Projet de Programmation
Master Informatique
2025–2026

université de BORDEAUX

Scrabble

Cahier des charges

29 décembre 2025
Version 1.0

Ce projet a pour objectif de développer un logiciel permettant de jouer au Scrabble de 2 à 4 joueurs. Le programme doit permettre aux joueurs de s'affronter sur un plateau de jeu en respectant les règles.

Mots-clefs: Jeu de plateau, deux à quatre joueurs, stochastique, information partielle.

1 Généralités

1.1 Langage de programmation

Le langage de programmation doit être choisi parmi :

- **C** : C11 (ISO/IEC 9899 :2011).
- **Java** : Java SE 21+.
- **Python** : Python 3.10+.

1.2 Style de codage

Le *coding style* du programme doit respecter :

- **C** : le *coding style* de GNU.
- **Java** : le *coding style* de Google.
- **Python** : le *coding style* du PEP8.

1.3 Documentation

Le programme et le code source doit être documenté : manuel utilisateur, option d'aide en ligne de commande et commentaires dans le code source. En outre, on utilisera :

- **C** : Doxygen.
- **Java** : Javadoc.
- **Python** : Sphinx.

1.4 Langue par défaut dans le code

La documentation, le nom des variables, des fonctions et des fichiers doivent être en anglais.

1.5 Système cible

Le programme doit fonctionner sur des systèmes GNU/Linux.

1.6 Tests

Le programme doit être testé et validé par suffisamment de tests pour garantir son bon fonctionnement (couverture globale d'au moins 85%). Les tests doivent être automatisés et facile à lancer via le build-system.

1.7 Bugs

Aucun bug menant à un crash ne doit être présent dans le programme. Les spécifications doivent être respectées et les erreurs doivent générer des messages d'erreur explicites.

1.8 Performances

Les performances du programme doivent être mesurées et documentées pour différentes entrées du programme ainsi que pour ses cas limites.

1.9 Bibliothèques

Seule la bibliothèque standard du langage choisi et celles qui sont listées ici sont acceptées.

1.10 Build-system

Le '*build-system*' utilise les outils suivants :

- **C** : `CMake`.
- **Java** : `Maven`.
- **Python** : `pyproject.toml` + `setuptools`.

1.11 Frameworks de tests

Le *framework* de test est basé sur :

- **C** : `CTest` + `googletest` + `gcovr`.
- **Java** : `JUnit` + `Jacoco`.
- **Python** : `pytest` + `coverage`.

1.12 Gestion des options

Les options en ligne de commande sont gérées par :

- **C** : le module `getopt_long`.
- **Java** : la bibliothèque `commons-cli`.
- **Python** : le module `argparse`.

1.13 Bibliothèque graphique

La bibliothèque graphique utilisée est :

- **C** : LibGtk 4.x.
- **Java** : JavaFx.
- **Python** : PyGObject.

1.14 Internationalisation

Le *framework* d'internationalisation est basé sur :

- **C** : la bibliothèque `gettext`.
- **Java** : `ResourcesBundle` et `Locale`.
- **Python** : le module Python `gettext`.

1.15 Apprentissage automatique

L'apprentissage automatique doit être basé sur:

- **C**: la bibliothèque `tensorflow` (C API).
- **Java**: la bibliothèque `tensorflow` (Java API).
- **Python**: la bibliothèque `keras`.

2 Besoins généraux

F1. Lancement du programme

Le programme doit se lancer depuis la ligne de commande via un exécutable ou un script du nom du projet.

```
#> scrabble [OPTIONS] [ARGUMENTS]
```

Il doit accepter à minima les options suivantes :

- '`-h`' ou '`--help`' : affiche l'aide du programme.
- '`-V`' ou '`--version`' : affiche la version du programme.
- '`-v`' ou '`--verbose`' : accroît la verbosité du programme.
- '`-d`' ou '`--debug`' : affiche les messages de debug.

Si une option est incorrecte, le programme doit afficher un message d'erreur et l'aide du programme.

