique un "mot compte triple", la valeur "MD" (cases oranges, y compris la case centrale) un mot compte double (un mot passant par cette case compte pour respectivement le double ou le triple de sa valeur); la valeur "LT" (cases bleu foncé) une lettre compte triple, la valeur "LD" (cases bleu clair) une lettre compte double (la lettre posée sur cette case compte pour respectivement 2 ou 3 fois sa valeur). Les cellules sans bonus (cases vertes) contiennent une chaîne vide.
- Une autre liste *jetons* (liste de listes de caractères) qui indique les positions des jetons lettres déjà joués sur ce plateau. Chaque cellule contient le caractère du jeton posé sur cette case, ou bien une chaîne vide si la case est vide.

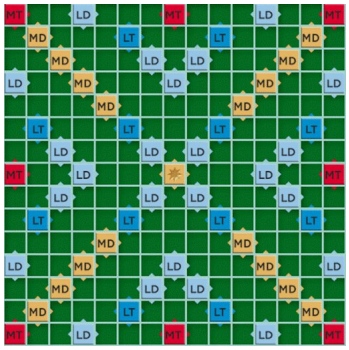

Figure 1: Plateau de Scrabble avec cases bonus

Avant de commencer

Inscrivez-vous en binôme sur Caséine pour la réalisation du projet. Cette inscription est nécessaire pour accéder au rendu du projet. Récupérez ensuite l'archive projet-scrabble-2025.zip sur Caseine. Elle contient les fichiers suivants :

- Un dictionnaire de mots : `littre.txt`
- La liste des lettres avec leur valeur et le nombre de jetons de cette lettre : `lettres.txt`
- Un squelette de code (à compléter) permettant de lire ces fichiers : `scrabble.py`

Initialisations et affichage

On va maintenant initialiser et afficher les 2 listes de listes représentant le plateau de jeu (une grille pour les bonus du plateau, et une grille pour les jetons posés dessus).

1. On fournit une fonction `init_bonus()` qui initialise et renvoie une liste de listes de caractères, contenant les bonus des cases du plateau tels qu'indiqués sur l'image. Appelez cette fonction et affichez le plateau résultant pour comprendre son format.

2. Écrire une fonction `init_jetons()` qui initialise et renvoie une liste de listes contenant uniquement des cases vides (aucun jeton sur la case = chaîne vide dans chaque cellule). Attention chaque ligne doit être une liste indépendante !

3. Écrire une fonction `affiche_jetons(j)` qui reçoit en paramètre les positions des jetons et affiche le plateau correspondant. On n'affichera pour l'instant pas les bonus, mais un espace pour les cases sans jetons. On utilisera des caractères pour séparer les lignes et les colonnes à l'affichage. *Exemple pour un plateau 5*5 avec un jeton A au centre :*

```
>>> affiche_jetons(jetons)
     01  02  03  04  05
    |---|---|---|---|---|
 01 |   |   |   |   |   |
    |---|---|---|---|---|
 02 |   |   |   |   |   |
    |---|---|---|---|---|
 03 |   |   | A |   |   |
    |---|---|---|---|---|
 04 |   |   |   |   |   |
    |---|---|---|---|---|
 05 |   |   |   |   |   |
    |---|---|---|---|---|
>>>
```

4. Améliorer cette fonction pour afficher aussi les bonus, en utilisant des symboles visibles même sur les cases contenant un jeton. *Par ex, afficher 'A' pour une case sans bonus contenant le jeton 'A', ou 'A*' pour une case avec bonus; on choisira un symbole différent pour chaque bonus possible.*

5. Écrire un programme qui génère et affiche le plateau.

6. Bonus si le reste fonctionne de manière textuelle: autre fonction pour l'affichage graphique (librairie de votre choix) avec des couleurs pour les cases bonus. Le jeu doit AUSSI fonctionner avec l'interface textuelle.

2 La pioche

Au Scrabble, la pioche est constituée par un sac de jetons pour les lettres de l'alphabet, chaque lettre étant présente en un certain nombre d'exemplaires, et valant un certain nombre de points. Les jetons sont représentés ici par des lettres majuscules de l'alphabet. La pioche contient aussi 2 jokers, valant 0 point, qu'on représentera par le caractère "?".

7. Pour l'instant, on va générer une liste aléatoire de jetons. Écrire une fonction `init_pioche_alea` qui génère une liste de 100 caractères majuscules aléatoires et 2 jokers. On s'assurera seulement que chaque lettre est présente au moins une fois.

