

Wordle

Groupe3

Anthony Genti
Youssef Boudount
Louis Rives-Lehtinen
Samy Seghir

Licence d'Informatique
Ingénierie logicielle
UE Projet Programmation

UFR
SCIENCES
TECHNOLOGIES
SANTÉ



19/05/2024

Responsables
Noe Cecillon
Jarod Duret

CENTRE
D'ENSEIGNEMENT
ET DE RECHERCHE
EN INFORMATIQUE
ceri.univ-avignon.fr

Sommaire

Titre	1
Sommaire	2
1 Introduction	5
1.1 Wordle	5
1.1.1 Qu'est-ce que Wordle ?	5
1.1.2 Règles du Jeu	5
1.1.3 Déroulement d'une Partie	5
1.2 Présentation du projet	5
1.3 Contexte et objectifs	6
1.4 Méthodologie et répartition des tâches (Intermédiaire)	6
1.5 Méthodologie et répartition des tâches (Finale)	7
1.6 Organisation interne et flux de travail	7
1.6.1 Clonage du projet	7
1.6.2 Structure des branches	8
1.6.3 Travailler dans sa propre branche	8
1.6.4 Ajout de modifications	8
1.6.5 Commit des modifications	8
1.6.6 Pousser les modifications sur GitLab	8
1.6.7 Gestion des identifiants GitLab	8
2 Conception	9
2.1 Architecture générale du logiciel	9
2.2 Diagrammes de conception (UML)	9
2.2.1 Diagramme de classes	9
2.2.2 Diagramme de séquence	10
2.2.3 Diagramme d'activité pour l'hôte	11
2.2.4 Diagramme d'activité pour le participant	13
2.3 Interface utilisateur et expérience utilisateur	15
3 Développement Intermediaire	16
3.1 Déroulement de la partie (Youssef)	16
3.1.1 Classe Server	16
3.1.2 Classe Lobby	16
3.1.3 Classe ClientHandler	17
3.1.4 Classe Client	18
3.2 Création d'un salon d'accueil (Youssef)	20
3.2.1 Classe CreateGameUI	20
3.2.2 Classe JoinLobbyUI	21
3.2.3 Classe GameLobbyUI	22
3.3 Déroulement de la partie (Anthony)	24
3.3.1 Classe Player	24
3.3.2 Classe HumanPlayer	25
3.3.3 Classe GameSession	26
3.3.4 Classe GameInfo	27
3.3.5 Classe GameConfig	28
3.3.6 Classe Bot	29
3.3.7 Interface ServerMessageListener	30

3.4	Conception des graphiques des statistiques (Samy)	32
3.4.1	Les fichiers de score	32
3.4.2	Création de la classe Graphique	32
3.4.3	Modification de la classe WordleView	32
3.5	Conception et mécanisme des agents (Louis)	33
3.5.1	Classe : Agent	33
3.5.2	Classe : WordleView	33
4	Développement Final	34
4.1	Animations (Anthony)	34
4.1.1	Animations	34
4.1.2	Classe AnimationManager	34
4.1.3	Classe Animation	34
4.1.4	Classe FadeAnimation	35
4.1.5	Classe BounceAnimation	35
4.1.6	Classe ShakeAnimation	35
4.2	Gestion des Scores Multijoueur (Anthony)	36
4.2.1	Classe ScoreManager	36
4.2.2	Classe Score	36
4.2.3	Classe FileHandler	37
4.2.4	Classe ScoreComparator	37
4.3	Gestion des joueurs - retrait (Youssef)	38
4.3.1	Description	38
4.3.2	Classe Lobby	38
4.3.3	Classe Server	38
4.4	Démarrage de la partie (Youssef)	38
4.4.1	Description	39
4.4.2	Classe Lobby	39
4.4.3	Classe GameSession	39
4.4.4	Classe ClientHandler	39
4.5	Progression du jeu (Youssef)	40
4.5.1	Gestion de la déconnexion d'un joueur	40
4.5.2	Possibilité de rejoindre une partie après déconnexion	40
4.5.3	Synchronisation des clients	41
4.5.4	Gestion de la fin du jeu	41
4.5.5	Affichage du classement final du jeu	41
4.6	Gestion des Bots dans le mode Multijoueur (Louis)	43
4.6.1	Implémentation du Bouton de création de bot	43
4.6.2	Intégration des Bots aux Parties Multijoueur, méthode <i>ajoutDunBot</i>	43
4.6.3	Création de la Classe Bot	43
4.7	Gestion des Graphiques de Scores (Samy)	44
4.7.1	Implémentation des Graphiques de Scores Dynamiques	44
4.7.2	Boutons de Navigation par Mode de Jeu	44
4.7.3	Nouveaux Types de Graphiques	44
4.7.4	Sauvegarde et Chargement des Scores	44
5	Tests	45
5.1	Classe Server	45
5.1.1	Tests Unitaires	45
5.2	Classe Lobby	46
5.2.1	Tests unitaires	46
5.2.2	Tests d'intégration	46

5.3	Classe ClientHandler	47
5.3.1	Tests Unitaires	47
5.3.2	Tests d'Intégration	47
5.4	Classe GameConfig	47
5.4.1	Tests Unitaires	47
5.4.2	Tests d'Intégration	48
5.5	Classe GameInfo	48
5.5.1	Tests Unitaires	48
5.5.2	Tests d'Intégration	48
6	Conclusion	50
6.1	Conclusion générale de groupe	50
6.2	Conclusion personnelle finale (Youssef)	50
6.3	Conclusion personnelle (Anthony)	51
6.4	Conclusion personnelle (Samy)	52
6.5	Conclusion personnelle (Louis)	52
7	Annexes	53
7.1	Code source commenté et documentation technique	53
7.2	Copies d'écran de l'application	53
7.2.1	Écran d'Accueil	53
7.2.2	Fenêtre Multijoueur	54
7.2.3	Fenêtre de Création de Partie	55
7.2.4	Fenêtre pour Rejoindre une Partie	56
7.2.5	Lobby de la Partie	57
7.2.6	Fenêtre du Graphique des Scores	58
7.3	README du Projet	62
8	Références	63
8.1	Bibliothèques et outils utilisés	63
8.2	Sources documentaires et tutoriels suivis	64
9	Remerciements	66

1 Introduction

1.1 Wordle

1.1.1 Qu'est-ce que Wordle ?

Wordle est un jeu de réflexion basé sur les mots où le but est de deviner un mot cible à l'aide d'indices basés sur la couleur. Chaque guess correct rapproche le joueur du mot cible grâce à un système de feedback intuitif. Cette version du projet étend ce concept en mode multijoueur, permettant à plusieurs joueurs de se défier en temps réel, enrichissant l'expérience par la compétition et la collaboration.

1.1.2 Règles du Jeu

- Le joueur doit trouver un mot a x lettres en 5 essais au plus.
- Chaque lettre du mot proposé reçoit un retour coloré :
 - Une lettre colorée en vert est correcte et bien placée.
 - Une lettre colorée en orange est correcte mais mal placée.
 - Une lettre colorée en rouge n'est pas dans le mot.

En mode multijoueur, des règles supplémentaires sont intégrées pour synchroniser les essais entre les joueurs et gérer les interactions en temps réel.

1.1.3 Déroulement d'une Partie

Voici comment se déroule une partie typique de Wordle :

1. Un mot cible est choisi aléatoirement par le jeu et caché au joueur.
2. Le joueur saisit une première proposition de mot.
3. Le jeu affiche des indices colorés pour chaque lettre du mot proposé.
4. Avec ces indices, le joueur continue à proposer des mots jusqu'à trouver le mot cible ou jusqu'à épuiser ses essais.
5. Si le joueur devine le mot avant d'avoir utilisé ses essais, il gagne ; sinon, le mot est révélé et le jeu se termine.

Le mode multijoueur de Wordle transforme l'expérience du jeu solo traditionnel en un environnement compétitif en ligne où les joueurs peuvent interagir et s'affronter en temps réel. Chaque partie commence lorsque le maître de jeu, le premier joueur à rejoindre le salon, lance la session. Il détermine les paramètres de la partie, tels que la taille du mot, le nombre d'essais, la durée de la partie, la difficulté de l'agent joueur, et le nombre de joueurs. Un mot cible est généré aléatoirement par le serveur et tous les participants tentent de le deviner en même temps, en utilisant des indices colorés pour chaque lettre proposée. Ce système ajoute une dimension dynamique et stratégique à Wordle, rendant chaque partie unique et stimulante.

1.2 Présentation du projet

Ce rapport documente le développement d'un logiciel de jeu basé sur le concept de Wordle, spécifiquement adapté pour la langue française.

Ce semestre, notre projet a franchi une nouvelle étape en développant une version multijoueur du populaire jeu de mots Wordle. Ce nouveau cadre a introduit des interactions en ligne directes entre les joueurs, des fonctionnalités avancées de jeu, et l'innovante intégration d'un agent joueur, utilisant l'intelligence artificielle pour challenger les compétences linguistiques des utilisateurs.

Notre adaptation du projet Wordle a intégré des technologies de pointe, notamment les websockets pour la communication en temps réel et les frameworks d'intelligence artificielle

pour générer des comportements de jeu dynamiques et réactifs. Ces technologies ont été choisies pour répondre efficacement aux exigences des joueurs modernes en ligne, offrant une plateforme stable et interactive.

Notre version de Wordle est une application Java, qui utilise une architecture orientée objet pour une facilité de maintenance et d'extension.

Le logiciel se distingue par son approche innovante de l'assistance aux joueurs, exploitant le modèle Word2vec pour fournir des indices contextuellement pertinents.

1.3 Contexte et objectifs

L'engouement pour les jeux de mots et l'apprentissage de la langue a encouragé le développement d'applications ludiques stimulant l'esprit. Notre projet s'inscrit dans cette tendance, en proposant un défi linguistique quotidien qui est à la fois éducatif et divertissant.

L'objectif de ce semestre était de transformer Wordle en une expérience multijoueur en ligne enrichissante, qui non seulement teste le vocabulaire des utilisateurs mais leur permet aussi d'interagir dans un cadre compétitif. En mettant en œuvre des fonctionnalités telles que les parties multijoueurs et la gestion de la configuration des parties, ce projet vise à stimuler l'apprentissage linguistique et à encourager une compétition saine entre les joueurs.

1.4 Méthodologie et répartition des tâches (Intermédiaire)

Le projet a été conduit suivant une méthodologie agile, permettant une flexibilité et une adaptation continue face aux retours et aux exigences évolutives. L'équipe de développement est composée de quatre membres, chacun responsable d'une composante spécifique du logiciel :

- Anthony Genti a continué de jouer un rôle crucial dans le développement du backend du jeu, en prenant en charge des éléments essentiels tels que la gestion de la logique des parties (GameSession), la configuration des jeux (GameConfig), et l'interface de communication (ServerMessageListener). Sa contribution a été fondamentale pour le bon fonctionnement des interactions multijoueurs et la gestion de l'information de jeu (GameInfo).
- Youssef Boudount a pris la responsabilité du développement client-serveur, en gérant à la fois le Client et le ClientHandler, ainsi que l'ensemble de l'infrastructure serveur (Server). Il a également été impliqué dans la création des interfaces utilisateur (CreateGameUI, GameLobbyUI, JoinLobbyUI), assurant une expérience utilisateur cohérente et intégrée.
- Louis Rives-Lehtinen, nouvellement intégré à l'équipe, a été chargé de développer des agents joueurs intelligents. Cette tâche comprenait la création d'un algorithme capable de jouer de manière autonome au jeu et de s'adapter à différents niveaux de difficulté, permettant ainsi une interaction enrichissante en partie multijoueur.
- Seghir Samy, également nouveau dans l'équipe, a pris en charge le développement des statistiques et des graphiques des performances des joueurs, ce qui permet de suivre et d'analyser l'évolution des compétences des utilisateurs au fil du temps.

Ces changements et ces réaffectations de tâches ont permis une diversification des compétences au sein de notre équipe et une meilleure adaptation aux défis techniques et conceptuels rencontrés lors du développement du projet. Chaque membre a contribué de manière significative à la réalisation des objectifs du projet, en s'assurant que toutes les composantes du logiciel fonctionnent harmonieusement ensemble et répondent aux exigences de notre base d'utilisateurs en expansion.

Cette phase finale du projet a également mis en évidence les défis de communication et de coordination dans un environnement de développement agile, en particulier avec l'intégration de nouveaux membres. Nos efforts pour maintenir une méthodologie

flexible et réactive nous ont cependant permis de surmonter ces obstacles et de progresser efficacement vers la réalisation de notre projet.

1.5 Méthodologie et répartition des tâches (Finale)

Pour cette phase finale du projet Wordle Multijoueur, nous avons maintenu notre approche agile, permettant des ajustements continus en réponse aux retours et aux évolutions des exigences. La répartition des tâches entre les membres de l'équipe a été affinée pour garantir une couverture complète et efficace des différentes composantes du projet. Voici un aperçu des responsabilités de chacun :

- **Anthony Genti** a continué à enrichir le backend du jeu en se concentrant sur les animations. Il a développé et intégré plusieurs classes d'animations (AnimationManager, FadeAnimation, BounceAnimation, ShakeAnimation), améliorant l'expérience utilisateur avec des effets visuels dynamiques et engageants.
- **Youssef Boudount** a pris en charge la gestion des sessions de jeu et des interactions réseau. Il a travaillé sur les classes essentielles telles que Server, Lobby, GameSession et ClientHandler, assurant une gestion fluide des connexions et des déconnexions des joueurs. Il a également optimisé la synchronisation des clients et le traitement des états de jeu, garantissant une expérience multijoueur cohérente et réactive.
- **Louis Rives-Lehtinen** s'est concentré sur le développement des bots pour le mode multijoueur. Il a implémenté le bouton de création de bots, intégré ces derniers dans les parties multijoueur et développé la classe Bot avec des algorithmes adaptés à différents niveaux de difficulté, enrichissant ainsi les interactions de jeu et offrant de nouveaux défis aux joueurs.
- **Seghir Samy** a été responsable de la gestion et de l'affichage des scores des joueurs. Il a mis en œuvre des graphiques dynamiques pour visualiser les performances des joueurs, ajouté des boutons de navigation pour différents modes de jeu, et développé des fonctionnalités pour la sauvegarde et le chargement des scores. Ces contributions ont permis une analyse détaillée des compétences des utilisateurs et ont enrichi l'aspect compétitif du jeu.

Cette répartition des tâches a permis de tirer parti des compétences spécifiques de chaque membre, tout en assurant une couverture complète des besoins du projet. La collaboration harmonieuse et le soutien mutuel au sein de l'équipe ont été cruciaux pour surmonter les défis techniques et conceptuels rencontrés, aboutissant à une version finale du jeu qui répond aux attentes de notre base d'utilisateurs en constante expansion.

1.6 Organisation interne et flux de travail

Cette section explique nos conventions de travail et les commandes Git utilisées. Nous avons pris soin d'assurer que chacun travaille dans sa propre branche pour éviter tout conflit et permettre une revue de code efficace avant l'intégration dans la branche principale.

Pour une gestion efficace du projet, nous avons établi un guide d'organisation clair, "ORGANISATION.md", que nous avons mis à disposition dans la branche principale dès la création du dépôt.

Voici le processus que chaque membre de l'équipe a suivi :

1.6.1 Clonage du projet

1. Ouvrez un terminal (invite de commande) sur votre ordinateur.
2. Naviguez vers le répertoire où vous souhaitez cloner le projet.
3. Utilisez la commande `git clone https://gitlab.com/ceri-projet-programmation-2023/semestre2-groupe3.git`.

1.6.2 Structure des branches

Chaque membre du groupe dispose d'une branche dédiée pour le développement, permettant un travail indépendant sans interférence :

- main
- samy-seghir
- youssef-boudount
- louis-rives
- anthony-genti

1.6.3 Travailler dans sa propre branche

1. Après avoir cloné le projet, naviguez dans le répertoire du projet avec la commande `cd`.
2. Basculez sur votre branche personnelle avec la commande suivante :

```
1 git checkout `nom-de-votre-branche-personnelle`
```

1.6.4 Ajout de modifications

1. Travaillez sur vos fichiers locaux dans votre branche.
2. Ajoutez toutes les modifications en préparation pour le prochain commit avec la commande suivante :

```
1 git add .
```

1.6.5 Commit des modifications

1. Committez vos modifications avec un message explicatif en utilisant la commande suivante :

```
1 git commit -m "Votre message descriptif"
```

1.6.6 Pousser les modifications sur GitLab

1. Pousser vos changements sur votre branche GitLab avec la commande suivante :

```
1 git push origin `nom-de-votre-branche-personnelle`
```

2. Assurez-vous d'utiliser le nom correct de votre branche personnelle.

1.6.7 Gestion des identifiants GitLab

Pour faciliter le processus de poussée, utilisez un gestionnaire de trousseaux avec la commande suivante :

```
1 git config --global credential.helper manager
```


2 Conception

2.1 Architecture générale du logiciel

Notre projet s'appuie sur une architecture Modèle-Vue-Contrôleur (MVC) qui divise l'application en trois composants interconnectés. Cette séparation permet de gérer les responsabilités de manière efficace, en isolant la logique métier de l'interface utilisateur et du contrôle de l'application.

Pour ce semestre, nous avons revu l'architecture de notre application pour intégrer le jeu en ligne, tout en maintenant la structure Modèle-Vue-Contrôleur (MVC) avec des adaptations pour le multijoueur.

Modèle (Model) : Ce package a été conservé pour gérer la logique liée aux données du jeu. Il comprend :

- `FrenchWordChecker.java` : Vérifie la validité des mots français.
- `RandomWordSelector.java` : Sélectionne aléatoirement les mots pour le jeu.
- `WiktionaryScraper.java` : Extrait des mots et leurs définitions du Wiktionnaire.

Vue (View) : Scindée en deux sous-packages pour différencier les interfaces locales et en ligne.

- **localView** : Contient les éléments d'interface pour le mode solo local avec des classes comme `CustomDialog.java` pour les dialogues, et `Main.java` pour la fenêtre principale.
- **onlineView** Gère les interfaces spécifiques au jeu en ligne avec `CreateGameUI.java`, `GameLobbyUI.java`, et `JoinLobbyUI.java`.

Contrôleur (Controller) : Facilite l'interaction entre le modèle et les vues et gère la logique du jeu en réseau.

- `WordleGame.java` : Coordonne les interactions de jeu de base.
- `Score.java` : Gère le calcul et l'affichage des scores.

Online : Nouveau package ajouté pour gérer spécifiquement les fonctionnalités en ligne.

- Contient des classes comme `Client.java`, `Server.java`, et `Lobby.java` pour la gestion de la communication réseau, la coordination des sessions de jeu, et l'administration des joueurs en ligne.

2.2 Diagrammes de conception (UML)

Dans cette section, nous présentons une série de diagrammes UML qui décrivent la conception architecturale et le comportement des composants de notre système de jeu en ligne. Les diagrammes fournissent une vue d'ensemble des interactions entre les différents éléments du système et mettent en évidence les flux de processus pour les scénarios de jeu clés.

2.2.1 Diagramme de classes

Le diagramme de classes (Figure 9) illustre la structure des packages et les relations entre les classes. Chaque classe est représentée avec ses attributs et méthodes significatifs, fournissant un aperçu de la hiérarchie et des dépendances au sein du système.