Les messages destinés à l'utilisateur seront écrits sur la sortie standard (`stdout`) et les messages d'erreurs devront être écrits sur la sortie d'erreur standard (`stderr`). De plus, le code de retour du programme devra être 0 en cas de succès et 1 (ou un autre code d'erreur) en cas d'erreur.

F2. Fichier de configuration

Les valeurs par défaut des options du programme doivent pouvoir être redéfinies via un fichier de configuration au format `INI`. Ce fichier de configuration doit être nommé '`.scrabblerc`' et être placé à la racine du répertoire de l'utilisateur (`HOME`) qui lance le programme. Le fichier de configuration doit être lu au démarrage du programme s'il est présent et les options qu'il contient doivent être utilisées par défaut. Les options passées en ligne de commande doivent supplanter les options du fichier de configuration.

Si le fichier de configuration n'est pas présent, le programme doit créer un fichier de configuration minimal et se lancer normalement en respectant les options par défaut. Si le fichier de configuration est présent mais n'est pas valide, le programme doit afficher un message d'avertissement sur la console et se lancer normalement sans écraser le fichier de configuration. Ci-dessous un exemple de fichier de configuration valide (le nom des options sont là à titre d'exemple) :

```
[defaults]
verbose = true
blitz = false
timeout = 30
```

F3. Internationalisation (`i18n`)

La langue par défaut du logiciel sera l'anglais. Le français, au moins, doit aussi être supporté. Si les variables d'environnement '`LANG`' ou '`LC_ALL`' sont définies, le programme doit les utiliser pour déterminer la langue à utiliser. Si la langue n'est pas supportée, le programme doit afficher un message d'avertissement et se lancer en anglais (la langue par défaut).

3 Options au lancement

F4. Interface en ligne de commande

Le programme démarre une interface en ligne de commande sur une nouvelle partie avec les options par défaut si aucune option spécifique n'est donnée. Si des options sont données, celles-ci seront appliquées à la partie qui est lancée. Enfin, si on met en argument un fichier de sauvegarde, le programme doit lancer une partie à partir de ce fichier.

F5. Mode `blitz`

L'option '`-b`', '`--blitz`' lance une partie en mode blitz. Dans ce mode, un temps maximum est donné à chaque joueur pour la partie. Dès que le temps d'un des joueurs est écoulé, il perd la partie et doit en être notifié.

De plus, '`-t TIME`', '`--time TIME`' spécifie la limite de temps en minutes pour chaque joueur en mode blitz. Par défaut, le temps de réflexion est de 30 minutes. Si cette option est donnée sans l'option '`-b`', '`--blitz`', elle sera ignorée mais un message d'avertissement sera affiché.

F6. Mode `contest`

L'option '`-c`', '`--contest`' lance le programme en mode '`contest`'. Dans ce mode, le programme doit lire un fichier donné en argument qui contient une position du jeu et répondre sur la sortie standard quel coup jouer.

F7. Interface graphique

Le programme doit posséder une interface graphique qui permet les mêmes interactions qu'avec le CLI via une fenêtre graphique. Cette interface graphique sera lancée si l'option '`-g`', '`--gui`' est présente au lancement.

F8. Mode IA

L'option '`-a [COLOR]`', '`--ai [COLOR]`' permet de remplacer le joueur de la couleur donnée par un joueur artificiel. Les couleurs possibles sont celles présentes dans le jeu plus '`A`' (pour *All*) et une couleur par défaut si aucune couleur n'est spécifiée après l'option.

F9. Options spécifiques

L'option '`-p N`', '`--players N`' permettra de spécifier le nombre de joueurs pour une partie (de 2 à 4). Par défaut, le jeu se jouera à deux joueurs.

L'option '`-l LANG`', '`--language LANG`' permettra de spécifier la langue du jeu. Par défaut, le jeu sera en anglais (`en`). Les langues supportées seront au moins : anglais (`en`), français (`fr`). Il devra être possible de lister les langues supportées avec l'option '`--list-languages`'.

L'option '`-s`', '`--super`' permettra de jouer en mode Super-scrabble. Dans ce mode, le plateau de jeu sera plus grand (21 par 21) et le nombre de lettres est augmenté.