À son tour, un joueur a le choix entre jouer un mot, ou échanger un certain nombre de lettres de sa main – au maximum 7 – avec de nouvelles lettres du sac (cela n'est autorisé que tant qu'il reste assez de jetons dans le sac). Tant que la pioche le permet, chaque joueur doit toujours avoir 7 lettres en main, donc après avoir joué un mot, il doit en repiocher autant que nécessaire pour compléter sa main.

8. Écrire une fonction `piocher(x, sac)` qui reçoit en argument un entier x, pioche au hasard x jetons dans la liste *sac*, et renvoie la liste ainsi piochée. Cette fonction **modifie** (effet de bord) le sac en supprimant les jetons piochés, et ne renvoie **rien**. Par exemple, pour $x = 7$, on pourrait renvoyer ["F","A","K","M","E","E","B"]

9. Écrire une fonction `completer_main(main,sac)` qui reçoit la main (liste de jetons d'un joueur) et le sac, qui utilise la fonction `pioche` pour compléter la main afin qu'elle compte 7 jetons. Cette fonction modifie les 2 listes reçues, et ne renvoie rien. *Attention au cas particulier où le sac ne contient plus assez de jetons, dans ce cas la fonction doit s'arrêter après avoir pioché tous les jetons restants, et la main résultante comptera moins de 7 jetons.*

10. Écrire une fonction `echanger(jetons, main, sac)` qui permet d'échanger les jetons listés de sa main avec des nouveaux jetons pris dans le sac. Il faut supprimer les jetons listés de la main du joueur, en piocher exactement autant dans le sac, remettre les jetons défaussés dans le sac, et mettre les nouveaux jetons piochés dans la main. Cette fonction **modifie** donc la main et le sac, qui gardent la même taille mais avec d'autres jetons. Attention, il faut vérifier que les jetons à échanger sont bien dans la main du joueur, et qu'il reste suffisamment de jetons dans le sac pour faire l'échange. Cette fonction renvoie un booléen indiquant si l'échange a réussi ou échoué.

11. Compléter le programme principal pour tester ces fonctions : créer des joueurs, leur donner à chacun une main initiale de 7 lettres, leur proposer d'échanger des jetons, et afficher le sac à chaque étape.

3 Construction de mots

Les joueurs doivent construire des mots avec les lettres de leur main pour les poser sur le plateau. Pour vérifier qu'un mot est autorisé, on se basera sur une liste exhaustive de tous les mots français, en lettres majuscules non accentuées. On fournit une fonction `generer_dictfr` qui lit le fichier `littre.txt` et renvoie la liste des mots français lus dans ce fichier.

12. Avant de commencer, appelez la fonction `generer_dictfr` et enregistrez la liste de mots renvoyée dans une variable *mots_fr*. Attention cette liste est très grande ! (73085 mots) Vérifiez sa longueur avec `len()`. Avec une boucle, affichez quelques mots de cette liste, par exemple tous ceux commençant par "U". Pour tester vos fonctions dans la suite, vous pouvez aussi créer une liste de quelques mots aléatoires, avant de manipuler la liste complète.

13. Écrire une fonction `select_mot_initiale(motsfr,let)` qui renvoie une liste contenant tous les mots de `motsfr` commençant par la lettre `let`. Testez votre fonction. *Par exemple, il y a 32 mots commençant par "Y".*

14. Écrire une fonction `select_mot_longueur(motsfr,lgr)` qui renvoie une liste contenant tous les mots de `motsfr` comptant exactement `lgr` lettres. Testez votre fonction. *Par exemple, il y a 39 mots de 19 caractères, un seul de 23 caractères, et aucun plus long.*

Les joueurs ont le droit d'utiliser les lettres de leur main et celles déjà posées (sans les déplacer) pour construire leurs mots. Mais pour simplifier, on va commencer par chercher des mots de la liste générée ci-dessus, jouables uniquement avec les jetons de la main du joueur.

15. Écrire une fonction `mot_jouable(mot, ll)` qui reçoit un mot et une liste de lettres, et renvoie un booléen indiquant si ce mot peut s'écrire à partir des lettres disponibles. **Attention :** cette fonction ne **doit surtout pas** modifier la liste *ll* (aucun effet de bord). *Par exemple,* `mot_possible("COURIR",["C","O","R","U","I","Z","X"])` *renvoie False (il manque un "R"), alors que* `mot_possible("PIED",["P","A","I","D","E","W","K"])` *renvoie True car toutes les lettres nécessaires sont bien dans la liste.*

