Rapport Boggle

Structure du code

Notre code est réparti dans quatre dossiers qui contiennent chacun les classes liées à un aspect du code.

Le dossier “util” contient les classes auxiliaires, qui permettant d’aider dans le reste du code. Par exemple, on y trouve “Defaults”, qui contient les chemins par défaut ou se trouvent les fichiers de sauvegardes et d’historique, ou encore “Logger” qui est un outil permettant d’afficher des informations de débogage.

Le dossier “ouga” contient l’entrée de notre programme. C’est là que se trouve la classe qui sera lancée en premier, et qui s’occupe de traiter les options de ligne de commande.

Le dossier “client” contient tout ce qui concerne le côté client de l’application. Dans le dossier “affichage” se trouvent les classes des différentes fenêtres qui composent la fenêtre principale du jeu, telle que l’affichage du chat, des points ou encore de la grille. Le dossier “client” contient également les classes qui gèrent les interfaces graphiques du lancement du serveur, de la partie et celle du jeu. Enfin, le dossier contient la classe Client qui est le cerveau de l’application côté client : elle s’occupe de communiquer avec le serveur et de réagir à ses messages.

Le dernier dossier est le dossier “serveur” qui regroupe quant à lui toutes les classes relatives au côté serveur de l’application. Ces classes sont elles-mêmes rangées dans différents sous-dossiers : “dictionnaire”, “jeu”, “message” et “plateau”. Le dossier “jeu” contient essentiellement les éléments généraux d’une partie, tels que la classe Joueur, la classe Manche ou encore les modes de jeu. Le dossier “message”, en revanche, contient les classes qui représentent les messages échangés entre le client et le serveur (“DebutManche”, “FinPartie”, …). Le dossier “dictionnaire” (comme son nom l’indique), contient la logique du dictionnaire, celle qui vérifie si un mot est dans le dictionnaire ou s’il ne l’est pas. Enfin, le dossier “plateau” contient les classes concernant de près ou de loin

le plateau, comme la grille et les fonctions qui la génèrent, les lettres et les mots.

Gestion du cahier des charges

Pour générer la grille aléatoirement, nous avons tout d'abord créé une fonction renvoyant une lettre aléatoire, qu'on appelait autant de fois que nécessaire. Cependant, cela était problématique. En effet, les voyelles représentent seulement 19 % des lettres de l'alphabet , mais apparaissent en proportion de 41 % en moyenne dans les mots de la langue française. Cela voulait dire que très souvent, il allait y avoir des grilles avec très peu, voire pas assez de voyelles pour que la grille soit jouable. C'est pour cela que nous avons adapté notre algorithme, pour le rendre pesé. Désormais, au lieu d'avoir une chance sur 26 d'avoir chaque lettre, on aura plus de chances d'avoir un e, un a, un i ou encore un s que un q, un z ou encore un k. Les taux sont générés à partir de leur fréquence dans le dictionnaire et sont donc différents pour chaque langue.

Nous avons implémenté un outil en ligne de commande afin de requêter des informations sur les dernières parties. Les informations stockées sont le score de chaque joueur et les mots qu'ils ont trouvés. On peut choisir le nombre de joueurs à afficher, le nombre de parties à afficher et si oui ou non il faut afficher les mots qu'ils ont trouvés.

Pour plus de confort pour le joueur, nous avons décidé d'implémenter aussi bien l'entrée des mots à la souris que l'entrée des mots au clavier. L'entrée à la souris est "click and drag", ce qui veut dire que l'on clique, et on déplace la souris sur la grille pour sélectionner le mot, puis on lâche pour valider. La vérification des mots sélectionnés par la souris via l'interface graphique était la plus facile à implémenter, car chaque lettre de la grille possède des coordonnées. Il suffit alors de vérifier les points suivants : que les lettres sont les unes à côté des autres, qu'une lettre n'est pas utilisée plusieurs fois et que le mot est dans le dictionnaire. Pour l'entrée au clavier, nous avons créé une fonction qui recherche toutes les occurrences de la première lettre du mot dans la grille. À chaque occurrence, on vérifie si la lettre suivante du mot se trouve dans les lettres voisines de la case sur laquelle l'on se trouve. Si c'est le cas, on effectue la même recherche pour la lettre d'après et

ainsi de suite, jusqu'à avoir trouvé toutes les lettres du mot. Si aucune occurrence de la première lettre ne donne un chemin complet, alors le mot ne se trouve pas dans la grille. On maintient aussi une liste des cases sur lesquelles nous sommes déjà passés, pour s'assurer qu'on n'est pas passés sur la même case plusieurs fois.