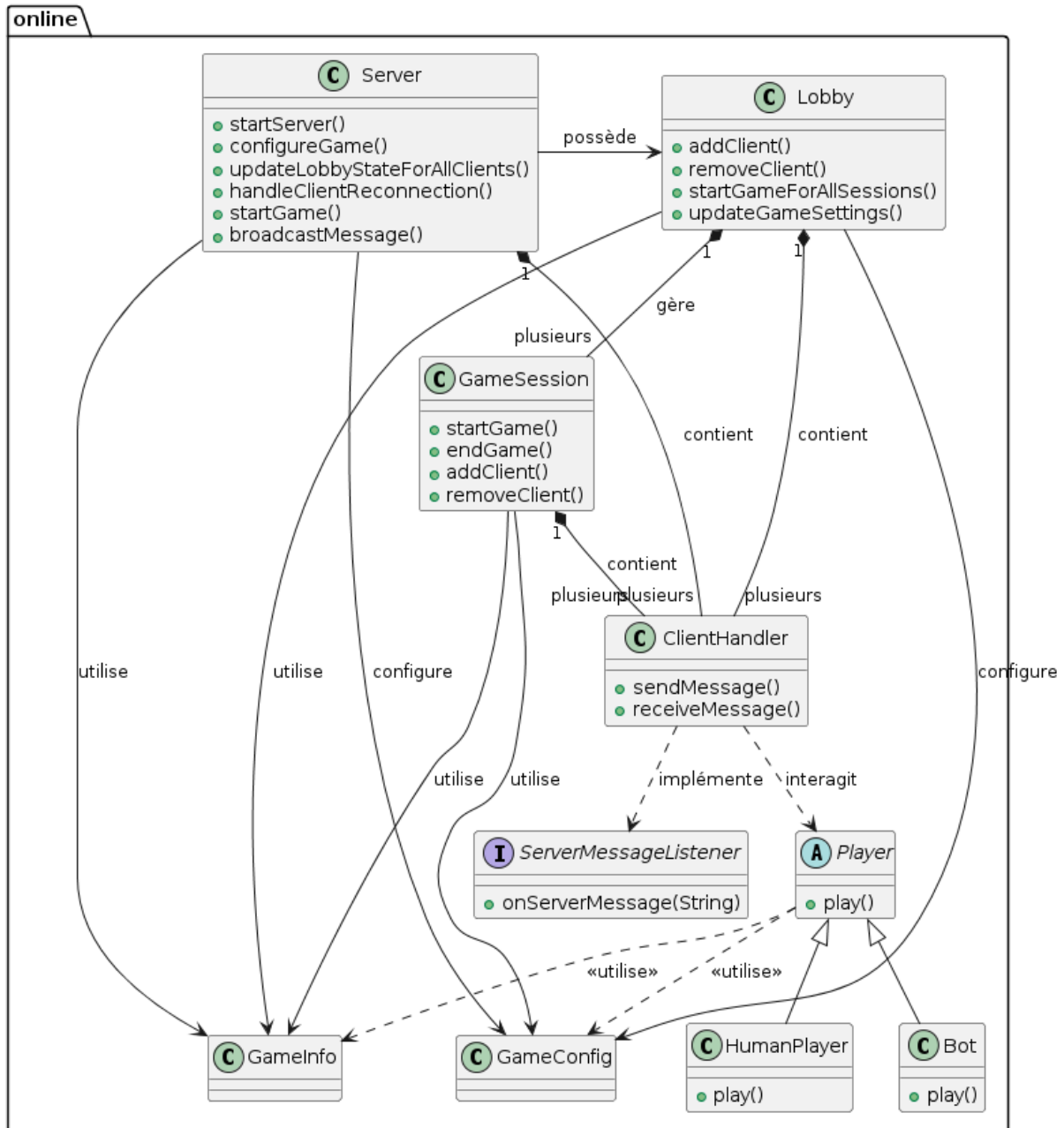


Figure 1. Diagramme de classes représentant la structure et les relations des composants du système.

2.2.2 Diagramme de séquence

Le diagramme de séquence (Figure 2) détaille le flux d'actions pour des scénarios typiques de jeu en ligne, tels que la connexion au serveur, la participation à une partie et la gestion des déconnexions.

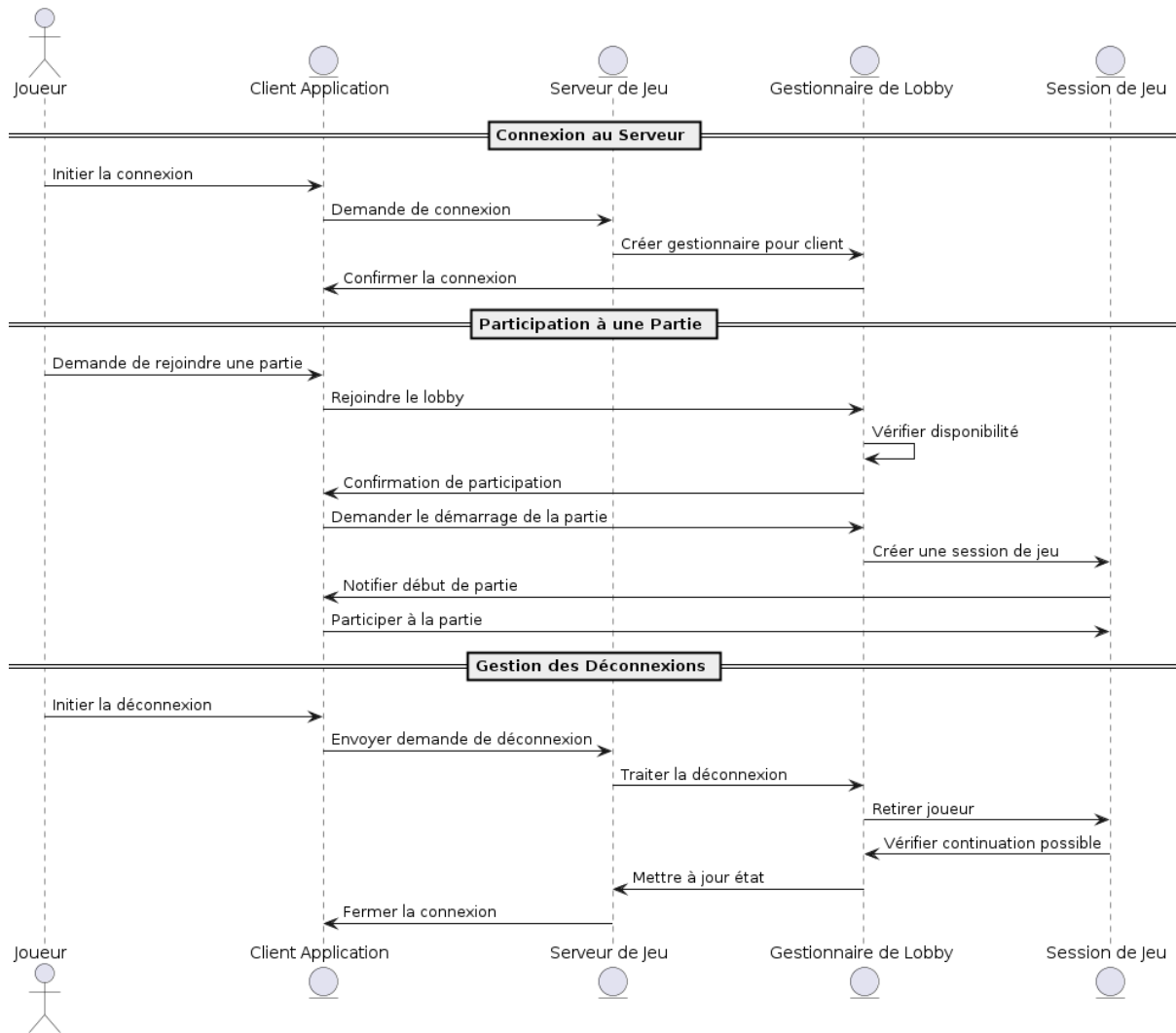


Figure 2. Diagramme de séquence montrant les interactions entre le joueur, l'application client et le serveur.

2.2.3 Diagramme d'activité pour l'hôte

Le diagramme d'activité dédié au joueur hôte (Figure 3) décrit les étapes que l'hôte suit pour configurer et lancer une partie multijoueur.

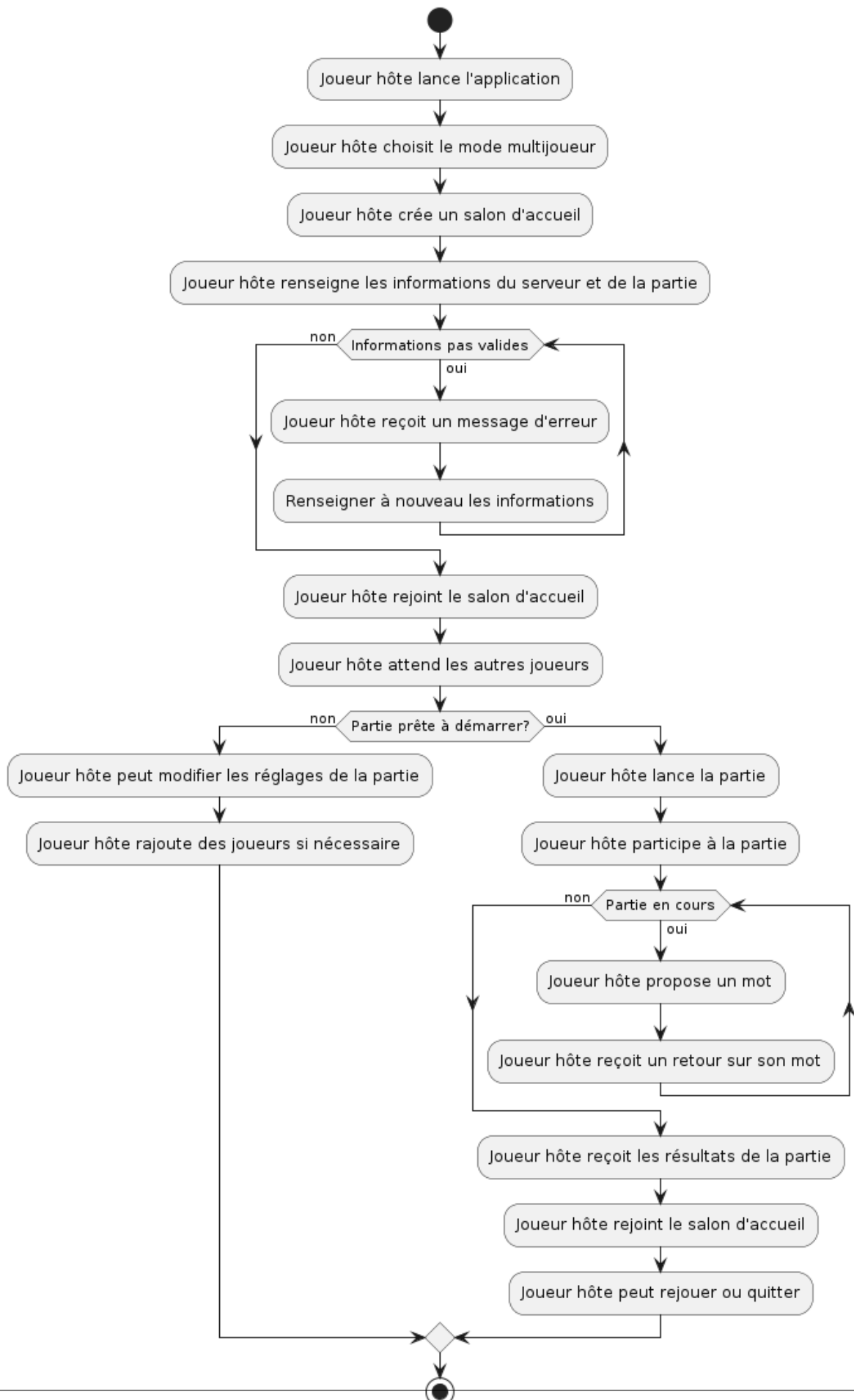
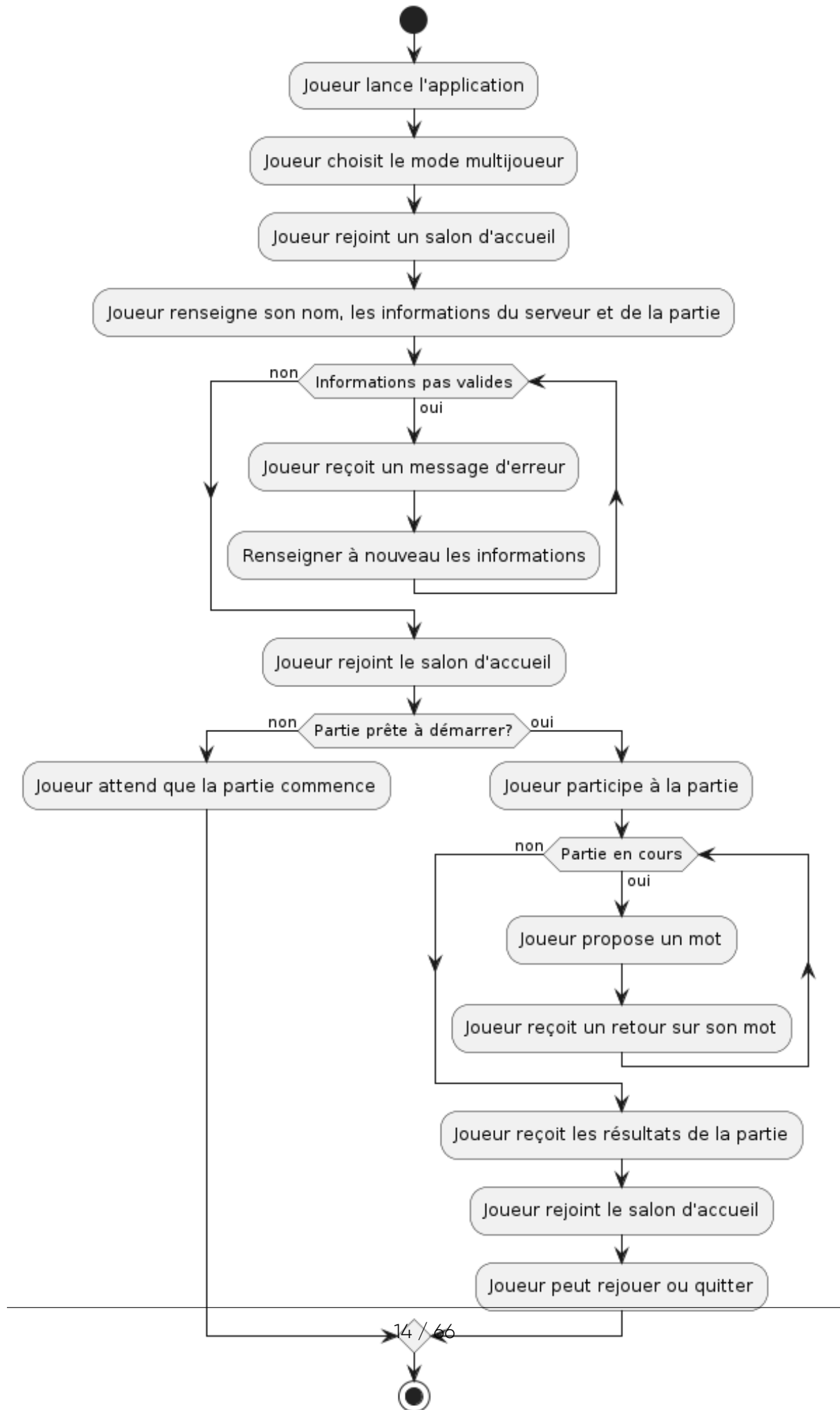


Figure 3. Diagramme d'activité pour le joueur hôte, décrivant la création et la gestion d'une session de jeu multijoueur.

2.2.4 Diagramme d'activité pour le participant

De même, le diagramme d'activité pour le joueur participant (Figure 4) présente les actions entreprises par un joueur lorsqu'il rejoint et participe à une partie multijoueur.



2.3 Interface utilisateur et expérience utilisateur

Pour le deuxième semestre, l'interface utilisateur a été adaptée pour supporter à la fois le jeu local et en ligne.

Design adaptatif : L'interface s'ajuste pour offrir une expérience optimale que ce soit sur un poste fixe ou en ligne, avec des éléments d'interface qui réagissent de manière fluide aux actions des utilisateurs.

Expérience multijoueur : Des efforts ont été faits pour rendre l'expérience en ligne aussi intuitive que le jeu local, avec des ajouts tels que des salles d'attente virtuelles, des notifications en temps réel, et des chats en jeu pour améliorer l'interaction et la communication entre les joueurs.

Retours visuels et sonores : En mode en ligne, des retours enrichis aident à garder les joueurs engagés et informés de l'état du jeu, améliorant ainsi l'immersion et la réactivité de l'application.

3 Développement Intermediaire

3.1 Déroulement de la partie (Youssef)

3.1.1 Classe Server

La classe **Server** joue un rôle central dans notre système de jeu multijoueur pour Wordle, en gérant les interactions réseau et en orchestrant la logique de session de jeu. Cette classe est conçue pour être robuste et efficace, afin de gérer simultanément plusieurs clients dans un environnement multijoueur dynamique.

Conception et Fonctionnalités La **Server** est responsable de l'initialisation du serveur, de la gestion des connexions client, et de la coordination des différentes parties du jeu. Elle utilise un **ServerSocket** pour écouter et accepter les connexions entrantes sur un port spécifié. Une fois une connexion établie, elle crée et gère un **ClientHandler** pour chaque client, permettant une communication asynchrone et non bloquante.

Principales Méthodes

- **startServer()** : Cette méthode démarre le serveur pour écouter et accepter les connexions entrantes. Elle crée des instances de **ClientHandler** pour gérer les interactions avec chaque client connecté.
- **configureGame()** : Permet la configuration dynamique des paramètres de jeu tels que le nombre maximum de joueurs par session, la durée du jeu, et le niveau de difficulté. Cette flexibilité est cruciale pour s'adapter aux préférences des utilisateurs et aux exigences de gameplay variées.
- **updateLobbyStateForAllClients()** : Synchronise l'état du lobby actuel avec tous les clients connectés, assurant que chaque joueur ait une vue cohérente de l'état du jeu.
- **removeClient()** et **removePlayer()** : Ces méthodes gèrent la déconnexion des clients et la suppression des joueurs du lobby, garantissant la gestion appropriée des ressources et la stabilité du serveur.

Intégration et Interaction La classe **Server** interagit étroitement avec plusieurs autres composants du système, notamment **Lobby**, **ClientHandler**, et **GameSession**. Elle agit comme un médiateur entre ces composants, facilitant et orchestrant le flux d'informations. Le **Lobby** gère les sessions de jeu actives et les clients en attente, tandis que chaque **ClientHandler** s'occupe des communications réseau spécifiques à chaque client.

Stratégie de Développement La mise en œuvre de **Server** a suivi une approche orientée objet stricte, avec un accent sur l'encapsulation, la modularité, et la réutilisabilité des composants. Les choix de conception ont été guidés par le besoin de maintenabilité et d'extensibilité du système. Par exemple, l'utilisation de **ExecutorService** pour gérer les threads de manière efficace et sûre minimise les risques de conditions de concurrence et optimise la performance du serveur.

3.1.2 Classe Lobby

La classe **Lobby** joue un rôle central dans notre projet de jeu multijoueur en ligne, en gérant l'organisation des sessions de jeu et la coordination des joueurs. Cette classe est essentielle pour assurer une expérience fluide et organisée pour les utilisateurs qui attendent de jouer ou qui participent à une partie.

Objectif de la Classe L'objectif principal de la classe **Lobby** est de servir de point de rassemblement pour les joueurs en ligne, en gérant les sessions de jeu actives et les clients en attente. Elle organise les joueurs en sessions de jeu basées sur les configurations définies et gère le cycle de vie complet des parties, de la création à la conclusion.

Fonctionnalités Clés

Gestion des Clients et des Sessions :

- **Clients en attente :** Le lobby maintient une liste de `waitingClients` qui sont les joueurs en attente d'entrer dans une partie.
- **Sessions actives :** Une liste de `activeSessions` garde une trace des parties en cours, permettant de gérer simultanément plusieurs sessions de jeu.

Configuration de la Partie : Chaque lobby est initialisé avec une configuration spécifique (`GameConfig`), qui détaille des paramètres tels que le nombre maximum de joueurs par session, la durée du jeu, et le niveau de difficulté.

Démarrage et Gestion des Parties : Le lobby vérifie constamment si suffisamment de joueurs sont présents pour démarrer une nouvelle session et procède à la création de sessions de jeu via `createNewGameSession`. Les sessions de jeu sont gérées de manière à permettre une transition fluide des joueurs de la liste d'attente aux sessions actives.

Méthodes Importantes

- `addClient(ClientHandler clientHandler)` : Ajoute un nouveau client au lobby. Si le lobby est plein, le client reçoit un message d'erreur.
- `removeClient(ClientHandler clientHandler)` : Retire un client du lobby et met à jour l'état des sessions actives si nécessaire.
- `startGameForAllSessions()` : Démarre toutes les sessions de jeu actives une fois que les conditions requises sont remplies.
- `updateLobbyStateForAllClients()` : Met à jour et notifie tous les clients de l'état actuel du lobby, assurant que tous les participants ont les informations les plus récentes.

Stratégie de Développement La mise en œuvre de Lobby a été guidée par les principes suivants :

- **Robustesse :** Gestion des erreurs et des états inattendus pour éviter les crashes du serveur et assurer une expérience utilisateur continue.
- **Modularité :** Conception de la classe de manière à faciliter les modifications et l'ajout de fonctionnalités sans perturber les fonctionnalités existantes.
- **Performance :** Optimisation des méthodes pour gérer efficacement un grand nombre de joueurs et de sessions simultanément.

3.1.3 Classe ClientHandler

Objectif de la Classe La classe `ClientHandler` joue un rôle crucial dans la gestion des interactions entre le client et le serveur dans notre jeu multijoueur en ligne. Elle sert d'intermédiaire entre le client et le serveur, traitant toutes les communications entrantes et sortantes. Cela comprend la réception de commandes du client, le traitement de ces commandes, et l'envoi de réponses ou de notifications pertinentes.