L'option '`-D FILENAME`', '`--dictionary FILENAME`' permettra de spécifier un fichier de dictionnaire personnalisé. Par défaut, le programme utilisera le dictionnaire fourni avec le programme.

4 Gestion du jeu

Besoins communs aux deux interfaces (CLI et GUI).

F10. Gestion des règles

Le programme doit respecter les règles du jeu. Il doit être capable de gérer les coups joués par les joueurs, de vérifier leur validité et de mettre à jour l'état du jeu en conséquence. Il doit aussi pouvoir détecter la fin d'une partie, calculer les scores des joueurs et déterminer s'il y a un gagnant ou un match nul.

F11. Gestion des erreurs de l'utilisateur

Le programme doit être capable de gérer les erreurs de manière simple et compréhensive pour l'utilisateur. Si un utilisateur se trompe, le programme doit afficher un avertissement explicite et lui demander de recommencer.

F12. Gestion du *undo*/*redo*

Le programme doit afficher l'historique des coups joués et permettre le '*undo*' et le '*redo*'. Seuls les joueurs humains peuvent utiliser ces fonctionnalités. Lorsqu'un joueur humain fait un '*undo*', il annule son dernier coup joué et tous les coups joués par les joueurs artificiels précédents jusqu'à ce que l'on arrive à un autre joueur humain. Le '*redo*' permet de rejouer les coups annulés en suivant le même principe. Si le joueur humain joue un coup différent de celui qui a été annulé, l'historique futur est effacé et la partie continue normalement à partir de là.

F13. Gestion des conseils

Le programme doit être capable de donner des conseils à l'utilisateur sur le coup à jouer. Ces conseils doivent être basés sur l'état actuel du jeu et doivent être affichés à l'utilisateur sous forme compréhensible et simple.

F14. Gestion du blitz

En mode blitz, l'interface doit afficher le temps restant pour chaque joueur et doit arrêter la partie et déclarer le joueur perdant à l'instant où le temps du joueur est écoulé. Un joueur doit toutefois pouvoir mettre le jeu en pause.

5 Command Line Interface (CLI)

F15. Shell interactif

L'interface en ligne de commande doit se présenter sous la forme d'un shell interactif qui affiche un prompt à l'utilisateur et qui soit capable de lire les coups des joueurs ainsi que des commandes et d'y répondre. En cas d'erreur, le shell doit afficher un message d'erreur explicite.

```
>> COMMAND [OPTIONS] [ARGUMENTS]
```

Les commandes reconnues doivent être les suivantes :

- `new [ARGS]` : Lance une nouvelle partie, si des arguments sont ajoutés il est possible de jouer contre une IA, de choisir sa couleur, etc.
- `help [CMD]` : Affiche l'aide du shell ou de '`CMD`'.
- `quit` : Quitte le programme.
- `load FILE` : Charge une partie depuis un fichier.
- `save FILE` : Sauvegarde la partie en cours.
- `pause` : Arrête le défilement du temps en blitz.
- `hint` : Affiche un conseil pour le coup à jouer.
- `undo [N]` : Annule le dernier coup joué (ou les `N`).
- `redo [N]` : Rejoue le dernier coup annulé (ou les `N`).
- `show board` : Affiche l'état courant du jeu.
- `show history` : Affiche l'historique des coups joués.
- `show time` : Affiche le temps restant de chaque joueur.
- `show configuration` : Affiche la configuration.
- `set PARAM=VALUE` : Change la configuration courante, par exemple : '`set debug=true`'.

F16. Proposer de sauvegarder avant de quitter

Si on quitte le programme et qu'aucune sauvegarde n'a été faite depuis le dernier coup joué, le programme doit demander si l'on veut sauvegarder la partie avant de quitter :

```
>> quit
Save the game before quitting? [y/N]
```

Si aucun choix reconnaissable n'est donné, le programme supposera que le choix par défaut a été choisi (`N`/*No*) et quitte le programme. Si on tape 'y' ou 'Y', le programme doit demander le chemin du fichier dans lequel sauvegarder et faire la sauvegarde. En cas d'erreur d'enregistrement, le programme doit redemander si l'utilisateur veut effectuer la sauvegarde.