16. Écrire une fonction `mots_jouables(motsfr, ll)` qui reçoit une liste de mots français autorisés, une liste de lettres disponibles en main, et qui génère et renvoie une nouvelle liste contenant la sélection des mots parmi ceux de la liste *motsfr* qui sont jouables avec les lettres de la liste *ll*. On appellera la fonction `mot_jouable` pour tester si chaque mot est jouable.
Par exemple, avec la liste de mots ["*COURIR*","*PIED*","*DEPIT*","*TAPIR*","*MARCHER*"] *et la liste de lettres* ["*P*","*I*","*D*","*E*","*T*","*A*","*R*"]*, cette fonction renvoie la liste* ["*PIED*","*DEPIT*","*TAPIR*"]*, ces 3 mots étant jouables avec ces lettres.*

17. Si ce n'est pas fait, considérer le cas des jokers dans ces fonctions, un joker pouvant remplacer n'importe quelle lettre manquante pour écrire un mot.

18. Les mots peuvent être construits en exploitant les jetons déjà posés sur le plateau. Compléter ces fonctions pour générer des listes de mots jouables avec les lettres de la main plus une (ou plusieurs) lettres manquantes (on pourra passer ce nombre de lettres supplémentaires en paramètre).

4 Valeur d'un mot

On veut maintenant calculer la valeur des mots posés. Chaque jeton a une valeur en points (entre 0 pour les jokers, et 10 pour les lettres les plus rares). On pourra se référer à `https://www.regles-de-jeux.com/regle-du-scrabble/` pour les nombres de jetons et les valeurs de chaque lettre. On fournit la fonction `generer_dico` qui lit les valeurs et nombres d'occurrences de chaque jeton dans le fichier `lettres.txt`, et renvoie le dictionnaire.

19. Avant de commencer, appelez la fonction `generer_dico` et affichez le dictionnaire résultant pour comprendre sa structure. Pour tester votre compréhension, affichez le nombre d'occurrences de la lettre "K", puis la valeur de la lettre "Z", selon ce dictionnaire.

20. Écrire une fonction `init_pioche(dico)` qui reçoit en argument un dictionnaire au format ci-dessus, et l'utilise pour initialiser la pioche, contenant exactement le bon nombre de jetons de chaque lettre, et les 2 jokers. Cette fonction renvoie la liste ainsi construite. Par ex: ["*A*","*A*","*A*","*A*","*A*","*A*","*A*","*A*","*A*","*B*","*B*",...,"?","?"]

21. Dans votre programme principal, remplacez la génération de pioche aléatoire par cette nouvelle génération de pioche.

On peut maintenant calculer la valeur d'un mot à partir de la valeur des lettres le formant. Pour simplifier, on commencera par calculer la valeur des mots indépendamment de leur placement (et donc indépendamment des éventuelles cases bonus). On trouvera la valeur de chaque jeton dans le dictionnaire reçu en argument, au format généré par la fonction `generer_dico()` ci-dessus. Si un joueur place ses 7 lettres d'un coup, c'est un Scrabble, et il reçoit un bonus de 50 points en plus de la valeur du mot.

22. Écrire une fonction `valeur_mot(mot, dico)` qui reçoit en paramètres un mot (en lettres majuscules), et le dictionnaire de jetons. Cette fonction calcule et **renvoie** la valeur de ce mot en points (la somme des valeurs de ses lettres. *Par exemple le mot "BEBE" vaut* $2+1+2+1=6$ *points.* Cette fonction doit considérer le bonus de 50 points pour un Scrabble dans le total.

23. Écrire une fonction `meilleur_mot(motsfr, ll, dico)` qui calcule et **renvoie** le meilleur mot (de plus haute valeur telle que calculée avec `valeur_mot`), parmi les mots autorisés de la liste *motsfr*, jouable avec les lettres de la liste *ll* (dont les valeurs sont données dans *dico*). Si aucun mot n'est jouable, renvoyer une chaîne vide.
Par exemple, avec la même liste de mots que plus haut, et les valeurs officielles des lettres du Scrabble, le meilleur mot jouable est "DEPIT" qui vaut 2+1+3+1+1=8 points, alors que "PIED" et "TAPIR" valent chacun 7 points.

24. Il peut y avoir plusieurs mots de **même** valeur maximale. Écrire une fonction `meilleurs_mots` qui renvoie la liste de **tous** les mots ayant la même valeur maximale. S'il n'y a qu'un seul meilleur mot, la fonction renvoie une liste d'un seul élément. S'il n'y a aucun mot jouable, elle renvoie une liste vide.