Fonctionnalités Clés**Gestion des Communications :**

- Réception et traitement des messages envoyés par le client.
- Envoi de réponses au client en fonction des actions demandées ou des événements survenus sur le serveur.

Gestion de l'État du Client : Maintient des informations sur l'état actuel du client, telles que le nom du joueur, l'état de connexion, et le score du joueur.

Gestion des Exceptions et de la Stabilité :

- Gère les exceptions liées aux opérations réseau pour éviter les plantages du serveur.
- Assure une déconnexion propre et informe le serveur lorsqu'un client se déconnecte.

Méthodes Importantes

- `run()` : Méthode principale du thread qui écoute les messages entrants du client et appelle `processCommand()` pour leur traitement.
- `sendMessage(String message)` : Envoie des messages au client. Utilisée pour envoyer des notifications ou des réponses aux requêtes.
- `closeEverything()` : Ferme toutes les connexions et les flux pour s'assurer qu'aucune ressource n'est laissée ouverte après la déconnexion d'un client.
- `processCommand(String message)` : Interprète les commandes reçues du client et effectue les actions appropriées, comme démarrer une partie, mettre à jour la configuration, ou gérer les messages de chat.

Stratégie de Développement La mise en œuvre de `ClientHandler` a été guidée par plusieurs principes clés :

- **Robustesse et Gestion des Erreurs** : Assurer que la classe peut gérer des scénarios d'erreur sans interruption de service, par exemple lors de la perte de la connexion réseau.
- **Sécurité et Validation** : Vérifier que toutes les entrées reçues des clients sont validées avant traitement pour prévenir des manipulations malveillantes.
- **Performance et Scalabilité** : Optimiser la gestion des threads pour supporter plusieurs clients simultanément sans dégradation des performances.

Détails de Conception

- **Gestion des Threads** : Chaque instance de `ClientHandler` s'exécute dans son propre thread, permettant ainsi un traitement asynchrone des requêtes client.
- **Modularité et Clarté** : Les responsabilités sont clairement délimitées au sein de la classe pour faciliter les modifications et la maintenance. Par exemple, la séparation des méthodes pour l'envoi de messages et la gestion des commandes simplifie la compréhension du flux de traitement.
- **Observation des Bonnes Pratiques** : Application des bonnes pratiques comme la fermeture des ressources dans un bloc `finally` pour garantir que toutes les ressources sont libérées correctement, même en cas d'erreur.

3.1.4 Classe Client

Objectif de la Classe La classe `Client` incarne le point d'entrée pour les utilisateurs qui se connectent au serveur afin de participer au jeu en ligne. Cette classe gère la communication entre le client et le serveur, facilitant les interactions nécessaires pour jouer, rejoindre des lobbies, et recevoir des mises à jour de l'état du jeu.

Fonctionnalités Clés**Gestion de la Connexion :**

- Établissement de la connexion avec le serveur via sockets.
- Gestion des entrées et sorties de flux de données pour la communication.

Gestion des Événements :

- Réception et traitement des messages venant du serveur.
- Envoi de commandes telles que les demandes de connexion, de chat, et les mises à jour de jeu.

Interface Utilisateur : Interaction avec l'interface graphique, permettant de refléter les changements d'état du jeu en temps réel.

Méthodes Importantes

- **connect()** : Connecte le client au serveur en utilisant l'adresse et le port spécifiés. Initialise les flux d'entrée et de sortie.
- **sendMessage(String message)** : Envoie des messages au serveur, utilisée pour toutes formes de communications émanant du client.
- **listenForServerMessages()** : Lance un nouveau thread qui écoute en continu les messages du serveur, garantissant que l'interface utilisateur soit mise à jour en temps réel.

Stratégie de Développement Le développement de la classe **Client** a été guidé par la nécessité d'assurer une communication efficace et fiable :

- **Robustesse** : Gestion des exceptions pour prévenir des crashes lors de problèmes de réseau.
- **Réactivité** : Utilisation de threads pour maintenir l'interface utilisateur réactive et à jour avec l'état du serveur.
- **Sécurité** : Validation des entrées pour éviter des injections ou autres failles de sécurité lors des échanges de données.

Détails de Conception

- **Modèle Thread** : Utilisation de threads pour écouter les messages du serveur sans bloquer l'interface utilisateur principale, permettant une expérience utilisateur fluide.
- **Gestion des Ressources** : Fermeture appropriée des sockets et des flux pour éviter les fuites de mémoire lors de la déconnexion.
- **Séparation des préoccupations** : Distinction claire entre la logique de réseau et la logique de jeu, facilitant les tests et la maintenance.

3.2 Création d'un salon d'accueil (Youssef)

3.2.1 Classe CreateGameUI

La classe `CreateGameUI` de notre application client-serveur est conçue pour faciliter la création d'un serveur de jeu dans l'interface utilisateur graphique. Elle permet aux utilisateurs de configurer et de lancer une session de jeu en spécifiant les paramètres tels que le port, le nombre de joueurs, la difficulté du jeu, et plus encore.

Fonctionnalités Clés

Configuration du Serveur : Permet aux utilisateurs de saisir les paramètres nécessaires pour lancer un serveur, incluant l'adresse IP, le port, et les détails de la partie.

Validation des Entrées : Assure que les entrées fournies par l'utilisateur sont valides et conformes aux attentes avant de tenter de lancer le serveur.

Démarrage du Serveur : Intègre la logique pour initialiser un serveur basé sur les paramètres fournis et gérer le démarrage du serveur en arrière-plan.

Gestion des Erreurs : Fournit des retours visuels en cas d'erreurs de configuration pour aider les utilisateurs à corriger les problèmes.

Méthodes Importantes

- `start(Stage primaryStage)` : Initialise l'interface utilisateur pour la création du serveur, préparant le terrain pour la saisie des paramètres du serveur.
- `createServer(int port, String playerName, String difficulty, int maxPlayers)` : Configure et démarre une instance de serveur basée sur les paramètres saisis.
- `getContent(Stage primaryStage, Consumer<Client> onLobbyReady)` : Construit les éléments graphiques nécessaires pour l'interface de création du serveur, y compris les champs de saisie et les boutons.

Stratégie de Développement

Le développement de `CreateGameUI` a été guidé par la nécessité de fournir une expérience utilisateur claire et directe pour la configuration du serveur :

Interface Utilisateur Intuitive : Utilisation de JavaFX pour créer une interface graphique conviviale qui guide les utilisateurs à travers le processus de configuration du serveur.

Validation Robuste : Implémentation de contrôles rigoureux pour la validation des entrées, s'assurant que les paramètres du serveur sont valides avant de tenter de démarrer le serveur.

Gestion Asynchrone : Démarrage des opérations serveur dans des threads séparés pour éviter de bloquer l'interface utilisateur, permettant une réactivité continue pendant la configuration.

Détails de Conception

Modularité et Réutilisabilité : Conception de la classe pour permettre la réutilisation dans d'autres parties de l'application, comme la reconfiguration du serveur existant ou le redémarrage après un échec.

Sécurité et Gestion des Erreurs : Inclut des mesures pour gérer les exceptions et les erreurs de réseau, fournissant des alertes contextuelles pour aider les utilisateurs à résoudre les problèmes.

Gestion des événements

- **createServerButton** : Lorsqu'activé, ce bouton déclenche la vérification des entrées et, si elles sont valides, procède à la création d'un serveur et à l'initialisation d'une session de jeu.
- **createServer()** : La méthode est appelée pour établir un nouveau serveur en utilisant les informations fournies, et gère le démarrage du serveur et la connexion initiale du client.

3.2.2 Classe JoinLobbyUI

La classe `JoinLobbyUI` est essentielle pour intégrer des fonctionnalités de réseau dans notre projet de jeu Wordle en ligne. Cette interface utilisateur permet aux joueurs de rejoindre un lobby existant en fournissant des informations telles que l'adresse IP du serveur, le port et le nom du joueur. L'objectif est de faciliter l'accès aux parties multijoueur en rendant le processus aussi intuitif et robuste que possible.

Fonctionnalités Clés

Interface Graphique pour la Connexion : Fournit des champs de texte pour saisir l'adresse IP, le port du serveur et le nom du joueur, permettant une connexion directe à un lobby de jeu.

Validation des Entrées : Implémente des contrôles pour s'assurer que les informations saisies sont dans un format correct avant d'établir la connexion, évitant ainsi les erreurs courantes comme les adresses IP mal formatées ou les ports non numériques.

Connexion au Serveur : Gère l'initialisation d'un client et sa connexion au serveur avec les paramètres fournis, facilitant une intégration transparente dans le lobby du jeu.

Gestion des Erreurs et Feedback Utilisateur : Utilise des alertes et notifications pour informer l'utilisateur des réussites de connexion ou des problèmes rencontrés, améliorant l'expérience utilisateur et aidant à la résolution rapide des problèmes.

Architecture et Conception

Méthode `start(Stage primaryStage)` : Configure et affiche la fenêtre principale de l'application JavaFX, établissant le contenu initial et les comportements de la fenêtre.

Méthode `joinLobby(String serverAddress, String playerName, int serverPort)` : Crée une instance de `Client` qui tente de se connecter au serveur avec les détails fournis. Cette méthode encapsule la logique de connexion réseau, permettant de séparer clairement la logique d'interface utilisateur de la logique réseau.

Méthode `getContent(Stage primaryStage, Consumer<Client> onLobbyReady)` : Construit les éléments visuels de l'interface utilisateur et associe des actions aux boutons, y compris la logique pour rejoindre le lobby. Cette méthode illustre comment l'architecture modulaire peut faciliter la gestion des composants UI et des événements.

Stratégie de Développement

Le développement de `JoinLobbyUI` a été orienté par plusieurs principes clés pour assurer une implémentation efficace et fiable :

Principes d'Interface Utilisateur Réactive : Utilise `Platform.runLater()` pour s'assurer que les modifications de l'interface utilisateur et les alertes sont exécutées de manière sûre sur le thread de l'application JavaFX, évitant les erreurs de concurrence et les problèmes de performance.

Validation Robuste des Données : Emploie des expressions régulières et des vérifications de formats pour prévenir les erreurs courantes avant même que la connexion réseau ne soit tentée, réduisant ainsi le risque d'échecs de connexion dus à des erreurs utilisateur.

Gestion des Erreurs : Implémente des gestionnaires d'erreurs qui alertent les utilisateurs en cas de problèmes, permettant des corrections immédiates et améliorant la robustesse de l'application.

Détails de Conception

Séparation des Préoccupations : Distincte la logique de connexion réseau de l'interface utilisateur, ce qui facilite les tests, la maintenance et les mises à jour futures de l'application.

Utilisation des Lambdas et des Fonctions de Haute Ordre : Exploite les expressions lambda et les `Consumer` pour gérer les événements de manière flexible et réutilisable, montrant un usage avancé des fonctionnalités Java 8+.

Feedback Visuel et Interactivité : Intègre des feedbacks immédiats via des dialogues d'alerte pour guider et informer les utilisateurs pendant le processus de connexion, renforçant l'engagement et la satisfaction utilisateur.

3.2.3 Classe `GameLobbyUI`

La classe `GameLobbyUI` constitue une interface utilisateur clé dans notre application multijoueur pour le jeu Wordle. Elle sert de salon d'accueil pour les joueurs en ligne, offrant une plateforme où les joueurs peuvent se regrouper avant le début de la partie. Cette interface permet aux joueurs de voir qui est connecté, de discuter via un chat, de configurer les paramètres du jeu si nécessaire, et finalement de lancer la partie.

Conception de la Classe

Extensions et Interfaces : La classe étend `Application`, ce qui est standard pour toute interface utilisateur JavaFX. Elle implémente `ServerMessageListener`, permettant à la classe de réagir aux messages du serveur concernant les mises à jour de l'état du lobby et du jeu.

Gestion de l'État du Lobby : Les attributs clés comme `playerList`, `maxPlayers`, et `isHost` permettent de stocker l'état du lobby, qui influence directement la façon dont les divers éléments de l'UI sont affichés et interactifs.

Composants UI Principaux :

- **Liste des Joueurs :** Utilise `ListView` pour afficher une liste dynamique des joueurs connectés.
- **Zone de Chat :** Comprend une `TextArea` pour l'affichage des messages et un `TextField` pour la saisie de messages.

- **Boutons de Contrôle :** Inclut des boutons pour démarrer le jeu, quitter le lobby, et, pour l'hôte, modifier les paramètres du jeu.

Fonctionnalités Détaillées

Mise à jour dynamique des joueurs : La méthode `handlePlayerListUpdate` traite les messages du serveur indiquant les changements dans la liste des joueurs, mettant à jour la `ListView` et les carrés représentant les joueurs.

Gestion des messages de chat : Intègre une fonctionnalité de chat permettant aux joueurs de communiquer. Les messages sont envoyés au serveur et diffusés à tous les clients.

Configuration du jeu : Permet à l'hôte de modifier des paramètres tels que la difficulté du jeu, la durée et le nombre de mots à trouver. Ces paramètres sont ajustables via une `Dialog` qui recueille les entrées de l'utilisateur et envoie les modifications au serveur via `sendGameConfiguration`.

Démarrage et Quitte du jeu : L'hôte peut démarrer le jeu pour tous les joueurs en envoyant un message spécifique au serveur. Tous les joueurs peuvent quitter le lobby via un bouton qui envoie également un message au serveur.

Gestion des Interactions Utilisateur

Réactions aux Actions Utilisateur : Les boutons et champs interactifs appellent des méthodes spécifiques qui traitent l'action (ex. envoi de messages, mise à jour de configurations) et communiquent avec le serveur au besoin.

Écoute des Messages du Serveur : Implémente un système d'écoute continu des messages du serveur, permettant une mise à jour en temps réel de l'interface en fonction des changements de l'état du jeu et des communications entre joueurs.

Design et Style

Styles CSS : Utilise des instructions de style inline pour définir l'apparence des composants de l'interface, tels que les couleurs, les bordures et les polices, assurant une cohérence visuelle et une expérience utilisateur engageante.

Responsive Design : L'interface s'ajuste dynamiquement à la taille de la fenêtre, assurant que les composants sont disposés de manière esthétique et fonctionnelle quelles que soient les dimensions de la fenêtre.

Stratégies de Développement Employées

Séparation des Préoccupations : Distincte la logique de gestion de l'interface utilisateur et la communication réseau, facilitant le développement et la maintenance.

Utilisation de JavaFX et de ses fonctionnalités : Exploite pleinement les capacités de JavaFX pour créer une interface riche et interactive.

Gestion d'État Robuste : Maintient un état cohérent de l'application à travers des structures de données appropriées et des mises à jour régulières basées sur les interactions serveur-client.

3.3 Déroulement de la partie (Anthony)

3.3.1 Classe Player

La classe **Player** est conçue pour représenter un joueur dans notre application de jeu multijoueur basée sur le jeu Wordle. Cette classe encapsule toutes les informations et comportements essentiels à un joueur, comme son nom, son score, et son état (si le joueur est actuellement engagé dans une partie). Elle est définie comme une classe abstraite pour permettre la spécialisation en différents types de joueurs, tels que des joueurs humains ou des agents automatisés, qui peuvent avoir des stratégies de jeu distinctes.

Structure et Conception

Attributs Principaux :

- **name** : String représentant le nom du joueur.
- **score** : Entier représentant le score accumulé du joueur durant le jeu.
- **isInGame** : Booléen qui indique si le joueur est actuellement dans une partie.

Constructeur : Le constructeur prend un argument **name** pour initialiser le joueur. Le score est initialisé à zéro et **isInGame** est mis à faux, reflétant que le joueur n'est pas encore engagé dans une partie au moment de la création.

Fonctionnalités et Méthodes

Accesseurs et Mutateurs :

- **getName()** : Renvoie le nom du joueur. Utile pour l'affichage des noms dans l'interface utilisateur ou lors du stockage des scores.
- **getScore()** et **setScore(int score)** : Permettent respectivement de récupérer et de définir le score du joueur, ce qui est crucial pour la gestion des points durant le jeu.
- **isInGame()** et **setInGame(boolean isInGame)** : Fournissent les moyens de vérifier et modifier l'état de participation du joueur, ce qui aide à contrôler le flux du jeu en permettant ou en interdisant des actions basées sur si le joueur est dans une partie ou non.

Gestion du Score :

- **addScore(int increment)** : Ajoute un score au score actuel. Cette méthode est appelée lorsqu'un joueur atteint un objectif dans le jeu, par exemple, trouve un mot correctement dans Wordle.
- **resetScore()** : Réinitialise le score du joueur à zéro, utilisée typiquement à la fin d'une partie ou lors du démarrage d'une nouvelle partie.

Jouabilité : **play()** : Méthode abstraite qui doit être implémentée par des sous-classes pour définir la logique spécifique de jeu pour différents types de joueurs. Cette approche offre une grande flexibilité pour intégrer diverses stratégies de jeu et comportements de joueur.

Utilisation de la Classe dans le Projet

Intégration avec l'Interface Utilisateur : La classe **Player** est utilisée pour lier les données de joueur avec les composants de l'interface utilisateur, tels que les listes de joueurs dans le lobby ou les affichages de score.

Extension pour Divers Types de Joueurs : En tant que classe abstraite, `Player` sert de base pour des extensions spécifiques comme `HumanPlayer` ou `AIPlayer`, où des méthodes comme `play()` sont implémentées différemment selon que le joueur est contrôlé par un humain ou par une IA.

Stratégies de Développement

Abstraction et Héritage : L'utilisation de l'abstraction avec `Player` permet une conception flexible et extensible du système de joueur, facilitant l'ajout de nouveaux types de joueurs sans perturber le code existant.

Encapsulation : En encapsulant le nom, le score, et l'état du jeu dans une classe unique, le code est plus organisé, plus sécurisé et plus facile à maintenir. Les modifications apportées à la gestion des joueurs peuvent être localisées à la classe `Player`, réduisant ainsi le risque d'erreurs.

Cohésion et Couplage : La classe maintient une forte cohésion en regroupant toutes les propriétés et comportements relatifs au joueur, tout en minimisant le couplage avec d'autres parties du système, ce qui renforce la modularité du code.

3.3.2 Classe `HumanPlayer`

La classe `HumanPlayer` est une extension de la classe abstraite `Player`, conçue spécifiquement pour gérer les interactions et le comportement des joueurs humains dans le jeu Wordle multijoueur. Elle hérite des propriétés de base de `Player` telles que le nom, le score, et le statut de participation à une partie, et implémente les spécificités liées à un joueur humain, notamment la façon dont il joue.

Structure et Conception

Héritage : `HumanPlayer` étend la classe `Player`, ce qui signifie qu'elle reprend tous les attributs et méthodes de `Player`, tout en ajoutant des spécificités liées aux interactions humaines.

Constructeur : Le constructeur de `HumanPlayer` accepte un argument `name`, qui est transmis au constructeur de la classe parente `Player` pour initialiser le nom du joueur. Cela assure que toutes les propriétés de base du joueur sont correctement initialisées dès la création de l'objet.

Fonctionnalités et Méthodes

Méthode `play()` : Cette méthode est une implémentation concrète de la méthode abstraite `play()` définie dans `Player`. Dans `HumanPlayer`, `play()` contient la logique permettant au joueur humain d'interagir avec le jeu, par exemple, choisir un mot ou répondre à une énigme.

Utilisation de la Classe dans le Projet

Interaction Utilisateur : `HumanPlayer` est essentiel pour gérer les actions des joueurs humains, depuis la saisie de leur nom jusqu'à leur participation active dans le jeu. Elle peut être utilisée pour interfacer directement avec les composants de l'interface utilisateur qui recueillent les entrées des joueurs, traitent les réponses et mettent à jour l'état du jeu en conséquence.

Extensibilité et Maintenance : Séparer les comportements des joueurs humains dans une classe distincte permet une meilleure organisation du code et facilite les modifications ou l'extension des fonctionnalités spécifiques aux joueurs humains sans affecter la logique des autres types de joueurs (comme les IA).

Stratégies de Développement

Principe de Responsabilité Unique : `HumanPlayer` respecte ce principe en se concentrant exclusivement sur les aspects des joueurs humains. Cela rend la classe plus facile à comprendre, tester et maintenir.

Couplage Faible : Bien que dépendante de la classe `Player` pour certaines fonctionnalités de base, `HumanPlayer` est conçue pour minimiser le couplage avec d'autres parties du système, ce qui favorise une plus grande modularité du code et réduit l'impact des modifications.

Cohésion Élevée : La classe présente une forte cohésion en encapsulant toutes les fonctionnalités pertinentes pour les joueurs humains, rendant le code plus robuste et moins sujet aux erreurs lors des modifications ou extensions futures.

3.3.3 Classe `GameSession`

La classe `GameSession` joue un rôle crucial dans la gestion des sessions de jeu multijoueur pour le jeu Wordle. Elle encapsule la logique de contrôle des sessions de jeu, y compris la gestion des joueurs (représentés par des `ClientHandler`), le démarrage et la fin des jeux, ainsi que la communication inter-joueurs.