F17. Édition de la commande en cours

Le shell doit permettre d'éditer la ligne de commande non encore validée en déplaçant le curseur avec les flèches pour permettre l'insertion ou la suppression de caractères. L'utilisation de bibliothèques comme GNU Readline (C), JLine (Java) ou readline (Python) est recommandée.

F18. Historique des commandes tapées

Le programme doit garder l'historique des commandes tapées par l'utilisateur. Il doit être possible de naviguer dans cet historique avec les flèches haut et bas du clavier. En tapant <Ctrl>+<R>, on doit pouvoir faire une recherche dans l'historique des commandes et obtenir la dernière commande qui correspond à la recherche.

F19. Complétion des commandes

Le programme doit être capable de compléter les commandes tapées par l'utilisateur. Par exemple, si l'utilisateur tape 'h' puis appuie sur la touche <Tab>, le programme doit afficher la commande qui correspond à 'h' si elle est unique, sinon un signal d'erreur doit être produit (clignotement du curseur ou son d'erreur) et un second appui sur la touche <Tab> affichera *toutes* les commandes commençant par 'h'.

F20. Notation des coups

À chaque coup pour un humain, le programme devra afficher le plateau ainsi que le temps pris par chaque joueur si la partie est en blitz.

En mode ASCII les cases vides seront représentées par un '_' et les lettres par leur lettre majuscule. Les cases spéciales seront mises en exergue par des couleurs différentes en utilisant les codes ANSI pour changer la couleur de background. Les cases 'lettre compte double' seront cyans, les cases 'lettre compte triple' seront bleues, les cases 'mot compte double' seront rouge et les cases 'mot compte triple' seront violettes.

Dans le cas du Super-scrabble, les cases 'lettre compte double' seront cyans léger (*light cyan*), 'lettre compte triple' seront cyans, les cases 'lettre compte quadruple' seront violettes, les cases 'mot compte double' seront rouges léger (*light red*), les cases 'mot compte triple' seront rouges et les cases 'mot compte quadruple' seront violettes.

```
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
a _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
b _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
c _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
d _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
e _  _  _  _  _  P  _  _  _  _  _  _  _  _  _
f _  _  _  _  _  H  _  _  _  _  _  _  _  _  _
g _  _  _  _  C  O  U  N  T  _  _  _  _  _  _
h _  _  _  _  _  T  _  _  R  _  _  _  _  _  _
i _  _  _  _  _  O  _  _  E  N  G  L  I  S  H
j _  _  _  _  _  _  _  _  E  _  _  _  _  _  _
k _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
l _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
m _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
n _  _  _  _  _  _  _  _  _  _  _  _  _  _  _
```

```
  a b c d e f g h i j k l m n o
o _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

Le joueur doit entrer les coordonnées de la case de départ, la direction du mot (horizontale ou verticale) et le mot à placer. Par exemple, pour placer le mot 'COUNT' en partant de la case g5 en direction verticale, le joueur entrera : '`g5v COUNT`'. Le programme doit vérifier que le mot peut effectivement être composé à partir de la réserve de lettres du joueur, que les cases sont bien vides là où il faut et qu'il est bien présent dans le dictionnaire de la partie. Si la proposition du joueur est incorrecte, un message d'erreur qui indiquera la cause du problème devra être affiché et le programme devra demander une autre action.

6 Format de sauvegarde

F21. Sauvegarde et restauration d'une partie

Les fichiers de sauvegarde doivent être au format texte ASCII. Le programme doit pouvoir convertir une partie en cours en un fichier à ce format et restaurer une partie à partir d'un fichier de sauvegarde. En cas d'erreur lors du chargement, afficher un message d'erreur explicite à l'utilisateur. Le parseur doit être robuste aux erreurs de format (par exemple, des lignes manquantes ou des valeurs incorrectes) et doit signaler les erreurs rencontrées.

La sauvegarde doit comporter les parties suivantes (dans l'ordre) : les paramètres du jeu (`settings`), l'état du jeu (`game`), et l'historique des coups joués (`history`).

F22. Commentaires dans la sauvegarde

La syntaxe de la sauvegarde doit gérer deux types de commentaires, les commentaires en ligne qui débutent par le caractère dièse ('`#`') jusqu'à la fin de la ligne, et les commentaires en bloc délimités par les caractères accolades ('`{ }`').

F23. Format des paramètres du jeu

Les paramètres du programme sont, par exemple, le temps restant pour chaque joueur (en mode blitz), le nombre de joueurs de la partie, . . . On les représente sous la forme d'une liste de couples clé-valeur comme suit :