5 Premier programme principal

On va écrire un premier programme principal pour tester les fonctions codées jusqu'à présent, dans un jeu à plusieurs joueurs mais sans placement de mot dans la grille. Chaque joueur est identifié par son nom, dispose d'une liste (sa "main") de 7 jetons, et d'un score. On pourra stocker toutes ces informations dans un dictionnaire. On pourra aussi stocker un numéro d'ordre de jeu.

Les joueurs jouent à tour de rôle. Comme on n'a pas encore codé le placement de mot, pour l'instant chacun peut soit passer, soit échanger des jetons de sa main contre de nouveaux jetons, soit proposer un mot et connaître sa valeur. Si un joueur doit piocher mais qu'il ne reste plus assez de jetons dans le sac, alors la partie se termine immédiatement. Les autres joueurs perdent un nombre de points égal à la somme des valeurs des jetons qu'ils ont encore en main.

25. Écrire une fonction `tour_joueur` qui gère le tour d'un joueur : affichage du plateau, demande du coup (passer, échanger, proposer), et appel de la fonction correspondante.

26. Écrire une fonction qui détecte la fin de partie, c'est-à-dire quand un joueur essaye de piocher mais qu'il n'y a plus assez de lettres dans le sac pour compléter sa main.

27. Écrire une fonction de détection du prochain joueur, c'est-à-dire le joueur suivant dans l'ordre (cf règle du Scrabble).

28. Écrire ensuite un programme principal qui gère la boucle de jeu à x joueurs :

    - Demande le nombre de joueurs et leurs noms. Crée le sac (la pioche). Crée et affiche le plateau vide. Crée le dictionnaire des joueurs avec leurs noms, scores initialement nuls, mains de 7 jetons piochés dans le sac.
    - A chaque tour :
      - Affiche le nom du joueur courant, lui demande son coup (le joueur doit pouvoir passer, échanger un certain nombre – maximum 7 – de lettres, ou placer un mot)
      - Si le joueur passe ou échange, son tour se termine
      - Si le joueur veut proposer un mot, le programme doit le filtrer pour vérifier qu'il est correct, puis indiquer sa valeur et mettre à jour le score ; pour l'instant le mot n'est pas placé sur la grille, le programme défausse les lettres du mot et en re-pioche autant.
    - Détecte et gère la fin de partie : calcule les malus (points négatifs pour les lettres encore en main) ; affiche les scores de tous les joueurs ; affiche le nom du gagnant.

En arrivant ici, vous êtes notés sur 10. Cela signifie que votre note maximale, si vous répondez correctement aux questions lors de la soutenance, sera de 10/20. Vous devriez avoir fini cette section avant le premier quick portant sur le projet (semaine 47). Les soutenances ont lieu en semaine 50.

6 Placement de mot

Les joueurs peuvent placer leurs mots sur la grille, horizontalement ou verticalement, au contact des mots déjà posés. On s'intéresse ici à la vérification du placement des mots sur le plateau de jeu.

29. Écrire une fonction `lire_coords()` qui demande au joueur des coordonnées, les filtre jusqu'à correspondre à une case vide du plateau, et les renvoie.

30. Écrire une fonction `tester_placement(plateau,i,j,dir,mot)` qui reçoit le plateau, les coordonnées de la case de placement de la première lettre, une direction (horizontal ou vertical), et un mot, et qui vérifie si le mot est plaçable à cet endroit. Attention il peut y avoir déjà des lettres posées sur le plateau dont le mot fait usage. La fonction renvoie la liste des lettres nécessaires pour placer ce mot à cet endroit dans ce sens, ou bien une liste vide si c'est impossible (par ex le mot est trop long, ou incompatible avec les lettres déjà placées...).

31. Écrire une fonction `placer_mot(plateau,lm,mot,i,j,dir)` qui reçoit le plateau, la liste des lettres en main, un mot, les coordonnées de départ et la direction ; qui teste les contraintes (on utilisera `tester_placement`, mais cela ne suffit pas); qui place le mot à cet endroit du plateau (modifie donc le plateau), en utilisant les lettres de la main (modifie donc la main). **Attention** à ne pas *consommer* les lettres de la main tant qu'on n'est pas sûr de pouvoir placer le mot. Cette fonction renvoie un booléen pour indiquer le succès ou l'échec du placement. En cas d'échec, ni la main ni le plateau ne doivent être modifiés.

32. Modifier le calcul de la valeur d'un mot en fonction de sa position : il faut tenir compte des cases bonus (affectant une lettre ou tout le mot) sur lesquelles le mot est placé. On peut supposer ici que toutes les cases bonus recouvertes par le mot sont comptabilisées. Les règles officielles indiquent cependant que les cases bonus ne comptent que pour le premier mot posé dessus, et sont ensuite désactivées : on peut alors envisager de modifier le plateau pour "supprimer" les bonus qui ont déjà été utilisés et ne sont plus valables.