Une fois qu'on a vérifié que le mot entré par le joueur était bien présent dans la grille, on vérifie s'il est également présent dans le dictionnaire.

Extensions créées

Au fur et à mesure que nous développions le jeu, nous avons ajouté différentes fonctionnalités afin d'améliorer la jouabilité et d'offrir plus de liberté au joueur.

Dans un premier temps, nous avons décidé d'implémenter la possibilité de jouer en multijoueur. Pour faire cela, nous avons ajouté un système de serveur/client qui permet à une personne de lancer un serveur puis d'inviter des personnes à la rejoindre, sans limite de connexion. La classe Serveur s'occupe d'envoyer les informations aux clients (par exemple : un joueur s'est connecté, un joueur a trouvé un mot, une nouvelle manche commence) et de recevoir les informations du client. La classe Client quant à elle s'occupe de transmettre au serveur les actions du joueur (par exemple : il a entré un mot, il a demandé une pause) et de recevoir les informations du serveur.

Pour lancer une partie, le joueur a le choix entre ouvrir les interfaces graphiques lui permettant de sélectionner les différents paramètres du serveur et du client, ou de passer ces informations sous forme de paramètres dans une ligne de commande. Cette fonctionnalité a été implémentée afin que le joueur puisse lancer un serveur à distance (via ssh par exemple). La plupart des paramètres ont des valeurs par défaut. Ainsi, si le joueur n'est pas sûr de comment configurer le serveur et le client, il peut utiliser ces valeurs par défaut et il n'a besoin que de rentrer son pseudonyme. En complément, les options de ligne de commande sont résumées

avec l'option "–help", et sont expliquées plus en détail dans la documentation. Après avoir lancé le serveur et le client, le joueur se retrouve dans le lobby depuis lequel il peut fixer les paramètres de la partie ou choisir de reprendre une partie qu'il a sauvegardée précédemment. Pour sauvegarder la partie en cours, la moitié des joueurs doivent voter pour mettre pause. Au lancement d'une pause, un fichier de sauvegarde enregistrant les différentes informations du jeu est créé et le serveur est fermé. Lorsque le joueur reprend une partie sauvegardée, toutes les informations sont donc restituées.

Nous avons implémenté une fenêtre de chat qui permet d'afficher des informations telles que la vérification des mots, la fin et le début de manche, etc. Mais le joueur peut également interagir avec cette fenêtre via l'entrée clavier en y tapant "/[commande]". Il peut ainsi envoyer un message dans le chat ("/chat mon message") ou bien mettre la partie en pause ("/pause") ce qui lui permet en autre de sauvegarder la partie comme on l'a vu précédemment.

Nous avons par ailleurs choisi de laisser le paramétrage de la partie libre, afin que le joueur puisse fixer les options qu'il préfère. Ces paramètres se décident dans le lobby et incluent la largeur et la longueur de la grille, la durée du minuteur, le nombre de manches, la langue et le mode de jeu. Chacun de ces paramètres à une valeur par défaut. De cette manière, si le joueur ne sait pas quoi mettre, il peut néanmoins lancer le jeu avec des paramètres permettant une expérience agréable. Ces valeurs par défaut sont celles de base du jeu Boggle. Le premier joueur à se connecter au lobby est désigné "chef du lobby", et lui seul peut fixer ces paramètres.