```
[settings] # Paramètres du programme
debug=true
turns-limit=100
ai-mode=minimax
```

F24. Format du plateau de jeu

Le programme devra être capable de sauvegarder et restaurer des fichiers dans un format ASCII simplifié qui représente un plateau de taille quelconque. Par exemple, un fichier de sauvegarde pourrait ressembler à ceci :

```
[game]
1 # joueur courant
# le plateau de jeu
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ P _ _ _ _ _ _ _ _ _
_ _ _ _ _ H _ _ _ _ _ _ _ _ _
_ _ _ _ C O U N T _ _ _ _ _ _
_ _ _ _ _ T _ _ R _ _ _ _ _ _
_ _ _ _ _ O _ _ E N G L I S H
_ _ _ _ _ _ _ _ E _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _

# la réserve de lettres
rack-1: A B C D E F G
rack-2: H I J K L M N

# les scores des joueurs
score-1: 15
score-2: 10

language = en
# dictionary = path/to/dictionary.txt
# 'super' mettre avec le mode Super-scrabble
```

Une erreur de lecture dans un fichier doit être signalée par une erreur à l'utilisateur en spécifiant le type d'erreur et la ligne où elle a été trouvée et le programme doit s'arrêter.

F25. Format de l'historique

Le programme devra être capable de lire et d'écrire des historiques de coups joués. Ils se présenteront comme une liste de mots placés par les joueurs, par exemple :