Structure et Conception

Attributs :

- `clientHandlers` : Une liste qui stocke les références aux gestionnaires de clients (`ClientHandler`) qui participent à la session de jeu.
- `gameConfig` : Une instance de `GameConfig` qui contient la configuration spécifique du jeu, telle que le nombre maximum de joueurs et le niveau de difficulté.
- `game` : Une instance de `WordleGame` qui représente le jeu actuel.
- `gameInProgress` : Un `AtomicBoolean` qui indique si une partie est actuellement en cours, garantissant ainsi une gestion thread-safe de l'état du jeu.

Méthodes :

- **Constructeur** : Initialise la session avec des `ClientHandler` et une configuration de jeu.
- `isFull()` : Vérifie si le nombre maximum de joueurs dans la session a été atteint.
- `closeSession()` : Termine la session en notifiant les joueurs et en fermant toutes les connexions.
- `removeClient()` : Supprime un client de la session.
- `relayAction()` : Relais les actions ou messages à tous les clients sauf à l'expéditeur.
- `startGame()` : Démarre une nouvelle partie si aucune n'est en cours.
- `processPlayerGuess()` : Traite les tentatives de devinette des joueurs.
- `endGame()` : Termine le jeu en cours et notifie les joueurs du mot correct.
- `notifyClients()` : Envoie un message à tous les clients de la session.
- `addClient()` : Ajoute un client à la session si de l'espace est disponible.

Utilisation de la Classe dans le Projet

Gestion des Joueurs : `GameSession` permet de gérer dynamiquement les joueurs dans une session de jeu, en ajoutant ou supprimant des clients en fonction des connexions et déconnexions.

Contrôle de Session : La classe contrôle le flux du jeu en démarrant et en terminant des sessions, garantissant que les règles du jeu sont respectées et que les sessions ne débordent pas au-delà des limites configurées.

Communication : Facilite la communication entre les joueurs via la méthode `relayAction`, qui est essentielle pour une expérience multijoueur interactive.

Stratégies de Développement

Cohésion et Couplage : La classe montre une forte cohésion en encapsulant toutes les fonctionnalités nécessaires à la gestion d'une session de jeu. Elle est faiblement couplée avec les autres parties du système, interagissant principalement via des interfaces bien définies.

Gestion des Concurrency : Utilisation d'`AtomicBoolean` pour la gestion de l'état de la session afin d'éviter les conditions de course dans des environnements multithread.

Extensibilité : La conception permet d'ajouter facilement de nouvelles fonctionnalités, telles que de nouveaux types de jeux ou des configurations de jeu, sans perturber la logique de gestion de session existante.

3.3.4 Classe `GameInfo`

La classe `GameInfo` sert de modèle pour encapsuler toutes les informations pertinentes d'une session de jeu spécifique dans le contexte du jeu Wordle multijoueur. Elle stocke des détails essentiels qui régissent les règles et le fonctionnement de la partie, tels que le nombre de joueurs, la durée du jeu, et le niveau de difficulté.

Structure et Conception

Attributs :

- `serverPort` : Le port sur lequel le serveur du jeu est ouvert.
- `numberOfPlayers` : Le nombre actuel de joueurs connectés à la session.
- `difficulty` : Le niveau de difficulté du jeu, qui peut influencer la complexité des mots à deviner.
- `gameDuration` : La durée totale de la partie.
- `numberOfWords` : Le nombre de mots que les joueurs doivent deviner pendant la partie.
- `maxPlayers` : Le nombre maximum de joueurs qui peuvent rejoindre la session.
- `hostName` : Le nom de l'hôte de la session, généralement le créateur de la partie.

Constructeurs :

- `GameInfo(params...)` : Initialise une instance avec les paramètres spécifiés.
- `GameInfo()` : Constructeur par défaut qui initialise les attributs avec des valeurs par défaut.

Méthodes :

- `update()` : Met à jour les paramètres de la partie en fonction des entrées de l'utilisateur ou des réglages du jeu.
- `fromString()` : Convertit une chaîne de caractères en une instance de `GameInfo`, facilitant la sérialisation et la désérialisation des informations du jeu.

- `toString()` : Convertit l'objet `GameInfo` en chaîne de caractères, idéal pour le stockage ou la transmission sur le réseau.

Utilisation de la Classe dans le Projet

Transmission d'Informations : La classe est utilisée pour transmettre les paramètres du jeu entre le serveur et les clients, garantissant que tous les joueurs ont une vue cohérente de l'état de la partie.

Gestion Dynamique : Permet la modification dynamique des paramètres de jeu, tels que la difficulté et la durée, à travers des interfaces utilisateur qui interagissent avec cette classe.

Facilitation de la Synchronisation : En centralisant les informations du jeu dans une seule classe, `GameInfo` facilite la synchronisation entre différents composants du système, réduisant ainsi les risques d'incohérences.

Stratégies de Développement

Modularité : La classe `GameInfo` illustre un bon exemple de modularité et d'encapsulation, où toutes les informations nécessaires sont regroupées de manière logique.

Flexibilité et Extensibilité : Le design permet des modifications faciles des règles du jeu ou des ajouts de nouvelles fonctionnalités sans impacter les autres composants.

Cohérence et Intégrité des Données : Des méthodes telles que `update()` et les constructeurs bien définis garantissent que l'objet `GameInfo` reste cohérent et valide tout au long de son cycle de vie.

3.3.5 Classe `GameConfig`

La classe `GameConfig` est conçue pour encapsuler toutes les configurations nécessaires à la gestion d'une session de jeu Wordle multijoueur. Elle stocke des paramètres cruciaux qui influencent directement le déroulement du jeu, comme le nombre de joueurs, la durée de la partie, et le niveau de difficulté. Ce modèle permet une gestion centralisée et une modification aisée des paramètres du jeu, ce qui est essentiel pour l'adaptabilité et la maintenabilité du système.

Structure et Conception

Attributs Principaux :

- `maxPlayers` : Le nombre maximum de joueurs que la session de jeu peut accueillir.
- `gameDuration` : Temps total alloué pour la session de jeu.
- `difficultyLevel` : Niveau de difficulté du jeu, impactant la sélection des mots.
- `maxWordFindTime` : Le temps maximal accordé pour trouver un mot.
- `wordLength` : La longueur des mots à deviner.
- `numberOfWords` : Le nombre de mots à deviner durant la partie.
- `defaultDifficultyLevel` et `defaultWordLength` : Valeurs par défaut pour le niveau de difficulté et la longueur des mots, utilisées pour réinitialiser ou configurer des sessions.

Constructeurs :

- `GameConfig(int maxPlayers, int gameDuration, int difficultyLevel, int wordLength)` : Initialise une configuration de jeu avec les paramètres spécifiés.
- `GameConfig(int maxPlayers, int gameDuration, int difficultyLevel)` : Une surcharge du constructeur qui utilise une longueur de mot par défaut, simplifiant la configuration pour des cas où la longueur du mot n'est pas une variable critique.

Méthodes de Gestion :

- **getters et setters** : Fonctions pour obtenir et définir les valeurs des différents paramètres. Ces méthodes permettent une modification dynamique des paramètres du jeu en réponse à des actions de l'utilisateur ou à des besoins du système.

Utilisation de la Classe dans le Projet

Flexibilité de Configuration : Permet aux administrateurs du jeu ou aux développeurs d'ajuster facilement les paramètres du jeu en fonction des retours des utilisateurs ou de nouvelles exigences, sans nécessiter de modifications profondes du code.

Facilitation de la Synchronisation : Centralise la gestion des configurations de jeu, facilitant la synchronisation entre le serveur et les clients. Cela assure que tous les joueurs ont une vision cohérente des règles du jeu.

Cohérence et Validation : Les méthodes de la classe aident à maintenir la cohérence des données de configuration et à valider les entrées pour prévenir les erreurs d'exécution qui pourraient survenir à cause de configurations incorrectes.

Stratégies de Développement

Encapsulation et Abstraction : Encapsule toutes les informations de configuration dans un seul objet, ce qui réduit la complexité du code ailleurs dans l'application et améliore l'abstraction.

Sécurité et Validité des Données : Les setters offrent un moyen de valider les données avant de les appliquer, ce qui protège l'application contre des états incohérents ou invalides.

Adaptabilité et Maintenance : La conception de la classe permet des modifications futures avec un minimum d'effort. Par exemple, ajouter un nouveau paramètre de configuration nécessite de modifier uniquement `GameConfig` et les parties du code qui utilisent ce nouveau paramètre.

3.3.6 Classe Bot

La classe `Bot` représente une implémentation d'un joueur artificiel (bot) pour le jeu de Wordle en ligne. Ce joueur non humain est programmé pour interagir avec le jeu de manière automatisée, simulant ainsi la présence et les actions d'un joueur humain.

Structure et Composition**Attributs statiques :**

- `BOT_NAMES` : Un tableau de chaînes constantes stockant une série de noms prédéfinis pour les bots. Ces noms, inspirés par les lettres de l'alphabet grec, sont utilisés pour générer aléatoirement un identifiant unique pour chaque instance de bot.

Constructeur : Le constructeur de la classe **Bot** sélectionne un nom aléatoire du tableau **BOT_NAMES** et initialise l'instance avec ce nom en utilisant le constructeur de la classe parent **Player**. Cette approche garantit que chaque bot a un nom distinctif tout en conservant une structure simple et efficace.

Méthodes de la Classe

chooseWord() : Cette méthode est destinée à être développée pour permettre au bot de sélectionner un mot à partir d'une liste donnée. La logique spécifique n'étant pas implémentée, elle sert de point de développement futur où l'algorithme de sélection de mots sera codifié.

readFeedbacks() et readHints() : Prévues pour permettre au bot de traiter les réactions ou indices fournis par le jeu, après une tentative de devinette de mot. Ces méthodes servent de placeholder pour de futures implémentations, permettant au bot d'ajuster ses choix en fonction des indices reçus.

play() : Méthode abstraite héritée de la classe **Player**, qui doit être implémentée spécifiquement pour définir comment le bot joue son tour. Cette méthode intégrerait les interactions entre choisir un mot, recevoir des feedbacks, et lire des indices pour faire des choix informés lors des tours subséquents.

Concepts de Programmation Utilisés

Héritage : En héritant de la classe **Player**, **Bot** bénéficie de toutes les propriétés et méthodes de **Player**, tout en permettant une extension spécifique pour les fonctionnalités propres aux bots.

Encapsulation : Les attributs spécifiques du bot sont privés et manipulés uniquement à travers des méthodes publiques, préservant l'intégrité des données et cachant les détails de l'implémentation interne.

Polymorphisme : **Bot** implémente la méthode **play()** qui est définie de manière abstraite dans **Player**. Cela permet d'utiliser des références de type **Player** pour manipuler des instances de **Bot**, rendant le code plus général et flexible.

3.3.7 Interface ServerMessageListener

L'interface **ServerMessageListener** joue un rôle fondamental dans la gestion de la communication entre le serveur et ses clients dans un environnement de jeu multijoueur, spécifiquement pour le projet qui réinvente le jeu de Wordle en ligne.

Objectif et Rôle L'interface **ServerMessageListener** est conçue pour définir un contrat pour les objets qui souhaitent écouter les messages envoyés par le serveur. Cette abstraction est cruciale pour le déroulement synchronisé et interactif du jeu, permettant une communication fluide et dynamique entre le serveur et ses clients.

Méthode Définie

- **onServerMessage(String message) :** Cette méthode doit être implémentée par toutes les classes qui s'inscrivent comme écouteurs auprès du serveur. L'argument **message** transportera les données nécessaires ou les commandes du serveur, ce qui peut inclure les actions des joueurs, les mises à jour de l'état du jeu, ou des alertes spécifiques au système.

Implications de la Conception

Dé-couplage : En utilisant une interface pour définir comment les messages du serveur sont reçus, le système est conçu de manière à réduire les dépendances entre le serveur et ses clients. Cela permet d'implémenter divers types de clients ou de gestionnaires de clients qui peuvent interagir avec le serveur de manière flexible.

Flexibilité et Extensibilité : Avec cette interface, le système peut facilement s'étendre pour supporter de nouveaux types de messages ou des comportements différents lors de la réception de messages sans perturber les composants existants. Par exemple, si une nouvelle fonctionnalité du jeu nécessite une nouvelle forme de communication, de nouveaux listeners peuvent être créés et intégrés sans affecter les anciens.

Maintenance et Évolutivité : Le fait d'encapsuler la logique de traitement des messages du serveur derrière une interface simplifie les modifications futures et la maintenance du code. Les membres du groupe peuvent mettre à jour ou modifier la gestion des messages en implémentant simplement cette interface dans une nouvelle classe sans avoir besoin de modifier le serveur ou d'autres parties du code client.

Testabilité : L'utilisation de l'interface facilite également le test du système. Les tests unitaires peuvent être écrits en mockant cette interface, permettant de simuler divers scénarios de communication entre le serveur et le client pour tester la robustesse et la réactivité du système de jeu.

3.4 Conception des graphiques des statistiques (Samy)

3.4.1 Les fichiers de score

Afin de pouvoir tester dans de bonnes conditions et étant donné que la gestion de sauvegarde du score n'a pas encore été implémenté, j'ai créé un nouveau dossier score, dans lequel j'ai mis 2 fichiers d'exemples, "score.txt" et "scoreMulti.txt". Ces fichiers servent d'exemple pour les futurs fichiers de scores. On a donc dû définir le format qu'aura la sauvegarde du score, même si elle ne sera pas forcément définitive, c'est pour l'instant l'idée qu'on en a. Voici le format du score : pseudo:temps+lettres+essais=total . Les variables temps, lettres et essais correspondent en fait au nombre de points gagnés dans ces catégories-là. La suite du code se base donc temporairement sur ces fichiers d'exemples.

3.4.2 Création de la classe Graphique

Le principe de la classe Graphique est très simple. Elle ne contient que 4 fonctions, une pour lire les fichiers de scores, et les 3 autres représentent les 3 graphiques faisables. Chaque fonction de graphiques prendra en argument le chemin du fichier de score concerné, et fera appel à la fonction de lecture de fichier pour en analyser les scores. Pour le moment, les graphiques sont générés dans des modales créées par l'appui des boutons correspondant. Elle contient quatre fonctions suivantes :

- **readLinesFromFile** : Cette fonction très simple permet de lire un fichier texte et d'en extraire, ligne par ligne, les informations nécessaires. C'est cette fonction qui va permettre aux autres de récupérer les données de scores, individuellement pour chaque joueur.
- **showGraph** : Cette fonction génère un graphique circulaire représentant les données de score pour un joueur donné. Elle affiche les proportions relatives des différentes catégories de scores (temps, lettres, essais) sous forme de secteurs dans le graphique circulaire. Chaque secteur est étiqueté avec le nombre de points correspondant à cette catégorie.
- **showMultiGraph** : Cette fonction crée un graphique à barres affichant les scores de plusieurs joueurs. Elle représente les totaux de points de chaque joueur sous forme de barres, où chaque joueur est représenté par une barre. Chaque joueur est identifié par son nom et la hauteur de la barre correspond au total de ses points.
- **showMultiDetailGraph** : Cette fonction génère un graphique à barres empilées détaillé affichant les scores des joueurs dans les différentes catégories (temps, lettres, essais). Elle représente les scores de chaque joueur dans chaque catégorie à l'aide de barres empilées. Chaque joueur est identifié par son nom et chaque catégorie est représentée par une couleur différente dans la barre empilée.

3.4.3 Modification de la classe WordleView

La classe WordleView a seulement été modifiée temporairement afin d'intégrer les graphiques pour des tests. En effet, étant donné que l'application n'est pas encore finie, le contexte n'est pas encore suffisant pour assurer la véritable mise en place de ces graphiques. Je me suis donc seulement contenté de rajouter trois boutons dans ce fichier, un pour chaque fonction/graphique, à côté des boutons d'actions contextuelles(indice, aide) afin de pouvoir librement les tester.

3.5 Conception et mécanisme des agents (Louis)

3.5.1 Classe : Agent

La classe Agent joue un rôle crucial dans la proposition de mots pour le joueur, simulant ainsi un adversaire virtuel dans le jeu Wordle. Cette section explore les principaux mécanismes de l'Agent, ainsi que ses différentes stratégies de jeu.

Méthode : agentDifficulte

La méthode agentDifficulte est le point d'entrée de l'Agent. Elle prend en compte plusieurs paramètres, notamment la taille de la grille de jeu, le niveau de difficulté sélectionné par le joueur et le mot cible à deviner. En fonction du niveau de difficulté choisi, cette méthode détermine la stratégie à adopter par l'Agent.

Méthode : agentPlayEasy

La méthode agentPlayEasy représente la stratégie d'agent la plus simple, adaptée au niveau de difficulté "Facile". L'Agent génère aléatoirement des mots de la même longueur que le mot cible. Ces mots sont ajoutés à une liste, représentant les propositions de l'Agent.

Méthode : agentPlayMedium

La méthode agentPlayMedium implémente une stratégie plus sophistiquée pour le niveau de difficulté "Moyen". L'Agent commence par choisir un mot aléatoire. Ensuite, pour chaque position dans le mot, il tente de choisir une lettre qui correspond à celle du mot cible, même si elle n'est pas nécessairement à la bonne place. Cette stratégie augmente les chances de trouver le mot cible plus rapidement.

Méthode : agentPlayHard

La méthode agentPlayHard représente la stratégie la plus complexe, réservée au niveau de difficulté "Difficile". Dans cette stratégie, l'Agent prend en compte à la fois la lettre correcte et sa position dans le mot cible. Il tente ainsi de deviner les lettres correctes à leurs emplacements respectifs, ce qui nécessite une analyse plus approfondie et une exploration plus intelligente de l'espace de recherche des mots.

3.5.2 Classe : WordleView

Méthode : showAgentDifficultySelection

La méthode showAgentDifficultySelection est responsable de l'affichage de la fenêtre de sélection de la difficulté de l'Agent. Elle permet aux joueurs de choisir parmi les niveaux de difficulté "Facile", "Moyen" ou "Difficile". Une fois la difficulté sélectionnée, les joueurs peuvent valider leur choix, déclenchant ainsi le début du jeu avec l'Agent.

Méthode : startAgentGame

La méthode startAgentGame initialise une nouvelle partie avec un agent. Elle affiche le jeu Wordle et démarre le décompte du temps. Cette méthode est appelée après la sélection de la difficulté de l'Agent par le joueur.

Méthode : agentPlayTurn

La méthode agentPlayTurn représente un tour de jeu de l'Agent. Elle récupère la difficulté sélectionnée par le joueur et déclenche la méthode agentDifficulte de la classe Agent pour obtenir une liste de mots proposés par l'Agent. Ces mots sont ensuite affichés dans la console, simulant ainsi les actions de l'Agent.

4 Développement Final

4.1 Animations (Anthony)

4.1.1 Animations

Pour enrichir l'expérience utilisateur du jeu Wordle multijoueur, j'ai créé des classes dédiées aux animations. Voici un aperçu des classes principales et de leur fonctionnement :

4.1.2 Classe **AnimationManager**

La classe **AnimationManager** est le gestionnaire central des animations dans le jeu. Elle coordonne et contrôle les différentes animations qui se déroulent pendant la partie.

Attributs :

- **animations** : Liste des animations en cours.
- **canvas** : La surface sur laquelle les animations sont dessinées.

Méthodes :

- **addAnimation(Animation animation)** : Ajoute une animation à la liste des animations à exécuter.
- **removeAnimation(Animation animation)** : Retire une animation de la liste une fois terminée.
- **update(long elapsedTime)** : Met à jour toutes les animations en cours.
- **draw(Graphics g)** : Dessine toutes les animations sur le canvas.

Utilisation :

- **Gestion centralisée** : Le **AnimationManager** permet de centraliser la gestion des animations, assurant que toutes les animations sont synchronisées et mises à jour correctement.
- **Extensibilité** : De nouvelles animations peuvent être facilement ajoutées et gérées via cette classe.

4.1.3 Classe **Animation**

La classe **Animation** est une classe abstraite qui définit le comportement de base de toutes les animations. Chaque animation spécifique héritera de cette classe et implémentera ses propres détails.

Attributs :

- **duration** : Durée de l'animation.
- **startTime** : Moment où l'animation a commencé.

Méthodes :

- **start()** : Démarre l'animation en enregistrant le temps de début.
- **isFinished(long currentTime)** : Vérifie si l'animation est terminée.
- **update(long elapsedTime)** : Met à jour l'état de l'animation.
- **draw(Graphics g)** : Dessine l'animation sur le canvas.