Un autre paramètre laissé au choix du joueur que nous avons implémenté en plus est le mode de jeu de la partie. Le premier mode de jeu est le mode "classique" qui respecte les règles de base du jeu de Boggle. Il peut donc se jouer aussi bien seul qu'à plusieurs. En revanche, le deuxième mode de jeu disponible se joue forcément à plusieurs car il s'agit d'une version "Battle Royale" de Boggle. En effet, à la fin de chaque manche, le joueur cumulant le moins de points se retrouve éliminé du jeu et la manche suivante se lance sans lui. Le jeu dure jusqu'à ce qu'il n'y ait plus qu'un joueur dans la partie et il est alors déclaré vainqueur. Ce mode de

jeu n’a donc pas un nombre de manches fixé , car tout dépend du nombre de joueurs et de leur score : deux joueurs avec le même nombre de points pourront être éliminés simultanément si c’est le score le plus faible. Si les deux finalistes sont à égalité, alors les manches s'enchaînent jusqu’à ce que l’un des deux prennent le dessus. De plus, nous avons implémenté ces modes de jeu dans des classes séparées, mais héritant toutes les deux d’une classe “Jeu” qui contient les fonctions communes aux différents modes. Ainsi, la classe d’un mode ne doit implémenter que les fonctions qui diffèrent selon les règles du mode. Cela permet de rajouter très facilement n’importe quel mode de jeu voulu en ayant que peu, voire pas, de modification à apporter au code, à part l’ajout d’une classe.

Enfin, la dernière fonctionnalité que nous avons décidé d’ajouter par rapport au cahier des charges fixé est un algorithme permettant de trouver tous les mots disponibles dans une grille générée aléatoirement. Cela permet entre autres au joueur d’avoir une idée du nombre de possibilités qui s’offrent à lui. Le nombre total de mots présents dans la grille est affiché dans la fenêtre contenant les informations relatives au score de la partie, ainsi que le nombre de mots trouvés dans cette grille par le joueur.

Problèmes rencontrés

L’une des fonctionnalités qui a été la plus dure à implémenter a été de trouver tous les mots présents dans la grille. Un premier algorithme a été implémenté. Il parcourait toutes les cases de la grille en cherchant tous les chemins possibles autour de cette case pour former un mot. Cependant, il est rapidement apparu que cette solution n’était pas viable, car l’algorithme était beaucoup trop lent et si la grille était trop grande, la partie se finissait avant l’algorithme. Nous avons donc décidé qu’au lieu de partir de la grille pour trouver tous les mots, nous sommes partis du dictionnaire et nous avons testé la présence de chacun de ces mots dans la grille. Cet algorithme s’est immédiatement révélé être beaucoup plus efficace que le précédent, mais lors de tests, nous avons réalisé qu’il annonçait un nombre anormalement grand de mots disponibles par rapport à la grille. Nous avons alors constaté que c’était parce

qu'il comptait également les mots d'une longueur inférieure à trois lettres, tandis que ces mots ne sont pas valables dans le jeu Boggle.

Nous avons pensé qu'il était intéressant de mettre des sons pour indiquer aux utilisateurs si leur mot était accepté ou refusé. Cependant, lors des tests, nous avions une exception qui disait que le format du fichier audio n'était pas reconnu. Or, peu importe le format de l'audio qu'on mettait, on obtenait toujours la même erreur. Nous avons également essayé d'utiliser plusieurs façons de jouer l'audio, mais en vain. Ainsi, on a décidé de mettre un message indiquant si le mot était valide dans le chat.

Un autre problème rencontré fut le rebasage. En effet, lorsque plusieurs personnes travaillent sur une même partie du projet en simultané, cela amène à des conflits. Ceci est arrivé plusieurs fois lors du développement et fut très chronophage.

Le dernier problème qu'on a rencontré était l'algorithme du dictionnaire. Tout d'abord, nous avions implémenté une trie (représentation des mots sous forme d'arbre). Cet algorithme était très performant, permettant de vérifier plusieurs millions de mots par seconde. Cependant, la trie pouvait prendre plusieurs secondes à se générer, voir plus d'une minute sur des petits ordinateurs portables. Nous avons ensuite essayé d'utiliser un HashSet (une structure de données déjà présente dans la librairie standard de java). Cela s'est avéré beaucoup plus rapide à générer (moins d'une seconde, voir deux ou trois secondes sur les petits PCs), tout en améliorant légèrement la vitesse de vérification.