33. ***Bonus:*** on a pour l'instant supposé qu'un joueur ne pouvait placer qu'un seul mot à la fois. En réalité, en plaçant un mot le joueur peut créer un (ou plusieurs) autres mots perpendiculaires en complétant des lettres déjà placées (voir les images ci-dessous). Ajouter ces mots multiples au calcul du score d'un coup.

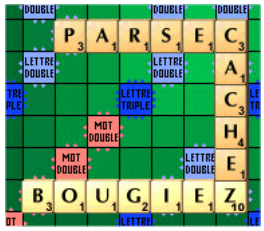

Figure 2: En plaçant "BOUGIEZ" le joueur complète aussi "CACHEZ" et marque donc les points de ces 2 mots.

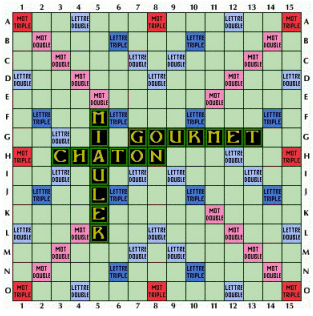

Figure 3: En plaçant "GOURMET", le joueur complète aussi "GO" et "ON" et marque donc les points de ces 3 mots.

7 Programme principal complet - boucle de jeu à x joueurs

On va maintenant compléter le programme principal de jeu à plusieurs joueurs, avec le placement des mots sur la grille. Les joueurs ont toujours 3 options à leur tour : passer, échanger la main contre de nouveaux jetons, ou proposer un mot **et le placer sur la grille**. En plaçant un mot, le joueur marque les points correspondants, et re-pioche les jetons manquants.

34. Modifier la fonction `tour_joueur` pour appeler la fonction de placement de mot
35. Modifier le programme principal qui gère la boucle de jeu. Si un joueur veut placer un mot, le programme doit :
    - le filtrer jusqu'à obtenir un mot correct **et qui peut être placé** ;
    - le placer effectivement sur la grille ;
    - re-piocher les jetons manquants ;
    - Afficher le plateau de jeu modifié avec le nouveau mot,
    - afficher la valeur du mot et mettre à jour les scores

En arrivant ici, vous êtes notés sur 15. Cela signifie que votre note maximale, si vous répondez correctement aux questions lors de la soutenance, sera de 15/20. Vous devriez avoir fini cette partie avant le second quick, en semaine 49. Les extensions proposées ci-dessous permettent d'augmenter la note maximale, mais vous devez toujours être capable d'expliquer et modifier votre code pour répondre aux questions.

8 Bonus et suggestions d'améliorations

Statistiques Permettre aux joueurs de jouer plusieurs parties, enregistrer les scores dans des fichiers, et générer des statistiques : meilleur score, plus de parties gagnées, score moyen, etc. On pourra aussi enregistrer des informations supplémentaires comme le nombre de Scrabble réalisés (par joueur, par partie).

Sauvegarde de partie Permettre d'enregistrer une partie en cours dans un fichier, et de la recharger pour la reprendre ultérieurement au même point. *Suggestion* : vous pourrez alors utiliser cette fonctionnalité pour préparer une partie proche de la fin pour faire une démonstration lors de la soutenance du projet.

Affichage graphique Permettre de jouer en mode graphique, en cliquant sur le plateau et la main. Les cases bonus seront alors affichées en couleur.

Aide de jeu

1. proposer une aide de jeu en suggérant un placement pour un mot
2. proposer une aide de jeu en suggérant des mots jouables avec les lettres en main

Intelligence Artificielle

1. Améliorer le calcul du meilleur mot pour faire usage des lettres déjà posées sur le plateau
2. Créer un joueur automatisé à intégrer dans la boucle de jeu

Scrabble sur le net et sur smartphone Inspirez vous du tutoriel BeeWare `https://tutorial.beeware.org/fr/latest/` pour faire une version de votre jeu de Scrabble pour navigateur / pour smartphone.

Avertissement : ce bonus est réservé à ceux qui ont déjà réalisé un autre bonus. En effet, si utiliser les outils BeeWare devrait faire augmenter votre expertise Python en systèmes, vous ne gagnerez pas en compétences sur le programme de l'UE INF101/131, et donc cela ne vous sera d'aucune aide pour les examens. **ATTENTION** : si vous décidez de vous lancer quand même dans l'aventure, gérez votre temps : vous avez aussi d'autres UE à réviser.