Utilisation :

- **Structure commune** : En fournissant une structure commune, la classe **Animation** permet de standardiser la création et la gestion des animations.
- **Facilité d'extension** : De nouvelles animations peuvent être créées en étendant cette classe et en implémentant les méthodes abstraites.

4.1.4 Classe FadeAnimation

La classe **FadeAnimation** gère les animations de fondu, où un élément apparaît ou disparaît progressivement.

Attributs :

- **opacity** : Niveau d'opacité actuel.
- **step** : Incrément de changement de l'opacité par mise à jour.

Méthodes :

- **update(long elapsedTime)** : Met à jour l'opacité en fonction du temps écoulé.
- **draw(Graphics g)** : Dessine l'élément avec l'opacité actuelle.

Utilisation :

- **Transitions douces** : Utilisée pour créer des transitions douces, comme l'apparition ou la disparition de messages ou d'éléments du jeu.
- **Effets visuels** : Améliore l'aspect visuel du jeu en ajoutant des effets de fondu.

4.1.5 Classe BounceAnimation

La classe **BounceAnimation** gère les animations de rebond, où un élément se déplace de manière élastique.

Attributs :

- **position** : Position actuelle de l'élément.
- **velocity** : Vitesse actuelle du rebond.
- **gravity** : Force de gravité appliquée au rebond.

Méthodes :

- **update(long elapsedTime)** : Met à jour la position et la vitesse en fonction du temps écoulé et de la gravité.
- **draw(Graphics g)** : Dessine l'élément à sa position actuelle.

Utilisation :

- **Effets dynamiques** : Utilisée pour créer des effets dynamiques, tels que le rebond d'une lettre correctement devinée.
- **Interactivité** : Ajoute un niveau d'interactivité visuelle en rendant les éléments du jeu plus réactifs et animés.

4.1.6 Classe ShakeAnimation

La classe **ShakeAnimation** gère les animations de secousse, où un élément tremble pour attirer l'attention.

Attributs :

- **amplitude** : Amplitude du tremblement.
- **frequency** : Fréquence des secousses.
- **initialPosition** : Position initiale de l'élément.

Méthodes :

- **update(long elapsedTime)** : Met à jour la position de l'élément en appliquant un mouvement de secousse.
- **draw(Graphics g)** : Dessine l'élément à sa position actuelle secouée.

Utilisation :

- **Alertes visuelles** : Utilisée pour attirer l'attention sur des erreurs ou des notifications importantes, comme une lettre incorrecte.
- **Renforcement de feedback** : Améliore le feedback visuel en fournissant des indications claires et immédiates des actions du joueur.

Ces classes permettent de gérer et de créer des animations variées et interactives, améliorant ainsi l'expérience utilisateur du jeu Wordle multijoueur.

4.2 Gestion des Scores Multijoueur (Anthony)

Pour offrir une expérience compétitive et mémorable aux joueurs de notre jeu Wordle multijoueur, j'ai développé des fonctionnalités dédiées à la gestion et à la sauvegarde des scores des joueurs. Voici un aperçu des classes principales et de leur fonctionnement :

4.2.1 Classe ScoreManager

La classe **ScoreManager** est le gestionnaire central des scores dans le jeu. Elle est responsable de la gestion des scores des joueurs, de leur sauvegarde dans des fichiers et de la récupération de ces données.

Attributs :

- `scores` : Un `HashMap` associant les noms des joueurs à leurs scores.
- `filePath` : Chemin du fichier où les scores sont sauvegardés.

Méthodes :

- `addScore(String playerName, int score)` : Ajoute ou met à jour le score d'un joueur.
- `getScore(String playerName)` : Récupère le score d'un joueur spécifique.
- `saveScores()` : Sauvegarde les scores dans un fichier.
- `loadScores()` : Charge les scores depuis un fichier.
- `resetScores()` : Réinitialise tous les scores, par exemple, pour une nouvelle saison de jeu.

Utilisation :

- **Gestion centralisée** : Le **ScoreManager** permet de centraliser la gestion des scores, garantissant que toutes les mises à jour et récupérations de scores sont effectuées de manière cohérente.
- **Persistant des données** : Sauvegarde et charge les scores des fichiers, assurant que les données sont conservées entre les sessions de jeu.

4.2.2 Classe Score

La classe **Score** est une classe modèle qui représente le score d'un joueur. Elle encapsule les informations essentielles liées au score.

Attributs :

- `playerName` : Nom du joueur.
- `scoreValue` : Valeur actuelle du score.

Méthodes :

- `getPlayerName()` : Renvoie le nom du joueur.
- `getScoreValue()` : Renvoie la valeur actuelle du score.
- `setScoreValue(int scoreValue)` : Met à jour la valeur du score.

Utilisation :

- **Encapsulation des données** : La classe **Score** encapsule les données liées au score, rendant le code plus organisé et plus facile à maintenir.
- **Interchangeabilité** : Permet une manipulation aisée des scores à travers différentes parties du système.

4.2.3 Classe `FileHandler`

La classe **`FileHandler`** est responsable des opérations de lecture et d'écriture des fichiers où les scores sont sauvegardés.

Attributs :

- `filePath` : Chemin du fichier à manipuler.

Méthodes :

- `writeToFile(Map<String, Integer> data)` : Écrit les données de scores dans le fichier spécifié.
- `readFromFile()` : Lit les données de scores depuis le fichier et les retourne sous forme de `HashMap`.
- `fileExists()` : Vérifie si le fichier de scores existe.

Utilisation :

- **Gestion de fichiers** : Centralise les opérations de lecture et d'écriture de fichiers, simplifiant le code de gestion des scores.
- **Vérification de l'existence** : Assure que les opérations de lecture et d'écriture sont effectuées de manière sécurisée.

4.2.4 Classe `ScoreComparator`

La classe **`ScoreComparator`** permet de comparer les scores des joueurs pour faciliter le classement et le tri.

Méthodes :

- `compare(Score score1, Score score2)` : Compare deux objets **`Score`** en fonction de leurs valeurs, permettant de trier les joueurs du meilleur au moins bon score.

Utilisation :

- **Classement des joueurs** : Utilisée pour trier et afficher les scores des joueurs dans un ordre décroissant, facilitant la création de tableaux de classement.
- **Flexibilité** : Permet d'intégrer facilement des méthodes de tri personnalisées si nécessaire.

Ces classes permettent de gérer et de sauvegarder les scores des joueurs de manière efficace, assurant une expérience de jeu compétitive et durable pour le jeu Wordle multijoueur.

4.3 Gestion des joueurs - retrait (Youssef)

Pour garantir une expérience de jeu fluide et équilibrée dans notre jeu multijoueur Wordle, il est essentiel de pouvoir gérer les joueurs efficacement, y compris la capacité de retirer des joueurs du lobby. Voici une description détaillée des fonctionnalités et des classes impliquées dans cette gestion.

4.3.1 Description

Le retrait des joueurs est une fonctionnalité critique qui permet à l'hôte du jeu de maintenir l'ordre et la qualité de la session de jeu en retirant des joueurs problématiques ou inactifs.

Fonctionnalités

- **Capacité pour l'hôte d'inviter de nouveaux joueurs** : L'hôte peut envoyer des invitations à de nouveaux joueurs pour rejoindre le lobby, assurant que le nombre de participants est suffisant pour commencer une partie.
- **Capacité pour l'hôte de retirer des joueurs** : L'hôte peut retirer des joueurs du lobby si nécessaire, par exemple, en cas de comportement inapproprié ou d'inactivité prolongée.

4.3.2 Classe Lobby

La classe **Lobby** joue un rôle central dans la gestion des joueurs, y compris leur ajout et leur retrait.

Méthodes Importantes

- **addClient(ClientHandler clientHandler)** : Ajoute un nouveau client au lobby.
- **removeClient(ClientHandler clientHandler)** : Retire un client du lobby et met à jour l'état des sessions actives si nécessaire.

Utilisation

- La méthode **addClient** est utilisée pour ajouter un nouveau joueur au lobby. Si le lobby est plein, le client reçoit un message d'erreur.
- La méthode **removeClient** est utilisée pour retirer un joueur du lobby et mettre à jour l'état des sessions actives. Cela garantit une gestion appropriée des ressources et maintient la stabilité du jeu.

4.3.3 Classe Server

La classe **Server** interagit également avec la classe **Lobby** pour gérer les joueurs.

Méthodes Importantes

- **removeClient(ClientHandler clientHandler)** : Supprime un client de la liste des clients actifs et du lobby.

Utilisation

- La méthode **removeClient** dans la classe **Server** garantit que le client est retiré correctement du lobby et que toutes les ressources associées sont libérées.

4.4 Démarrage de la partie (Youssef)

Le démarrage de la partie est une fonctionnalité essentielle qui permet à l'hôte de lancer le jeu une fois que tous les joueurs sont prêts et que les paramètres du jeu sont configurés. Voici une description détaillée des fonctionnalités et des classes impliquées dans cette opération.

4.4.1 Description

Le démarrage de la partie assure que tous les joueurs sont prêts et que toutes les conditions requises sont remplies avant de lancer le jeu.

Fonctionnalités

- **Bouton "READY" pour les joueurs non-hôtes** : Les joueurs non-hôtes peuvent indiquer qu'ils sont prêts en appuyant sur un bouton "READY".
- **Bouton ou commande pour démarrer la partie** : L'hôte peut lancer la partie en utilisant un bouton.
- **Vérification que tous les joueurs sont prêts** : Le jeu ne commence que lorsque tous les joueurs ont indiqué qu'ils sont prêts.

4.4.2 Classe Lobby

La classe `Lobby` gère le démarrage de la partie en vérifiant l'état de préparation des joueurs et en lançant la partie lorsque toutes les conditions sont remplies.

Méthodes Importantes

- `startGameForAllSessions()` : Démarre toutes les sessions de jeu actives une fois que les conditions requises sont remplies.
- `canStartGame()` : Vérifie si le nombre de joueurs en attente est suffisant pour commencer une nouvelle session de jeu.

Utilisation

- La méthode `startGameForAllSessions` est utilisée pour lancer toutes les sessions de jeu actives lorsque tous les joueurs sont prêts.
- La méthode `canStartGame` vérifie si les conditions pour commencer une nouvelle session de jeu sont remplies.

4.4.3 Classe GameSession

La classe `GameSession` est responsable de la gestion individuelle des sessions de jeu et de leur démarrage.

Méthodes Importantes

- `startGame()` : Démarre la session de jeu pour tous les joueurs de cette session.

Utilisation

- La méthode `startGame` initialise et lance la session de jeu, informant tous les joueurs que le jeu a commencé.

4.4.4 Classe ClientHandler

La classe `ClientHandler` gère les interactions avec les clients et relaie les commandes de démarrage de jeu.

Méthodes Importantes

- `processCommand(String message)` : Interprète les commandes reçues du client, y compris la commande de démarrage de la partie.

Utilisation

- La méthode `processCommand` interprète les commandes comme "START GAME" pour déclencher le démarrage de la partie via le `Lobby`.

4.5 Progression du jeu (Youssef)

4.5.1 Gestion de la déconnexion d'un joueur

La gestion de la déconnexion d'un joueur est une fonctionnalité essentielle pour assurer la fluidité du jeu, même en cas de déconnexion involontaire ou volontaire d'un joueur. Cette fonctionnalité détecte et gère les déconnexions pour maintenir l'état du jeu cohérent et éviter les interruptions.

Fonctionnalités Clés

- **Détection des déconnexions** : Utilisation de mécanismes de détection automatique pour identifier quand un joueur se déconnecte.
- **Maintien de l'état du jeu** : Préservation de l'état actuel du jeu afin de minimiser les perturbations pour les autres joueurs.
- **Notification des joueurs** : Informer les autres joueurs de la déconnexion d'un joueur pour assurer une transparence.
- **Reconnexion des joueurs** : Permettre aux joueurs de se reconnecter sans perdre leur progression.

Méthodes Importantes

- `detectDisconnection(ClientHandler clientHandler)` : Méthode pour détecter la déconnexion d'un client.
- `maintainGameState()` : Assurer que l'état du jeu reste cohérent malgré les déconnexions.
- `notifyPlayers(String message)` : Envoyer une notification à tous les joueurs actifs.

Détails de Conception

- **Gestion des Threads** : Chaque `ClientHandler` surveille les connexions pour détecter toute interruption de communication.
- **Mécanismes de Reconnexion** : Permettre la reconnexion rapide des joueurs pour qu'ils puissent reprendre leur partie.

4.5.2 Possibilité de rejoindre une partie après déconnexion

Cette fonctionnalité permet aux joueurs de se reconnecter et de rejoindre une partie en cours après une déconnexion. Cela garantit une expérience de jeu continue et sans interruption pour les joueurs qui pourraient être temporairement déconnectés.

Fonctionnalités Clés

- **Sauvegarde de l'état du joueur** : Conserver l'état du joueur pour permettre une reconnexion fluide.
- **Interface de reconnexion** : Fournir une interface utilisateur pour faciliter la reconnexion.
- **Validation des informations de connexion** : Vérifier les informations du joueur lors de la reconnexion.

Méthodes Importantes

- `savePlayerState(ClientHandler clientHandler)` : Sauvegarder l'état actuel du joueur.
- `reconnectPlayer(String playerName, Socket socket)` : Gérer la reconnexion d'un joueur en utilisant les informations sauvegardées.
- `restoreGameState(ClientHandler clientHandler)` : Restaurer l'état du jeu pour un joueur reconnecté.

Détails de Conception

- **Structures de Données** : Utilisation de structures de données pour sauvegarder et restaurer l'état du joueur.
- **Sécurité et Validation** : Validation des informations de connexion pour assurer la sécurité et l'intégrité du jeu.

4.5.3 Synchronisation des clients

La synchronisation des clients est cruciale pour garantir que tous les joueurs voient le même état du jeu en temps réel. Cette fonctionnalité permet de synchroniser l'état du jeu entre le serveur et tous les clients connectés, assurant l'équité et la cohérence du gameplay.

Fonctionnalités Clés

- **Mécanismes de synchronisation** : Techniques pour synchroniser l'état du jeu entre le serveur et les clients.
- **Mises à jour en temps réel** : Envoi de mises à jour en temps réel aux clients.
- **Gestion des désynchronisations** : Détection et correction des désynchronisations entre les clients et le serveur.

Méthodes Importantes

- `syncGameStateWithClients()` : Méthode pour synchroniser l'état du jeu avec tous les clients.
- `sendRealTimeUpdates()` : Envoi de mises à jour en temps réel aux clients.
- `handleDesynchronization(ClientHandler clientHandler)` : Gestion des désynchronisations détectées.

Détails de Conception

- **Protocole de Communication** : Définition d'un protocole de communication robuste pour la synchronisation des états de jeu.
- **Performance et Scalabilité** : Assurer une synchronisation efficace même avec un grand nombre de clients.

4.5.4 Gestion de la fin du jeu

La gestion de la fin du jeu détermine le moment où le jeu se termine, soit par l'atteinte des objectifs, soit par d'autres critères prédéfinis. Cette fonctionnalité assure une clôture propre du jeu et déclenche les actions de nettoyage nécessaires.

Fonctionnalités Clés

- **Détection de la fin du jeu** : Identifier automatiquement la fin du jeu.
- **Notification des joueurs** : Informer les joueurs de la fin de la partie.
- **Actions de nettoyage** : Exécution des actions nécessaires pour clôturer le jeu proprement.

Méthodes Importantes

- `detectGameEnd()` : Détection des conditions de fin de jeu.
- `notifyGameEnd()` : Notification aux joueurs de la fin de la partie.
- `cleanupGameResources()` : Nettoyage des ressources après la fin du jeu.

Détails de Conception

- **Conditions de Fin de Jeu** : Définition claire des critères de fin de jeu.
- **Gestion des Ressources** : Libération des ressources utilisées par le jeu pour éviter les fuites de mémoire.

4.5.5 Affichage du classement final du jeu

À la fin de chaque partie, le classement des joueurs est affiché pour montrer les scores finaux et les positions des joueurs. Cette fonctionnalité collecte et affiche les résultats du jeu, offrant une vue d'ensemble des performances des joueurs.

Fonctionnalités Clés

- **Collection des scores** : Recueil des scores et des performances des joueurs.
- **Affichage des résultats** : Présentation claire et informative des résultats finaux.

- **Mise en forme des classements** : Formatage des classements pour une présentation visuellement agréable.

Méthodes Importantes

- `collectScores()` : Recueillir les scores des joueurs à la fin du jeu.
- `displayFinalRankings()` : Afficher les classements finaux des joueurs.
- `formatRankings()` : Formater les classements pour une présentation claire.

Détails de Conception

- **Interface Utilisateur** : Conception d'une interface utilisateur pour afficher les classements.
- **Présentation des Données** : Utilisation de graphiques ou de tableaux pour rendre les résultats plus compréhensibles.

4.6 Gestion des Bots dans le mode Multijoueur (Louis)

4.6.1 Implémentation du Bouton de création de bot

Un bouton intitulé « Ajouter un bot » a été ajouté à l'interface utilisateur du lobby multijoueur. Ce bouton est visible uniquement pour l'hôte de la partie. Lors du clic sur le bouton, une `ChoiceDialog` s'ouvre, permettant à l'utilisateur de sélectionner la difficulté du bot à ajouter.

4.6.2 Intégration des Bots aux Parties Multijoueur, méthode *ajoutDunBot*

Les bots ajoutés par l'hôte sont désormais visibles dans les cases de connexion des joueurs. Cela a été accompli en modifiant la méthode `setupPlayerSpacesUI()` pour inclure les noms des bots dans les cases disponibles. Les cases occupées par les bots restent de couleur blanche afin de les différencier des joueurs humains.

4.6.3 Création de la Classe Bot

La classe `Bot` a été créée en s'inspirant de la classe `Agent` utilisée dans le mode solo. La classe `Bot` hérite des caractéristiques de `Joueur` et ajoute des fonctionnalités spécifiques aux bots dans un contexte multijoueur. Lors de la création d'un bot, il lui est attribué un nom aléatoire parmi les lettres de l'alphabet grec (« Alpha », « Beta », « Gamma », « Delta », « Epsilon », etc.) ainsi que sa difficulté récupérée dans le `ChoiceDialog`. Lorsque la méthode `Play()` est appelée, selon la difficulté du bot (facile, moyen ou difficile), la méthode `agentPlayEasy`, `agentPlayMedium`, ou `agentPlayHard` est utilisée, retournant une liste de chaînes de caractères contenant les mots joués par le bot. Cette liste de mots est utilisée plus tard pour les résultats de fin de partie.

Méthode : `agentPlayEasy` La méthode `agentPlayEasy` représente la stratégie d'agent la plus simple, adaptée au niveau de difficulté « Facile ». L'Agent génère aléatoirement des mots de la même longueur que le mot cible. Ces mots sont ajoutés à une liste, représentant les propositions de l'Agent.

Méthode : `agentPlayMedium` La méthode `agentPlayMedium` implémente une stratégie plus sophistiquée pour le niveau de difficulté « Moyen ». L'Agent commence par choisir un mot aléatoire. Ensuite, pour chaque position dans le mot, il tente de choisir une lettre qui correspond à celle du mot cible, même si elle n'est pas nécessairement à la bonne place. Cette stratégie augmente les chances de trouver le mot cible plus rapidement.

Méthode : `agentPlayHard` La méthode `agentPlayHard` représente la stratégie la plus complexe, réservée au niveau de difficulté « Difficile ». Dans cette stratégie, l'Agent prend en compte à la fois la lettre correcte et sa position dans le mot cible. Il tente ainsi de deviner les lettres correctes à leurs emplacements respectifs, ce qui nécessite une analyse plus approfondie et une exploration plus intelligente de l'espace de recherche des mots.

4.7 Gestion des Graphiques de Scores (Samy)

4.7.1 Implémentation des Graphiques de Scores Dynamiques

L'ajout des graphiques dynamiques de scores dans notre projet Wordle Multijoueur vise à fournir une visualisation claire et intuitive des performances des joueurs. Ces graphiques permettent de suivre l'évolution des scores en temps réel, offrant ainsi une expérience de jeu enrichie.

Les graphiques sont intégrés à l'interface utilisateur dans les modes solo et multijoueur. Selon le mode sélectionné, les graphiques affichent les données pertinentes :

- **Mode Solo** : Affiche les scores individuels du joueur au fil du temps, permettant une analyse détaillée de sa progression personnelle.
- **Mode Multijoueur** : Compare les scores de différents joueurs sur une même partie, facilitant une vue d'ensemble des performances relatives.

4.7.2 Boutons de Navigation par Mode de Jeu

Des boutons spécifiques ont été ajoutés pour faciliter la navigation entre les différents modes de jeu et l'affichage des scores :

- **Bouton Afficher Graphique Solo** : Permet de basculer vers l'affichage des scores solo, offrant une visualisation centrée sur la performance individuelle.
- **Bouton Afficher Graphique Multi** : Permet de basculer vers l'affichage des scores multijoueur, permettant de comparer les performances des différents participants.
- **Bouton Rafraîchir Graphique** : Met à jour le graphique en temps réel en récupérant les derniers scores sauvegardés, assurant ainsi que les données affichées sont toujours actuelles.