Pistes envisagées

Nous avions envisagé d'autres fonctionnalités pour le jeu que nous n'avons malheureusement pas eu le temps d'implémenter, comme d'autres modes de jeu. Comme mentionné par le paragraphe à ce sujet, notre code a été construit de manière à ce qu'il soit facile d'ajouter de nouveaux modes. De plus, ayant à disposition un algorithme permettant de compter tous les mots présents dans la grille, nous avions imaginé créer un mode de jeu

qui n’aurait pas de minuteur et qui se finirait lorsque tous les mots ont été trouvés, ou que tous les joueurs ont abandonné.

Un autre mode de jeu auquel nous avons pensé consistait à modifier la façon de compter les points en attribuant une valeur à chaque lettre, selon le modèle du Scrabble.

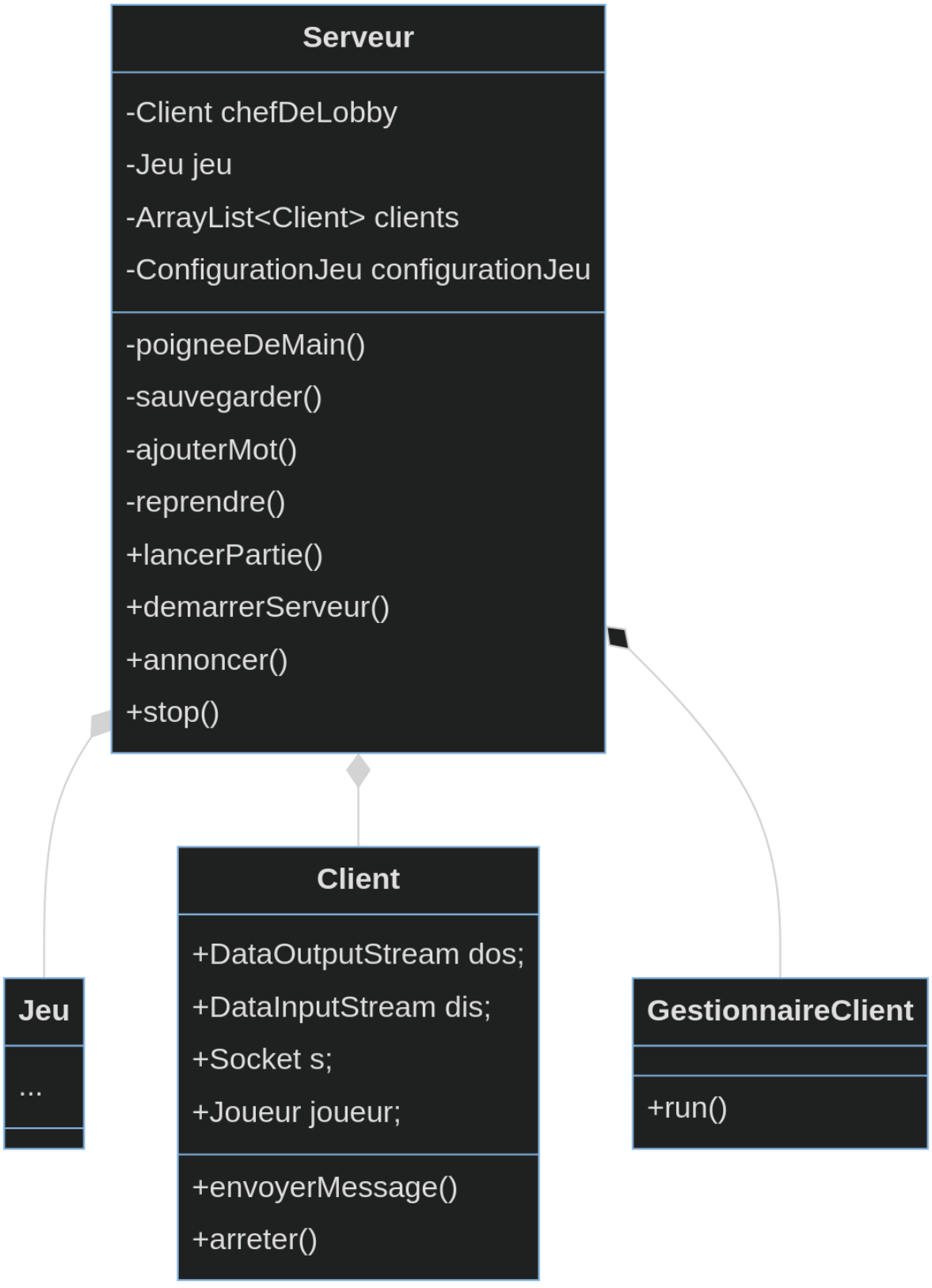
Par ailleurs, nous avons également réfléchi à implémenter d’autres commandes utilisables depuis la fenêtre de saisie au clavier, tel que la demande d’un indice, si l’on est bloqué.