```
[history]
1 g5h COUNT 2 e6v PHoTO
1 g9v tREE 2 i9h eNGLISH
1 pass 2 j10v hI
...
```

F26. Dictionnaire

Le programme doit comporter un module `lexicon` qui permet de charger un dictionnaire de mots. Ce dictionnaire est utilisé pour vérifier si un mot est valide. Le format d'entrée du dictionnaire sera un fichier texte contenant un mot par ligne. Le programme doit charger ce dictionnaire au démarrage en fonction de la langue choisie par l'utilisateur. Par défaut, le programme charge le dictionnaire qui correspond à la locale de l'utilisateur s'il possède un dictionnaire pour cette langue, sinon il chargera le dictionnaire anglais par défaut.

F27. Module `dictionary`

Le module `dictionary` est une interface abstraite qui utilisera une des deux structures de données (parmi `dawg` ou `gaddag`) pour gérer le dictionnaire de mots. Il doit permettre de vérifier si un mot est valide, de trouver tous les mots possibles à partir d'un ensemble de lettres et de trouver le meilleur mot possible à partir d'un ensemble de lettres.

F28. Module `dawg`

Le module `dawg` doit proposer une implémentation de l'interface abstraite `dictionary` avec des structures de données de type *Dawg* (Directed Acyclic Word Graph).

F29. Module `gaddag`

Le module `gaddag` doit proposer une implémentation de l'interface abstraite `dictionary` avec des structures de données de type *Gaddag* [1, 2].

7 Graphical User Interface (GUI)

F30. Menus de l'application

L'interface graphique doit comporter le menu '`File`' avec :

- `New Game` : Lancer une nouvelle partie.
- `Load Game` : Charger une partie depuis un fichier.
- `Save Game` : Sauvegarder la partie en cours.
- `Configuration` : Édition de la configuration.
- `Info` : Informations du logiciel (version, auteurs, ...).
- `Quit` : Quitter le programme.

Ainsi que le menu '`Game`' avec les items suivants :

- `Undo` : Annuler le dernier coup joué.
- `Redo` : Rejouer le dernier coup annulé.
- `Pause` : Arrêter le défilement du temps en blitz.
- `Hint` : Afficher un conseil pour le coup à jouer.

F31. Raccourcis clavier

Le GUI doit proposer les raccourcis clavier suivants :

- <Ctrl>+<N> : Nouvelle partie.
- <Ctrl>+<L> : Charger une partie sauvegardée.
- <Ctrl>+<S> : Sauvegarder la partie en cours.
- <Ctrl>+<,> : Afficher la configuration du logiciel.
- <Ctrl>+<I> : Afficher les informations du logiciel.
- <Ctrl>+<Q> : Quitter le programme.
- <Ctrl>+<U> : Annule le dernier coup joué.
- <Ctrl>+<R> : Rejoue le dernier coup annulé.
- <Ctrl>+<P> : Mettre le jeu en pause.
- <Ctrl>+<H> : Afficher un conseil de coup à jouer.

L'utilisateur doit pouvoir proposer ses propres raccourcis clavier pour chaque action listée. Les raccourcis clavier personnalisés seront sauvegardés dans le fichier de configuration du logiciel afin d'être restaurés à chaque lancement de l'application.

F32. *Drag'n'Drop*

Le *glisser-déposer* doit permettre de jouer à la souris en déplaçant les pièces sur le plateau.

8 Joueur artificiel

F33. Temps de réflexion borné

L'option '`--ai-time TIME`' permet de spécifier le temps de réflexion de l'algorithme de recherche. Si le temps est écoulé, le programme doit jouer le meilleur coup trouvé jusqu'à présent. Par défaut, le temps de réflexion sera de 5 secondes.

F34. Recherche de mots possibles

Le programme doit être capable de trouver tous les mots possibles à partir des lettres d'un joueur et d'un plateau de jeu en se basant sur le module `gaddag`.

F35. Recherche du meilleur coup

Le programme doit pouvoir trouver le meilleur coup parmi les possibilités en maximisant le gain en points pour le joueur actuel, c'est à dire en utilisant les cases spéciales du plateau.

F36. Algorithme Exptiminimax

La fonction de recherche du meilleur coup donne une optimisation locale du score. Si l'on veut trouver le meilleur coup sur l'ensemble du jeu, il faut anticiper les coups des adversaires et le tirage probable de lettres. Pour cela, on utilise l'algorithme Exptiminimax pour évaluer le meilleur coup avec une profondeur de recherche limitée. L'option '`--ai-exptiminimax`' permet d'activer cette fonctionnalité.

F37. Recherche de mots par *Machine Learning*

Le programme doit proposer une fonction de recherche de mots à partir d'un ensemble de lettres qui sera obtenue par apprentissage à partir d'un dictionnaire fixé. L'option '`--ai-ml`' permet d'utiliser cette fonction de recherche.

9 Jeu en réseau

F38. Découverte de serveurs et connexion client-serveur basique

Le programme doit implémenter les commandes réseau de base suivantes (en mode CLI mais aussi dans le GUI) :

- `server list` : Affiche la liste des serveurs de jeu disponibles sur le réseau local. Les serveurs diffusent périodiquement (toutes les 10 secondes) leur présence via un message UDP en broadcast sur le port 12346 contenant le nom du serveur et le port TCP. Un serveur est retiré de la liste si aucun message n'est reçu pendant 30 secondes.
- `server start [PORT]` : Démarre un serveur de jeu simple sur le port TCP indiqué (12345 par défaut). Le serveur doit pouvoir accepter une connexion client à la fois. Une fois démarré, le serveur diffuse sa présence sur le réseau via UDP broadcast. Si le port est déjà utilisé, afficher un message d'erreur.
- `server stop` : Arrête le serveur de jeu. Le client connecté est notifié de l'arrêt et déconnecté.
- `join [IP[:PORT]]` : Se connecte au serveur à l'adresse IP et au port indiqués (localhost :12345 par défaut). Une fois connecté, le programme passe en mode client. Si la connexion échoue, afficher un message d'erreur explicite.
- `quit` : Quitte le serveur et revient en mode local.
- `ping` : Envoie un message ping au serveur. Le serveur répond avec un pong et le temps de réponse est affiché. Cette commande permet de tester la connexion.

Le serveur doit gérer les déconnexions inopinées avec un timeout de 1 minute.

Les messages échangés entre client et serveur doivent être de simples mots en ASCII terminés par un retour chariot ('`\n`'). Par exemple :