Ces boutons sont intégrés de manière intuitive à l'interface utilisateur, facilitant une navigation fluide et une expérience utilisateur améliorée.

4.7.3 Nouveaux Types de Graphiques

Comparé à la première version du projet, plusieurs nouveaux types de graphiques ont été introduits pour une analyse plus détaillée des performances des joueurs :

- **Graphique Linéaire** : Suivi de la progression des scores au fil du temps, permettant de visualiser les tendances et les améliorations.
- **Graphique en Secteurs** : Répartition des victoires et défaites dans une série de parties, offrant une vue d'ensemble des performances globales.

Ces nouveaux types de graphiques enrichissent l'analyse des données, permettant aux joueurs de mieux comprendre leurs performances et d'identifier des domaines d'amélioration.

4.7.4 Sauvegarde et Chargement des Scores

Les scores des joueurs sont sauvegardés dans des fichiers spécifiques à chaque mode de jeu. Ces fichiers sont ensuite utilisés pour mettre à jour dynamiquement les graphiques lors du chargement du jeu :

- **Classe GestionScores** : Gère la sauvegarde et le chargement des scores.
- **Méthode : sauvegarderScore(Score score, String mode)** : Sauvegarde le score dans le fichier approprié, garantissant que les données sont stockées de manière structurée.
- **Méthode : chargerScores(String mode)** : Charge les scores à partir du fichier et les renvoie sous forme de liste, permettant une mise à jour dynamique des graphiques.

Cette gestion efficace des scores assure que les données sont toujours à jour et accessibles, permettant une visualisation précise et en temps réel des performances des joueurs.

5 Tests

Cette section détaille les stratégies et méthodologies de tests adoptées pour assurer la fiabilité et la robustesse de notre application Wordle.

Les tests unitaires ont été conçus pour vérifier la fonctionnalité de chaque composant individuel du jeu, tels que le générateur de mots, le système de scoring et l'interface utilisateur. Nous avons utilisé des frameworks comme JUnit pour automatiser ces tests, permettant une validation rapide et efficace de chaque fonctionnalité. Les tests d'intégration, quant à eux, ont été mis en place pour s'assurer que tous les composants du jeu fonctionnent harmonieusement ensemble. Cette approche combinée garantit que chaque élément du jeu fonctionne correctement en isolation, ainsi qu'en conjonction avec les autres composants.

5.1 Classe Server

La classe **Server** sert de point central pour la gestion des connexions clients et la coordination des sessions de jeu. Les tests suivants visent à s'assurer de la fiabilité et de l'efficacité de ces fonctionnalités :

5.1.1 Tests Unitaires

Test du Constructeur :

- Vérifier que le **ServerSocket** est correctement initialisé avec le port fourni.
- Confirmer que le **ExecutorService** est instancié pour gérer les threads clients.
- S'assurer que le **Lobby** est configuré avec les paramètres initiaux appropriés (nombre de joueurs, durée du jeu, niveau de difficulté).

Test de startServer :

- Confirmer que le serveur commence à accepter les connexions seulement si **isServerStarted** est **true**.
- Tester la réaction du serveur face à une exception lors de la création du **ServerSocket** (simulation d'erreur).

Test de configureGame :

- Vérifier que les paramètres du jeu (nombre maximum de joueurs, durée du jeu, niveau de difficulté) sont correctement mis à jour.
- Tester le comportement en cas de commandes de configuration invalides.

Test de updateLobbyStateForAllClients :

- S'assurer que tous les clients reçoivent les mises à jour de l'état du lobby correctement.

Test de removeClient :

- Tester la suppression correcte d'un client de toutes les listes et vérifier l'actualisation de l'état du lobby.

Tests d'Intégration

Interaction avec ClientHandler :

- Tester l'intégration du **Server** avec les instances de **ClientHandler** pour s'assurer que les clients sont correctement ajoutés, gérés et retirés.
- Vérifier la communication entre le serveur et les clients via des messages envoyés et reçus.

Gestion des Sessions de Jeu :

- Tester la création et la gestion des sessions de jeu à partir des requêtes des clients.
- Vérifier que les sessions de jeu commencent et se terminent correctement en réponse aux actions des joueurs.

Reconnexion et Gestion des États :

- Simuler des scénarios de déconnexion et de reconnexion des clients pour tester la robustesse du serveur.
- Vérifier que le serveur maintient un état cohérent et correct lors de ces événements.

5.2 Classe Lobby

La classe `Lobby` gère les clients en attente et les sessions de jeu actives, et offre diverses méthodes pour manipuler ces éléments. Les tests suivants visent à s'assurer de la fiabilité et de l'efficacité de ces fonctionnalités.

5.2.1 Tests unitaires

Test du Constructeur : Vérifier l'initialisation correcte des listes de clients et de sessions, ainsi que la configuration initiale du jeu.

- S'assurer que les listes `waitingClients` et `activeSessions` sont initialisées à vide.
- Confirmer que la configuration du jeu (`gameConfig`) est correctement définie avec les paramètres fournis (nombre maximum de joueurs, durée du jeu, niveau de difficulté).

Test de `addClient` :

- Tester l'ajout d'un client dans un lobby non plein.
- Vérifier le comportement lorsque le lobby est plein (doit refuser l'ajout et notifier le client).
- Confirmer que le premier client ajouté devient l'hôte si aucun hôte n'est présent.

Test de `canStartGame` :

- Confirmer que la méthode retourne `true` lorsque le nombre de joueurs en attente atteint le nombre maximum de joueurs par session.
- Vérifier que la méthode retourne `false` lorsque le nombre de joueurs est insuffisant.

Test de `createNewGameSession` :

- Tester la création d'une session de jeu lorsque le nombre requis de joueurs est atteint.
- Vérifier que les joueurs sont correctement transférés de la liste d'attente à une nouvelle session active.
- S'assurer que la session est correctement configurée avec la configuration du jeu courante.

Test de `removeClient` :

- Tester la suppression d'un client de la liste d'attente.
- Vérifier la suppression d'un client d'une session active et la mise à jour de l'état du lobby.

Test de `updateGameSettings` :

- Vérifier la mise à jour des paramètres de jeu et la notification correcte à tous les clients.

5.2.2 Tests d'intégration

Interaction avec `ClientHandler` :

- Tester les interactions avec les objets `ClientHandler` pour simuler l'envoi de messages et la gestion des clients dans le lobby.
- S'assurer que les modifications de l'état du lobby (par exemple, ajout ou suppression de clients) déclenchent les notifications appropriées aux clients via `ClientHandler`.

Interaction avec `GameSession` :

- Vérifier que les sessions de jeu sont créées avec les bons paramètres et que les clients sont correctement gérés entre les sessions actives et la liste d'attente.
- Tester le démarrage et la fermeture des sessions de jeu à partir du lobby et observer les impacts sur l'état global du jeu.

5.3 Classe `ClientHandler`

La classe `ClientHandler` gère les interactions réseau d'un client connecté, en traitant les messages entrants et en envoyant des réponses. Les tests suivants visent à s'assurer de la fiabilité et de l'efficacité de ces fonctionnalités. :

5.3.1 Tests Unitaires

Test du Constructeur :

- Vérifier que le `Socket` et le `Server` sont correctement initialisés.
- Confirmer que les flux `PrintWriter` et `BufferedReader` sont ouverts.
- Tester la gestion des exceptions lors de l'initialisation des flux.

Test de `sendMessage` :

- Vérifier que le message est correctement formaté et envoyé à travers le flux de sortie.

Test de `closeEverything` :

- S'assurer que toutes les ressources (`Socket`, `PrintWriter`, `BufferedReader`) sont fermées proprement.
- Confirmer la suppression du client du `Server` et du `Lobby`.

Test de run :

- Tester le comportement de la boucle de lecture des messages entrants, y compris le traitement correct des commandes.
- Simuler la fermeture de la connexion et vérifier le nettoyage des ressources.

5.3.2 Tests d'Intégration

Interaction avec `Server` :

- Vérifier l'intégration avec le serveur pour l'ajout et la suppression de clients.
- Tester la communication bidirectionnelle entre le `ClientHandler` et le `Server` pour l'envoi de messages et la réception de commandes.

Gestion des commandes :

- Tester l'exécution de différentes commandes (`SEND_NAME`, `CHAT`, `START_GAME`, etc.) et vérifier les réponses attendues et les actions effectuées.

Interaction avec `Lobby` :

- Simuler des scénarios où le client interagit avec le lobby, comme rejoindre ou quitter des sessions de jeu.
- Vérifier que le lobby met à jour correctement son état en réponse aux actions du client.

5.4 Classe `GameConfig`

La classe `GameConfig` stocke et gère les paramètres de configuration d'une session de jeu, tels que la durée du jeu, le niveau de difficulté, et le nombre de mots. Les tests suivants visent à s'assurer de la fiabilité et de l'efficacité de ces fonctionnalités. :

5.4.1 Tests Unitaires

Test des Constructeurs :

- Vérifier que tous les paramètres (nombre maximum de joueurs, durée de la partie, niveau de difficulté, etc.) sont correctement initialisés.
- Tester les valeurs par défaut pour les paramètres non spécifiés dans le constructeur.

Tests des Getters et Setters :

- Vérifier que chaque setter modifie correctement l'attribut correspondant.
- Tester que chaque getter retourne la valeur attendue après l'initialisation et après des modifications.

Tests des Valeurs Limites :

- Tester les setters avec des valeurs extrêmes (par exemple, durées négatives, nombre de joueurs excessif) pour s'assurer que la classe gère correctement ces cas.

5.4.2 Tests d'Intégration

Intégration avec Lobby et GameSession :

- Simuler l'utilisation de `GameConfig` lors de la création d'une session de jeu pour vérifier que les configurations sont correctement appliquées.
- Tester la mise à jour des configurations en cours de jeu et vérifier que les changements sont effectués en temps réel dans le comportement du jeu.

Consistance de la Configuration :

- Vérifier que les modifications de la configuration ne perturbent pas les sessions de jeu déjà en cours de manière inattendue.

5.5 Classe GameInfo

La classe `GameInfo` encapsule les détails d'une partie de jeu, tels que le port du serveur, le nombre de joueurs, le niveau de difficulté, la durée du jeu, et d'autres paramètres pertinents. Ces tests assurent l'intégrité et la fiabilité des informations tout au long du jeu.

5.5.1 Tests Unitaires

- **Test des Constructeurs :**
 - Vérifier que tous les paramètres (port, nombre de joueurs, niveau de difficulté, durée du jeu) sont correctement initialisés.
 - Tester les conditions initiales pour s'assurer qu'aucune donnée essentielle n'est omise.
- **Tests des Getters et Setters :**
 - Confirmer que chaque getter retourne précisément ce qui a été configuré par les setters ou le constructeur.
 - Tester les setters pour vérifier qu'ils modifient correctement les attributs internes, avec des cas tests incluant des valeurs limites et incorrectes.
- **Test de la méthode update :**
 - Vérifier que les attributs sont correctement mis à jour lors de l'appel de cette méthode.
 - Assurer que les valeurs incorrectes sont gérées sans provoquer de crash (par exemple, format de numéro incorrect).
- **Test des méthodes fromString et toString :**
 - Confirmer que la méthode `fromString` reconstitue correctement un objet `GameInfo` à partir d'une chaîne de caractères formatée.
 - Tester que `toString` retourne une représentation fidèle de l'état actuel de l'objet.

5.5.2 Tests d'Intégration

- **Intégration avec le système de jeu :**
 - Tester l'utilisation de `GameInfo` dans le contexte du jeu pour s'assurer que les informations sont correctement transmises entre le serveur et les clients.
 - Simuler des scénarios où `GameInfo` est mis à jour en cours de jeu et vérifier que les changements sont bien communiqués à tous les clients.
- **Consistance des données entre sessions :**
 - Vérifier que les modifications apportées à `GameInfo` lors d'une session ne perturbent pas inopinément d'autres sessions ou parties du jeu.

Dans chaque cas, les anomalies ont été documentées et classées selon leur sévérité. Des correctifs ont été développés, testés et intégrés dans le projet principal pour les problèmes critiques, tandis que les problèmes moins urgents ont été planifiés pour résolution dans les cycles de développement suivants. Cette approche méthodique a permis une amélioration continue de la stabilité et de la performance de l'application.

6 Conclusion

6.1 Conclusion générale de groupe

La finalisation du projet Wordle Multijoueur a été une entreprise collective qui nous a permis de renforcer nos compétences techniques et de travailler ensemble de manière harmonieuse. Ce projet a été l'occasion de transformer une idée initiale en un jeu interactif, dynamique et compétitif, tout en intégrant des fonctionnalités avancées telles que le mode multijoueur, les bots intelligents, les animations et les graphiques de scores.

Anthony Genti a apporté une contribution majeure en perfectionnant les animations et en optimisant l'architecture du jeu. Son travail sur les classes d'animation et les améliorations des fonctionnalités existantes a grandement enrichi l'expérience utilisateur, rendant le jeu plus attractif et fluide.

Youssef Boudount a joué un rôle crucial dans la gestion des interactions réseau et des sessions de jeu. Son travail sur les classes `Server`, `Lobby`, `GameSession` et `ClientHandler` a assuré une communication stable et efficace entre les clients et le serveur, garantissant une expérience multijoueur cohérente et réactive.

Louis Rives-Lehtinen s'est distingué par son développement des bots intelligents, offrant aux joueurs des défis à différents niveaux de difficulté. L'intégration des bots a non seulement enrichi le gameplay mais a également permis de compléter les parties multijoueur en l'absence de joueurs humains, ajoutant une nouvelle dimension au jeu.

Seghir Samy a su intégrer des graphiques dynamiques permettant de visualiser les performances des joueurs de manière interactive et ludique. Son travail a permis d'améliorer l'analyse des compétences des utilisateurs et d'encourager une compétition saine et motivante.

Ce projet nous a offert une précieuse expérience de travail en équipe dans un environnement agile. La communication et la collaboration ont été des éléments clés de notre succès, nous permettant de surmonter les défis techniques et conceptuels rencontrés tout au long du développement.

En conclusion, le projet Wordle Multijoueur a été une aventure passionnante et formatrice pour chacun de nous. Les compétences techniques acquises, les défis relevés et les leçons tirées sont des atouts précieux que nous emporterons dans nos futures entreprises.

6.2 Conclusion personnelle finale (Youssef)

La reprise et la finalisation du projet Wordle en Java ce semestre ont représenté une étape décisive et enrichissante dans mon parcours de développeur. L'ajout de la fonctionnalité multijoueur a transformé notre projet initial en une plateforme interactive, dynamique et compétitive. Travailler sur ce projet m'a permis de renforcer mes compétences techniques et de découvrir de nouvelles perspectives en développement logiciel.

L'implémentation de la classe `Server` m'a particulièrement marqué, car elle m'a offert une compréhension profonde des interactions réseau et de la gestion des connexions dans un environnement multijoueur. Gérer les connexions simultanées et assurer la fluidité des interactions a été un défi technique stimulant qui m'a permis de mettre en pratique les concepts d'encapsulation, de modularité et de réutilisabilité des composants.

La création de la classe `Lobby` m'a permis de comprendre l'importance de la gestion des sessions de jeu et des clients en attente. Assurer une transition fluide des joueurs de la liste d'attente aux sessions actives et gérer simultanément plusieurs sessions de jeu ont été des tâches complexes mais enrichissantes.

Le développement de la classe `ClientHandler` a également été une expérience précieuse, car elle m'a permis de comprendre l'importance de la gestion des communications entre le client et le serveur, ainsi que la nécessité de maintenir un état cohérent de l'applica-

tion. Gérer les exceptions et assurer une déconnexion propre ont été des aspects essentiels pour garantir la stabilité et la robustesse du système.

L'interface utilisateur, développée avec JavaFX, a été un autre aspect clé de ce projet. Concevoir une interface intuitive et réactive, capable de gérer les interactions en temps réel, a été un défi que j'ai relevé avec enthousiasme. La validation des entrées, la gestion des événements et l'optimisation des performances ont été des aspects cruciaux pour offrir une expérience utilisateur fluide et agréable.

Enfin, ce projet m'a offert une occasion précieuse de travailler en équipe dans un environnement agile. La communication et la collaboration ont été essentielles pour le succès de ce projet. Les défis rencontrés et les solutions trouvées m'ont appris l'importance de la flexibilité et de l'adaptabilité dans le développement logiciel.

En conclusion, le projet Wordle multijoueur a été une aventure fascinante et formatrice. Les compétences techniques acquises, les défis surmontés et les leçons apprises sont des atouts précieux que j'emporterai avec moi dans mes futurs projets. Je suis fier du travail accompli et reconnaissant pour les opportunités d'apprentissage offertes par ce projet. Je suis impatient de mettre en pratique ces acquis dans de nouveaux défis et de continuer à développer mes compétences en tant que développeur de logiciels.

6.3 Conclusion personnelle (Anthony)

Au cours de la phase finale du projet de développement du jeu Wordle multijoueur, j'ai continué à contribuer de manière substantielle à plusieurs aspects cruciaux de l'architecture du jeu. Mon travail s'est concentré sur l'achèvement de la conception et l'implémentation de plusieurs classes clés, ainsi que sur l'optimisation des fonctionnalités existantes pour assurer une expérience de jeu fluide et robuste.

L'un des éléments centraux de ma contribution a été le perfectionnement de la classe **Player** et de ses dérivés, **HumanPlayer**. Cette tâche m'a permis de peaufiner les principes de programmation orientée objet tels que l'encapsulation, l'héritage et le polymorphisme, tout en adaptant ces concepts pour répondre aux besoins spécifiques d'un environnement de jeu interactif et dynamique.

Le développement de la classe **GameSession** a été particulièrement enrichissant mais aussi challenging, m'ayant confronté à la gestion de la concurrence et de l'état partagé dans un environnement multijoueur. J'ai dû faire preuve de rigueur pour assurer que l'accès aux ressources partagées soit sécurisé et cohérent, en utilisant des mécanismes comme **AtomicBoolean**. J'ai également introduit des améliorations pour optimiser la performance et la réactivité de la session de jeu, assurant une meilleure expérience utilisateur.

J'ai également continué à travailler sur l'interface **ServerMessageListener**, qui joue un rôle pivot dans la facilitation de la communication entre le serveur et les clients. Cette partie du projet a souligné l'importance du découplage dans la conception logicielle, permettant une plus grande flexibilité et facilité de maintenance du système. Des tests approfondis ont été menés pour s'assurer que la communication reste fluide et fiable, même sous forte charge.

Une autre contribution significative a été l'implémentation d'animations pour enrichir l'expérience utilisateur en mode multijoueur. Pour un jeu comme Wordle, les animations peuvent inclure des éléments visuels tels que : l'animation des lettres, des effets de transition entre les différentes étapes du jeu, et des notifications de victoire ou de défaite. Ces animations ont été développées pour améliorer l'interactivité et l'engagement des joueurs, rendant le jeu plus attractif et intuitif.

Tout au long de ce projet, j'ai rencontré des défis qui m'ont poussé à approfondir ma compréhension des patterns de conception et des meilleures pratiques en programmation. Par exemple, l'élaboration de méthodes de gestion des erreurs et de validation des données entrantes dans les classes **GameInfo** et **GameConfig** a été cruciale pour prévenir des

défaillances du système en production.

Bien que le projet soit maintenant achevé, cette évaluation finale a été une opportunité précieuse pour réfléchir sur le travail accompli et sur les compétences que j'ai pu développer. Les difficultés rencontrées m'ont appris l'importance de concevoir avec soin et de tester rigoureusement, des leçons que je continuerai d'appliquer dans mes futurs projets.

En regardant vers l'avenir, il serait intéressant d'explorer des fonctionnalités supplémentaires pour le jeu, telles que des modes de jeu variés et des options de personnalisation pour les joueurs, afin de continuer à enrichir l'expérience utilisateur.

6.4 Conclusion personnelle (Samy)

L'ajout de cette fonctionnalité dont je me suis occupé permet non seulement de suivre les performances des joueurs de manière visuelle et interactive, mais elle enrichit également l'expérience utilisateur en rendant les parties plus dynamiques et compétitives.

Chaque graphique offre une perspective unique sur les performances des joueurs, ce qui encourage une analyse plus fine et une meilleure compréhension de leurs forces et faiblesses.

Ce projet m'a permis de développer des compétences avancées en Java et en utilisation de bibliothèques graphiques comme JavaFX. De plus, l'intégration des graphiques de scores m'a apporté une compréhension approfondie de la visualisation de données.

En conclusion, l'implémentation des graphiques de scores dans le projet Wordle Multi-joueur a été une expérience enrichissante et stimulante, tout en offrant aux joueurs des outils pour suivre et analyser leurs performances de manière ludique et interactive.