Annexes

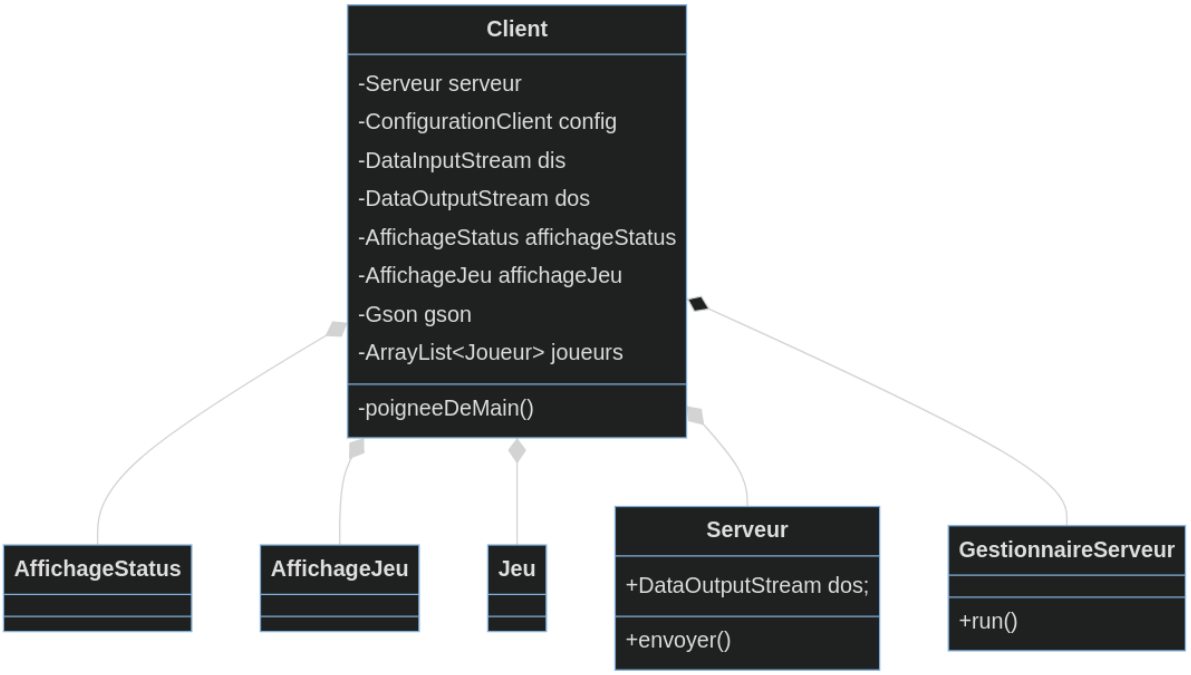
1. Diagrammes

A. Jeu

B. Serveur


Serveur
-Client chefDeLobby
-Jeu jeu
-ArrayList<Client> clients
-ConfigurationJeu configurationJeu
-poigneeDeMain()
-sauvegarder()
-ajouterMot()
-reprendre()
+lancerPartie()
+demarrerServeur()
+annoncer()
+stop()
Client
+DataOutputStream dos;
+DataInputStream dis;
+Socket s;
+Joueur joueur;
+envoyerMessage()
+arreter()
Jeu
...
GestionnaireClient
+run()

C. Client


Client
-Serveur serveur
-ConfigurationClient config
-DataInputStream dis
-DataOutputStream dos
-AffichageStatus affichageStatus
-AffichageJeu affichageJeu
-Gson gson
-ArrayList<Joueur> joueurs
-poigneeDeMain()
AffichageStatus
AffichageJeu
Jeu
Serveur
+DataOutputStream dos;
+envoyer()
GestionnaireServeur
+run()