```
Client: PING
Server: PONG TIME=42ms
Client: QUIT
Server: BYE
```

F39. Serveur multi-clients et gestion de parties

Le serveur doit être étendu pour gérer plusieurs clients simultanément en utilisant un thread par client ou de la programmation asynchrone. L'accès aux données

partagées doit être protégé par des mécanismes de synchronisation (verrous/mutex) pour éviter les problèmes d'accès concurrent aux données.

Les commandes suivantes doivent être ajoutées :

- `server status` : Affiche le statut du serveur : port d'écoute, nombre de clients connectés, nombre de parties en cours.
- `players` : Affiche les joueurs connectés au serveur avec leur identifiant, nom, et statut (`idle`, `ingame`).
- `scoreboard` : Affiche le tableau des scores des joueurs qui ont joué sur le serveur (nombre de victoires, défaites, parties jouées).
- `new PLAYER_ID` : Démarre une nouvelle partie avec le joueur spécifié s'il est `idle`. Si le joueur n'existe pas ou n'est pas disponible, afficher une erreur. Si le jeu accepte plus de deux joueurs, il suffit de mettre plusieurs joueurs en argument à la commande.

Le serveur doit maintenir l'état de toutes les parties en cours et router les commandes de jeu (coups) vers la bonne partie. Chaque coup joué par un client est validé par le serveur puis transmis à l'autre joueur.

Le programme doit aussi proposer les options de lancement suivantes :

- `'-s [PORT]'` ou `'--server [PORT]'` : Lance directement en mode serveur sur le port indiqué.
- `'-d'` ou `'--daemon'` : Lance le serveur en mode headless (sans interface).

Exemple de protocole pour les coups :

```
Client1: MOVE e2-e4
Server -> Client2: OPPONENT_MOVE e2-e4
Client2: MOVE e7-e5
Server -> Client1: OPPONENT_MOVE e7-e5
```

Le serveur doit valider que les coups sont légaux pour empêcher la triche (coups invalides, actions hors tour).

F40. Système d'invitations et gestion avancée des joueurs

Le système de création de parties doit être amélioré avec un mécanisme d'invitation et d'acceptation :

- `players [PLAYER_ID]` : Si un ID est fourni, affiche les informations détaillées du joueur (statut, score, statistiques). Les statuts possibles sont maintenant : `idle`, `away`, `waitgame` (en attente de réponse à une invitation), `ingame`.
- `new PLAYER_ID` : Envoie une invitation au joueur spécifié. Le joueur invité reçoit une notification et peut accepter (`accept`) ou refuser (`decline`) l'invitation. Si aucune réponse n'est reçue après 5 minutes, l'invitation expire. Le demandeur peut annuler l'invitation avec `cancel`.
- `accept` : Accepte l'invitation en cours. La partie démarre immédiatement.
- `decline` : Refuse l'invitation. L'inviteur est notifié.
- `cancel` : Annule l'invitation envoyée (si en attente).
- `away` : Change le statut du joueur en `away`. Les invitations ne peuvent être envoyées aux joueurs `away`.
- `back` : Revient au statut `idle`.

Le serveur doit gérer les timeout d'invitation (5 minutes), notifier les joueurs des changements d'état, et empêcher les actions invalides (inviter un joueur déjà en partie, accepter sans invitation, etc.). Exemple d'interaction :

```
Client1: NEW 2
Server -> Client1: INVITATION_SENT
                   PLAYER=Bob TIMEOUT=300s
Server -> Client2: INVITATION_RECEIVED
                   FROM=Alice EXPIRES=300s
Client2: ACCEPT
Server -> Client1: INVITATION_ACCEPTED
                   STARTING_GAME
Server -> Client2: GAME_START
                   OPPONENT=Alice
```

Références

[1] Andrew W. Appel and Guy J. Jacobson. The world's fastest scrabble program. *Communications of the ACM*, 31(5) :572–578, 1988.

[2] Steven A. Gordon. A faster scrabble move generation algorithm. *Journal of Software : Practice and Experience, Wiley*, 24(2) :219–232, 1994.