6.5 Conclusion personnelle (Louis)

La réalisation de ce projet Wordle Multijoueur représente pour moi une expérience très enrichissante. Notre collaboration au sein de l'équipe est caractérisée par une harmonie et un soutien mutuel constants, ce qui a grandement contribué à l'avancée du projet.

L'intégration du Bot a considérablement enrichi le jeu Wordle, offrant aux joueurs la possibilité de relever des défis à différents niveaux de difficulté, ainsi que de compléter une partie multijoueur s'il manque des joueurs. En concevant et en mettant en œuvre diverses stratégies, nous avons exploré de nouvelles perspectives dans le domaine des jeux.

Ce projet m'a permis d'acquérir de précieuses compétences techniques, notamment en programmation et en conception de jeux, ainsi qu'en travail d'équipe et en gestion de projet. Chaque membre de notre équipe a apporté une contribution significative, mettant en avant nos forces individuelles tout en travaillant ensemble efficacement pour atteindre nos objectifs communs.

Au-delà des compétences techniques, ce projet m'a également offert une opportunité unique d'apprentissage et de développement personnel. La résolution de problèmes complexes, la prise de décisions stratégiques et la recherche de solutions novatrices ont nourri ma curiosité intellectuelle et ma passion pour le développement de jeux.

En conclusion, le projet Wordle Multijoueur a été une aventure passionnante, ponctué de défis stimulants et de moments gratifiants. Grâce à notre engagement, notre détermination et notre esprit d'équipe, nous avons réussi à créer un début de jeu multijoueur divertissant et enrichissant.

7 Annexes

Cette section des annexes comprend des informations supplémentaires et des ressources qui complètent le corps principal du rapport. Les annexes offrent un aperçu détaillé des aspects techniques du projet, illustrant concrètement les méthodes et les résultats obtenus.

7.1 Code source commenté et documentation technique

Le code source commenté de notre projet est fourni pour offrir une compréhension approfondie de la structure et de la logique de notre application Wordle. Chaque segment du code est soigneusement commenté, expliquant non seulement la fonctionnalité implémentée, mais aussi les motivations derrière certaines décisions de conception. Cette approche vise à rendre le code accessible et compréhensible, facilitant ainsi la maintenance et les éventuelles extensions futures. Pour un accès complet au code source et à sa documentation technique, veuillez consulter notre dépôt GitLab : <https://gitlab.com/ceri-projet-programmation-2023/semestre2-groupe3>.

7.2 Copies d'écran de l'application

Les captures d'écran de l'application illustrent les améliorations apportées à l'interface utilisateur et aux nouvelles fonctionnalités du mode multijoueur. Ces images montrent non seulement le résultat de notre travail sur l'interface graphique mais aussi les nouveaux éléments interactifs tels que les options de jeu solo et multijoueur, la gestion des parties et les graphiques de scores, offrant une vue tangible de l'expérience utilisateur améliorée.

7.2.1 Écran d'Accueil

L'écran d'accueil offre désormais le choix entre les modes de jeu solo et multijoueur, permettant aux utilisateurs de sélectionner leur expérience préférée.

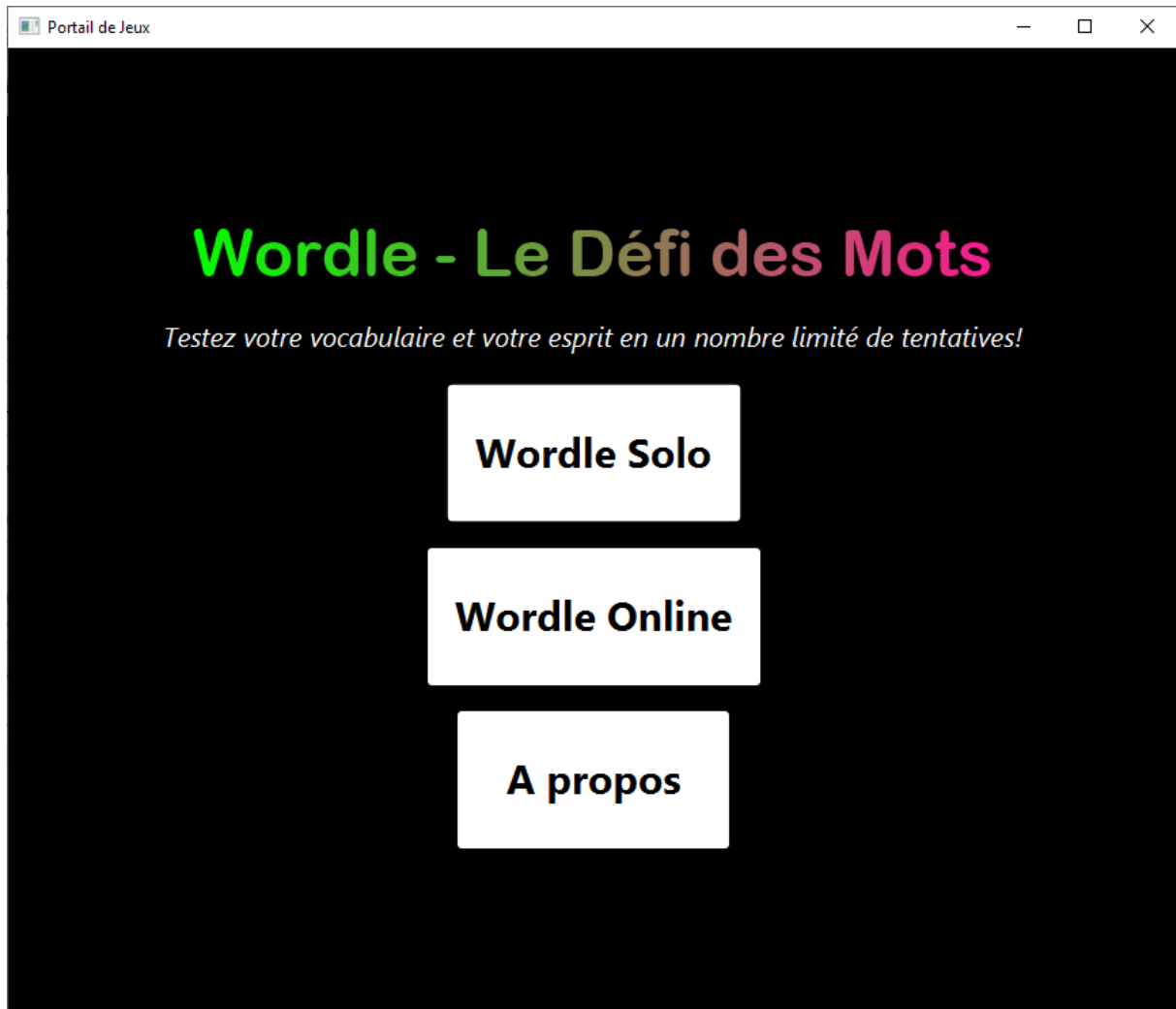


Figure 5. Écran d'Accueil – Choix du mode de jeu

7.2.2 Fenêtre Multijoueur

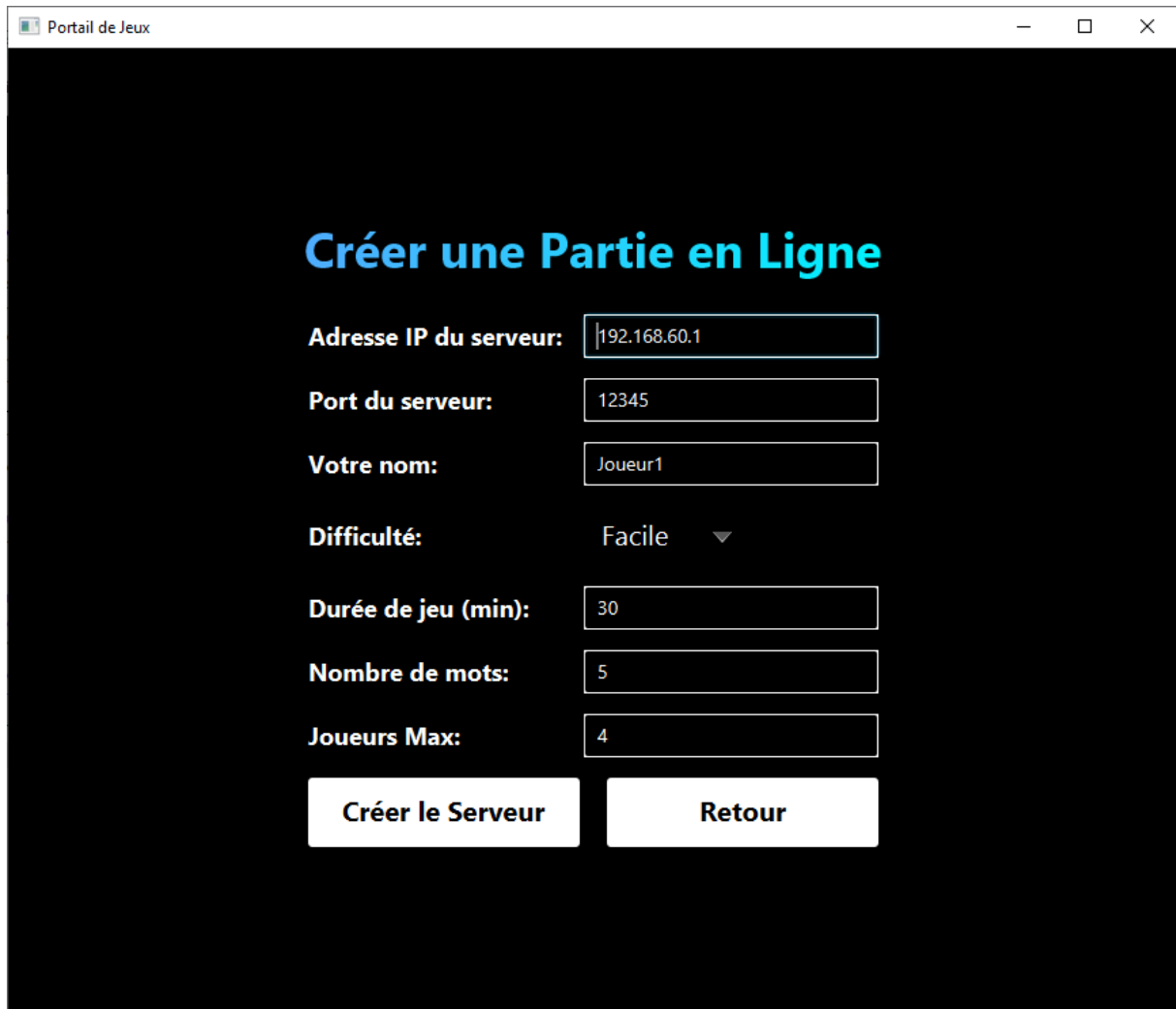
Cette fenêtre permet aux utilisateurs de choisir entre créer une nouvelle partie ou rejoindre une partie existante.



Figure 6. Fenêtre Multijoueur – Options de création et de rejoindre

7.2.3 Fenêtre de Création de Partie

La fenêtre de création de partie permet aux utilisateurs de configurer les paramètres du jeu, tels que le nombre de joueurs et la durée de la partie.



The screenshot shows a web application window titled 'Portail de Jeux'. The main heading is 'Créer une Partie en Ligne' in a large, bold, light blue font. Below the heading, there are several form fields for creating a game session:

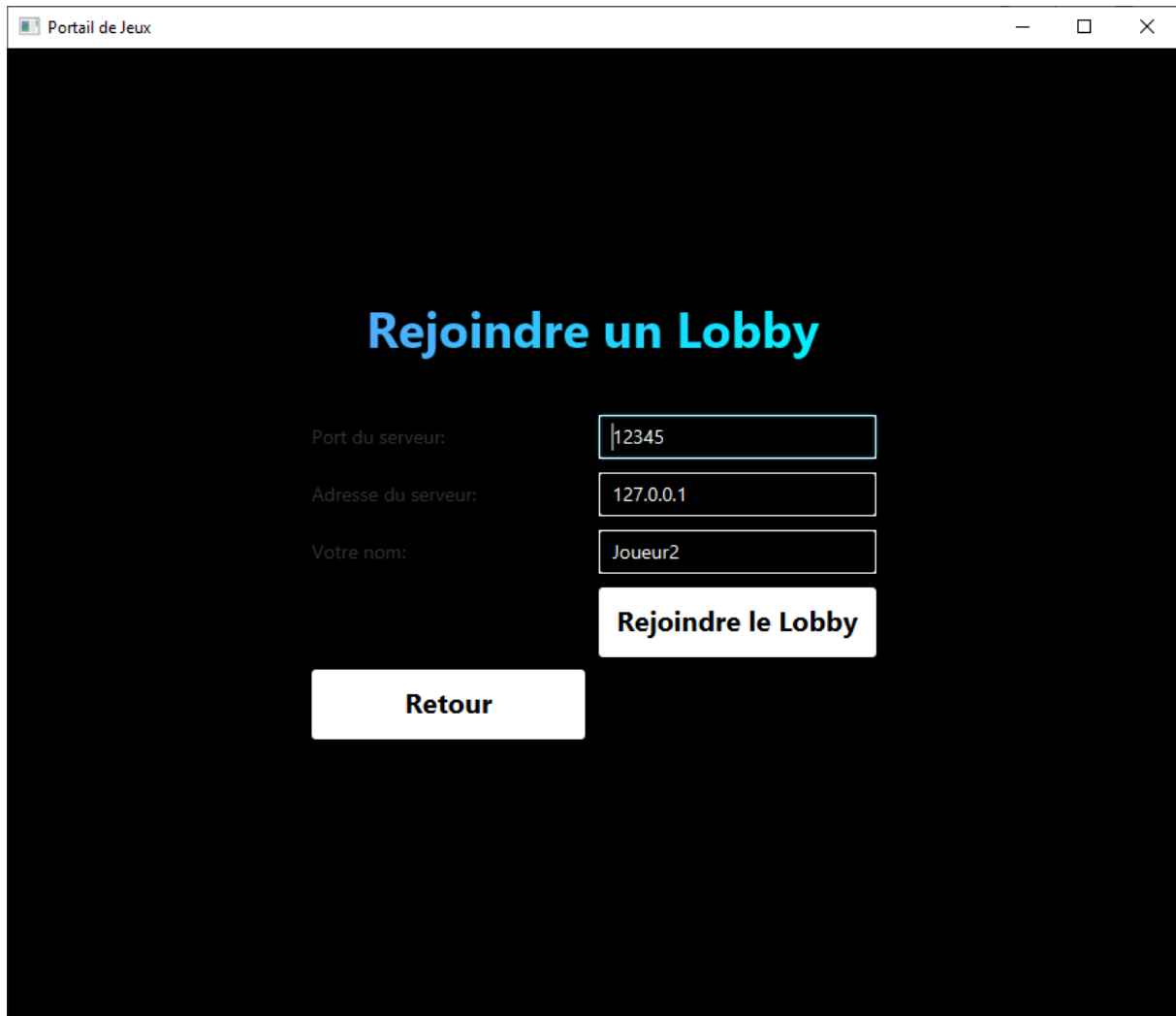
- Adresse IP du serveur:** A text input field containing '192.168.60.1'.
- Port du serveur:** A text input field containing '12345'.
- Votre nom:** A text input field containing 'Joueur1'.
- Difficulté:** A dropdown menu with 'Facile' selected and a downward arrow.
- Durée de jeu (min):** A text input field containing '30'.
- Nombre de mots:** A text input field containing '5'.
- Joueurs Max:** A text input field containing '4'.

At the bottom of the form, there are two buttons: 'Créer le Serveur' and 'Retour'.

Figure 7. Fenêtre de Création de Partie

7.2.4 Fenêtre pour Rejoindre une Partie

Cette interface permet aux joueurs de saisir un code de jeu pour rejoindre une partie déjà créée par un autre utilisateur.



The screenshot shows a window titled "Portail de Jeux" with a black background. In the center, the text "Rejoindre un Lobby" is displayed in a large, bold, cyan font. Below this, there are three input fields with labels to their left: "Port du serveur:" with the value "12345", "Adresse du serveur:" with the value "127.0.0.1", and "Votre nom:" with the value "Joueur2". To the right of these fields is a white button with the text "Rejoindre le Lobby". Below the input fields is another white button with the text "Retour".

Figure 8. Fenêtre pour Rejoindre une Partie

7.2.5 Lobby de la Partie

Le lobby affiche les joueurs actuellement connectés et prêts à commencer la partie, ainsi que les options pour démarrer le jeu.

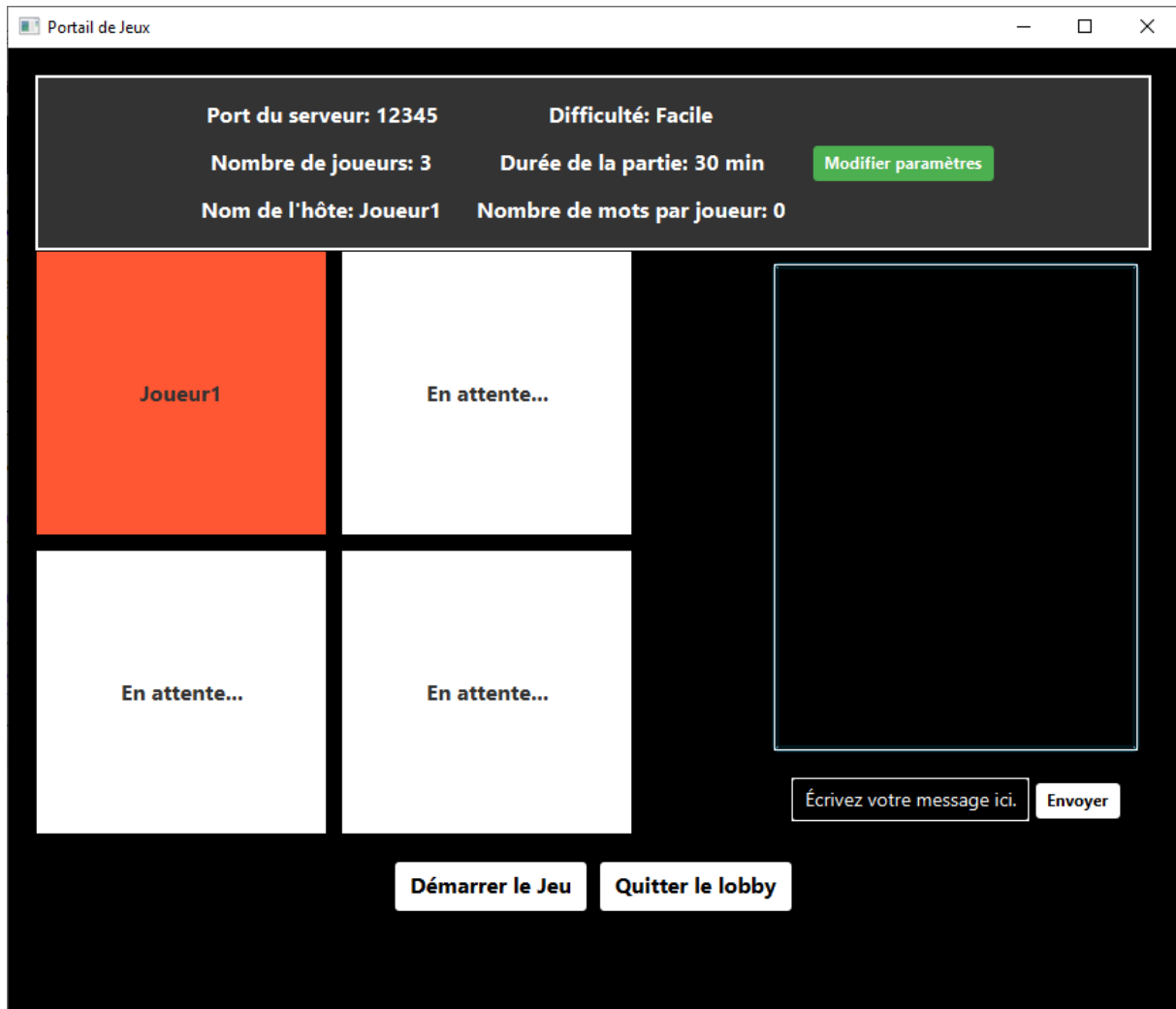


Figure 9. Lobby de la Partie

7.2.6 Fenêtre du Graphique des Scores

Cette fenêtre affiche les graphiques des scores des joueurs à la fin de la partie, permettant une analyse visuelle des performances variée.

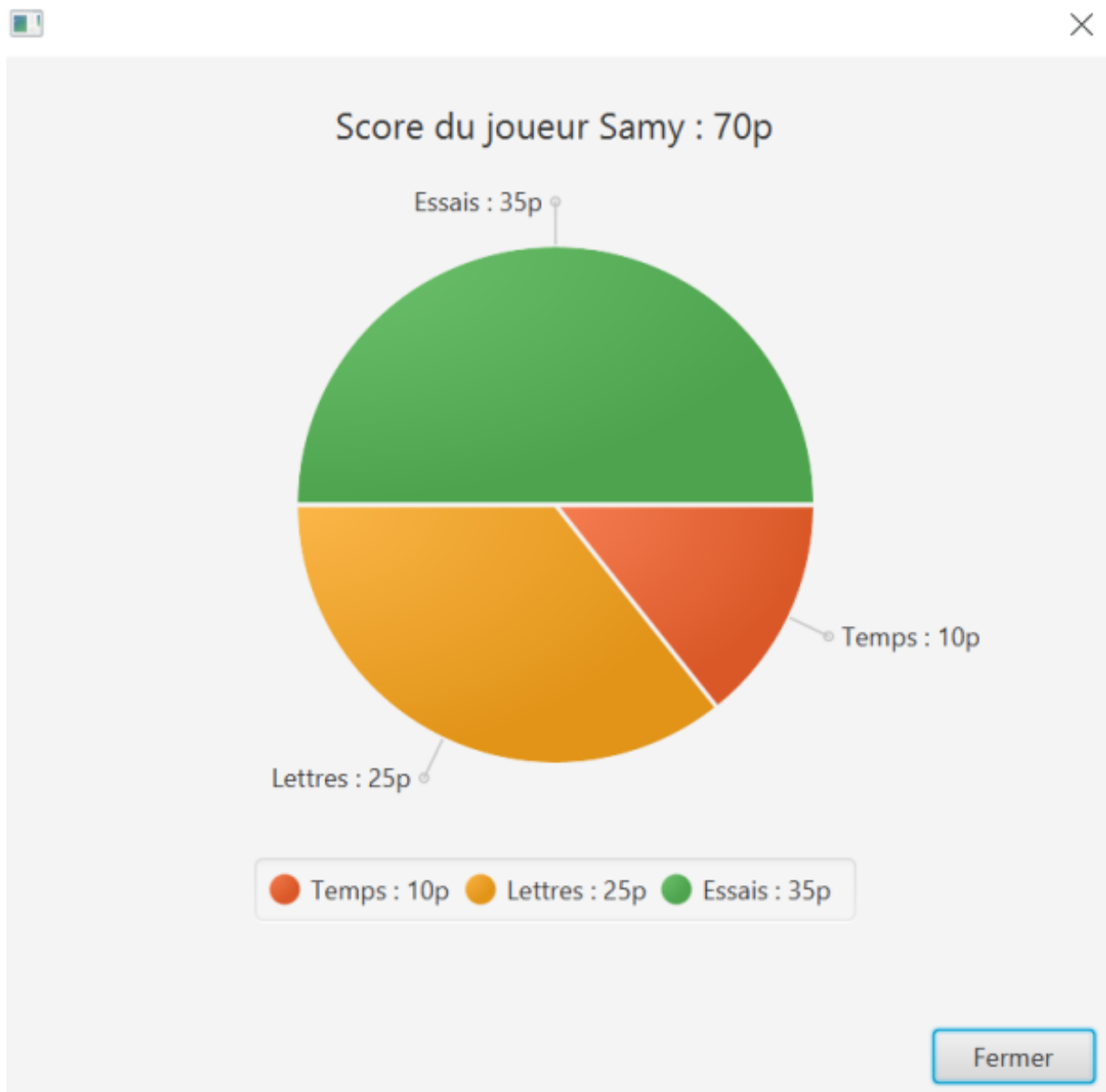


Figure 10. Graphique circulaire des Scores

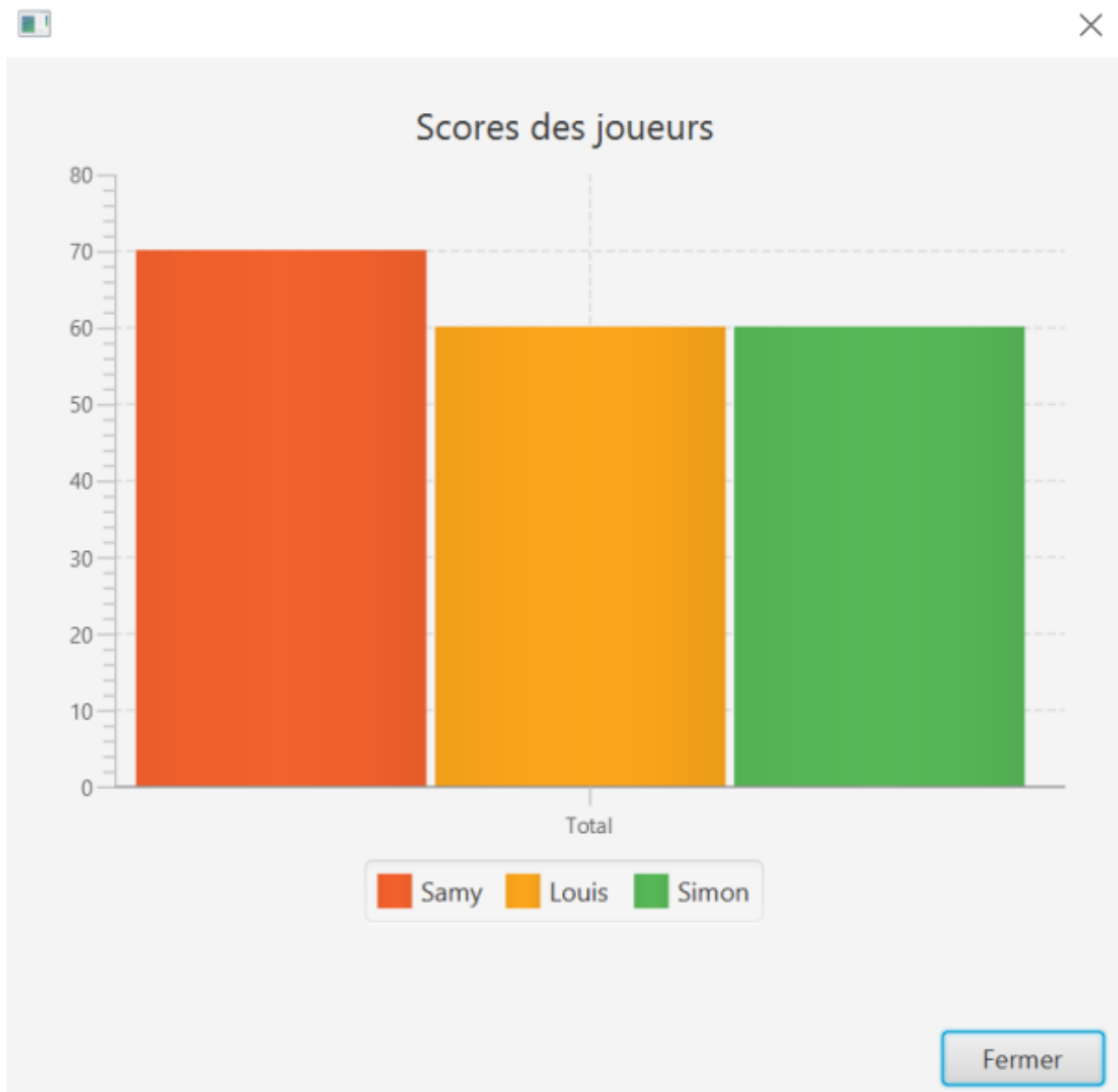


Figure 11. Graphique barre des Scores

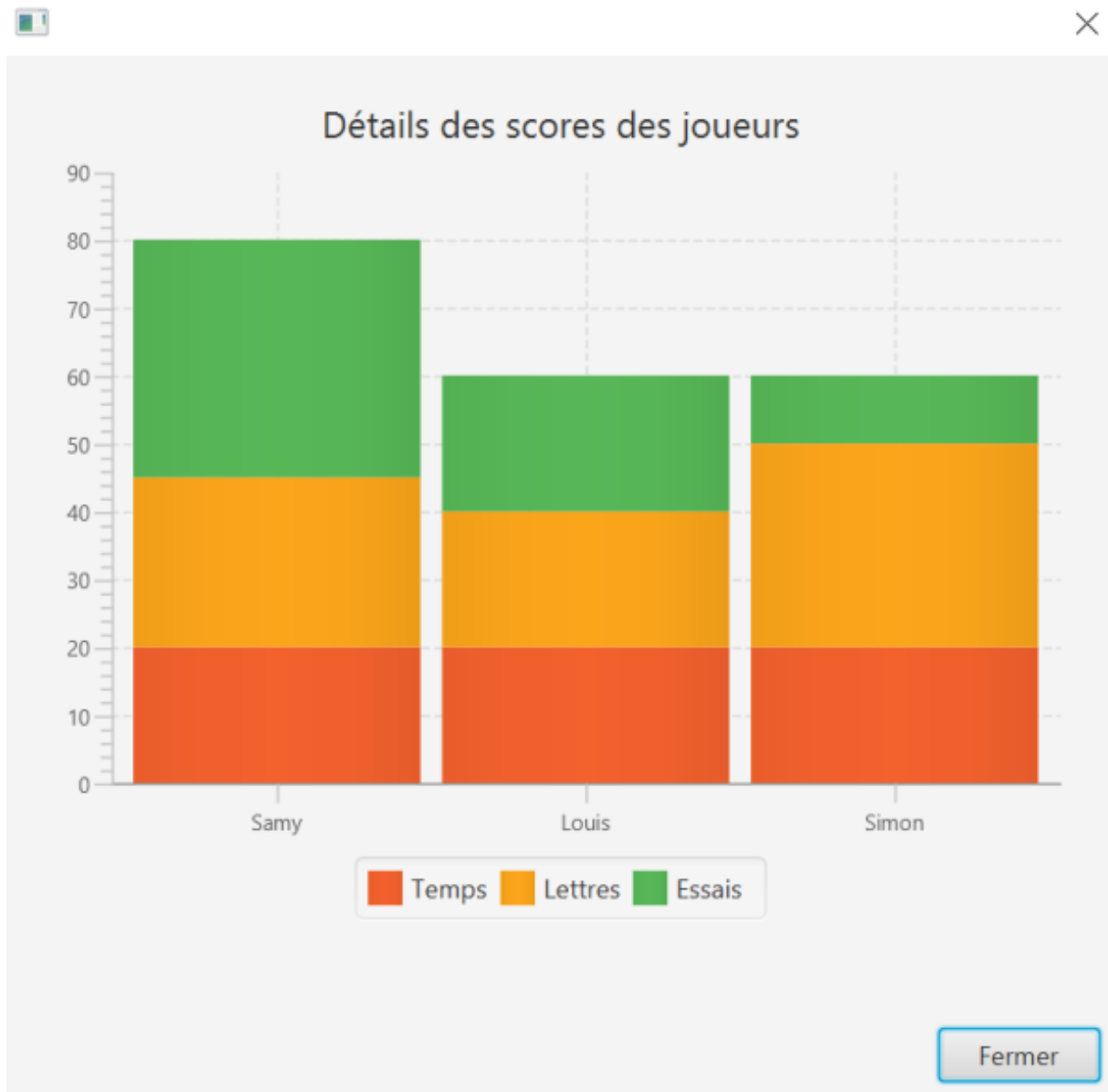


Figure 12. Graphique détaillé des Scores

7.3 README du Projet

Projet Wordle en Java :

Description

Ce projet est une implémentation en Java du jeu de vocabulaire Wordle, réalisé dans le cadre

Fonctionnalités

Portail général au lancement de l'application pour sélectionner le jeu.

Affichage des éléments du jeu : grille de jeu, clavier virtuel.

Jouabilité selon les règles classiques de Wordle.

Sauvegarde des informations relatives à chaque partie.

Système d'indices basé sur la similarité entre mots avec Word2vec.

Installation et Exécution

Assurez-vous d'avoir Java et Python installés sur votre machine.

Clonez le dépôt Git :

```
git clone https://gitlab.com/ceri-projet-programmation-2023/groupe-3.git
```

Ouvrez le projet dans Eclipse ou un autre IDE compatible avec les projets Eclipse.

Lancez le fichier Main.java situé dans le dossier src/view pour démarrer l'application.

Structure du Projet

Le projet est organisé comme suit :

src: Contient le code source Java.

controller: Logique de jeu et gestion des indices.

model: Modèles de données, incluant le traitement des mots.

view: Interface utilisateur du jeu.

resources: Scripts Python et modèle Word2vec.

data: Dictionnaire de mots et scripts de traitement.

bin: Fichiers compilés Java.

Dépendances

JavaFX pour l'interface utilisateur.

Gensim pour Word2vec en Python.

Apache HttpClient pour les requêtes HTTP.

Jsoup pour le scraping Web.

Plusieurs autres bibliothèques Java, voir pom.xml ou équivalent pour plus de détails.

Anthony Genti

Youssef Boudount

Louis Rives-Lehtinen

Samy Seghir

Licence

Ce projet est publié sous la licence GPL.

8 Références

Cette section compile l'ensemble des ressources, bibliothèques et documents de référence utilisés tout au long du développement du projet. Ces références ont joué un rôle crucial dans la conception et l'implémentation des divers aspects du jeu Wordle, en nous fournissant les outils et les connaissances nécessaires pour surmonter les défis techniques rencontrés.

8.1 Bibliothèques et outils utilisés

Dans le cadre de ce projet, nous avons utilisé une gamme de bibliothèques et d'outils pour faciliter le développement et améliorer les fonctionnalités de notre application :

- **JavaFX** : Pour la construction de l'interface utilisateur, nous avons utilisé JavaFX, une bibliothèque puissante qui nous a permis de créer une interface graphique riche et interactive.
 - **javafx.application.Application** : bibliothèque de base pour les applications JavaFX.
 - **javafx.geometry.Pos** : Utilisé pour gérer l'alignement au centre des éléments du portail et du jeu dans le container
 - **javafx.scene.Scene** : Bibliothèque qui gère les scènes de fenêtres
 - **javafx.scene.control.Button** : Utilisé pour créer des boutons (clavier virtuel)
 - **javafx.scene.control.Label** : Utilisé pour créer des labels (Titre du Portail)
 - **javafx.scene.control.TextField** : Utilisé pour créer une zone de texte (cellule de la grille de jeu)
 - **javafx.scene.layout.*** : Bibliothèque qui permet d'utiliser tout les container (border-Pane, StackPane, VBox, Hbox)
 - **javafx.stage.Stage** : Bibliothèque qui gère les fenêtres (Stage)
 - **javafx.stage.StageStyle** : Utilisé pour définir le style de la fenêtre (Stage) (en UTILITY)
 - **javafx.scene.Node** : Utilisé pour obtenir les cellules de la grille de jeu qui sont sous forme de noeuds
- **Apache HttpClient** : Utilisée pour les requêtes HTTP, notamment pour l'intégration avec les API externes et les opérations de scraping web. Les modules spécifiques utilisés incluent :
 - **httpclient5** : Pour les requêtes HTTP de base.
 - **httpclient5-cache** : Pour la mise en cache des requêtes HTTP.
 - **httpclient5-fluent** : Pour une interface de programmation plus fluide.
 - **httpclient5-testing** : Pour tester les requêtes HTTP.
 - **httpclient5-win** : Spécifique aux systèmes Windows.
 - **httpcore5** : Fondement des requêtes HTTP.
 - **httpcore5-h2** : Support du protocole HTTP/2.
 - **httpcore5-reactive** : Pour le traitement asynchrone.
 - **httpcore5-testing** : Pour tester les fonctionnalités HTTP de base.
- **Jansi** : Pour améliorer la sortie dans la console, permettant des affichages colorés et formatés.
- **JNA (Java Native Access)** : Bibliothèque qui permet aux programmes Java d'appeler des fonctions de bibliothèques partagées natives.
 - **jna** : Fonctionnalités de base de JNA.
 - **jna-platform** : Extensions spécifiques à la plateforme.
- **Reactive Streams** : Pour le développement d'applications asynchrones non bloquantes.

- **reactive-streams** : API standard pour la programmation asynchrone avec des flux de données.
- **Apache Commons CLI** : Utilisé pour l'analyse des lignes de commande dans les applications.
 - **commons-cli** : Fournit des outils pour interpréter les arguments de ligne de commande.
- **Apache Commons Lang** : Bibliothèque utilisée pour les opérations courantes sur les chaînes de caractères, les collections et les dates. Les composants spécifiques comprennent :
 - **commons-lang3** : Pour les opérations sur les chaînes de caractères et autres utilitaires courants.
 - **javadoc, sources, test-sources, tests** : Modules supplémentaires pour la documentation, le code source, les tests, etc.
- **Gson** : Pour la manipulation des données JSON, notamment dans le traitement des scores et la gestion des configurations.
- **Eclipse** : L'environnement de développement intégré (IDE) Eclipse a été notre principal outil de développement, offrant un support robuste pour Java et une intégration facile avec diverses bibliothèques et plugins.
- **Jsoup** : Pour le scraping des données du Wiktionnaire, Jsoup a été essentiel pour analyser et extraire les informations nécessaires depuis les pages web.
- **Python**
 - **gensim** : Sert à charger et utiliser le modèle word2vec pour le module d'indices.
 - **flask** : Sert à créer le serveur HTTP pour faire des requêtes depuis le module d'indices.
- **Autres.**
 - **java.util.Random** : Utilisé pour générer une taille de grille aléatoire dans le jeu Wordle (entre 2 et 14 colonnes)

8.2 Sources documentaires et tutoriels suivis

Pour compléter notre compréhension théorique et pratique des technologies et techniques employées, nous avons consulté diverses sources documentaires et suivi plusieurs tutoriels :

- **Documentation officielle de JavaFX** : Pour maîtriser les aspects de l'interface utilisateur JavaFX, la documentation officielle a été une source d'information inestimable.
[Openjfx – install javaFX](#)
[Documentation BorderPane](#)
[Documentation VBox](#)
[Documentation TextField](#)
- **Tutoriels JavaFX sur YouTube** : Plusieurs tutoriels sur YouTube ont été suivis pour obtenir des conseils pratiques et des démonstrations sur la mise en place d'interfaces utilisateur complexes.
[Tuto install javafx](#)
- **Documentation de Jsoup** : La documentation officielle de Jsoup a guidé notre utilisation de cette bibliothèque pour le scraping web efficace.
- **Stack Overflow** : Pour les problèmes de codage spécifiques et les questions techniques, Stack Overflow a été une ressource fréquemment consultée.
- **Blogs de développement Java et forums** : Des articles techniques et des discussions

de forums ont été consultés pour des conseils sur les meilleures pratiques de développement Java et la résolution de problèmes complexes.

[GeeksforGeeks - Tutoriel JavaFX](#)

[Gestion des fenêtres](#)

[Layout JavaFX](#)

[HBox et VBox](#)

[Style des fenêtres](#)

[Documentation GridPane](#)

[Documentation GridPane](#)

[Responsive](#)

- **Tutoriels et guides en ligne sur le traitement JSON avec Gson** : Ces ressources ont été essentielles pour comprendre l'utilisation efficace de Gson dans le contexte de notre application.

Ces références ont été essentielles pour nous guider à travers les différentes phases du projet, en nous offrant les connaissances et les outils nécessaires pour développer une application robuste et fonctionnelle.

9 Remerciements

Nous tenons à exprimer notre sincère gratitude à toutes les personnes qui ont contribué au succès de ce projet. Leur soutien, leurs conseils et leur expertise ont été inestimables tout au long de cette aventure.

Nous adressons tout d'abord nos plus chaleureux remerciements à nos professeurs et superviseurs de projet, M. Noe CECILLON et M. Jarod DURET. Leurs précieux conseils, leur encadrement rigoureux et leur patience ont été des piliers dans la réalisation de ce projet. Leurs orientations éclairées et leurs critiques constructives ont grandement contribué à la direction et à la qualité de notre travail.

Nous exprimons également notre reconnaissance envers l'ensemble du corps enseignant du CERI pour leur enseignement rigoureux et leur engagement à fournir une éducation de qualité, nous préparant ainsi à relever les défis de ce projet.

Un remerciement spécial à nos camarades de classe, pour l'environnement stimulant qu'ils ont créé, et pour toutes les discussions enrichissantes et les séances de brainstorming partagées. Cette atmosphère collaborative a été un élément clé de notre motivation et de notre progression.

Enfin, nous tenons à remercier toutes les personnes qui ont indirectement contribué à ce projet, en particulier les développeurs et les contributeurs des bibliothèques et outils que nous avons utilisés. Leur travail a été essentiel pour réaliser nos objectifs techniques.

Cette expérience a été incroyablement formatrice, et nous sommes fiers de ce que nous avons accompli ensemble. Merci à tous pour avoir rendu cela possible.

L'équipe du projet Wordle - Groupe 3