

GENIE LOGICIEL

Projet "WORDHUNT"

(la chasse aux mots)

W	O	E	H	N	P
A	R	D	K	U	T
G	C	N	I	O	N
B	E	F	L	E	Q
L	G	H	C	E	L
O	D	I	M	I	E

Mme D'AGOSTINO Eléonore

M. GHOZLANI Karim

M. KUNZMANN David

M. MINDER Valentin

M. NTAWURUHUNGA Paul

Table des matières

1. Introduction	2
2. Analyse.....	2
2.1. Règles du jeu.....	2
2.2. Partage des responsabilités entre le client et le serveur	3
2.3. Diagramme d'activité général.....	6
2.4. Cas d'utilisation.....	6
3. Conception du projet	11
3.1. Protocole d'échange entre le client et le serveur.....	11
3.2. Modèle de domaine	12
3.3. Base de données : Modèle conceptuel et relationnel	13
3.4. Interface utilisateur.....	15
4. Implémentation du projet.....	15
4.1. Technologies et langages utilisés.....	15
4.2. Problèmes rencontrés	15
5. Gestion du projet.....	18
5.1. Rôles des participants	18
5.2. Plan d'itérations initial	18
5.3. Suivi du projet.....	21
6. État des lieux	27
7. Autocritique	28
8. Conclusion.....	28

1. Introduction

Dans le cadre du projet de génie logiciel, nous avons eu l'occasion de réaliser une application ou un jeu de notre choix. Nous avons décidé d'implémenter un jeu de "chasse aux mots", WordHunt, sous forme d'une application android avec un serveur séparé. Ceci pour nous permettre d'être assez intéressés par notre projet pour aider à nous motiver à avoir un résultat qui nous fasse envie.

2. Analyse

2.1. Règles du jeu

But du jeu (simplifié)

Soit une grille de lettres de taille $n \times m$ cases. Dans le jeu "La chasse aux mots" (*wordhunt*), le but du jeu est de retrouver le maximum de mots de 3 lettres minimum, en passant d'une lettre à l'autre dans les 8 directions (haut, bas, droite, gauche et les quatre diagonales), en n'utilisant chaque lettre qu'une seule fois par mot. Parmi les extensions possibles, nous proposons les suivantes. Le temps pourrait être limité. Des compétitions multi-utilisateurs pourraient être organisées. Le comptage des points pourrait subir plusieurs variantes. La langue pourrait varier. Il serait possible de jouer "hors-ligne" pour s'entraîner.

Exemple de grille 3 x 3

E	H	R
Y	T	E
A	D	A

Dans la grille ci-dessus, nous pouvons par exemple identifier les mots suivants de la langue anglaise:

- 6 lettres: *thread*
- 5 lettres: *death, tread, ether*
- 4 lettres: *date, data, they, read*
- 3 lettres: *the, tea, day, red, ate, eat, tad, dye, thy, ada*

Variantes de langues

Il serait possible de développer des grilles dans les langues suivantes:

- Français
- Anglais
- les deux à la fois (dans la même instance de grille)
- ou encore en jargon informatique (dans l'exemple ci-avant, les mots tels que *thread*, *data*, *ada*, *read* seraient avantagés et/ou exclusifs).

Variantes de comptage des points

Les points pourraient être comptés

- Par mot (1 mot = 1 point)
 - en proportion linéaire de la longueur du mot (3 lettres = 3 points, etc.)
 - en proportion quadratique de la longueur du mot moins deux, c'est à dire: $\text{points} = (\text{longueur} - 2)^2$ (c'est-à-dire 3 lettres = 1 point, 4 lettres = 4 points, 5 lettres = 9 points, 6 lettres = 16 points, etc..)
 - avec un système de bonus à thème (p.ex. les mots reliés à l'informatique comptent triples)
- Afin de normaliser les scores, ceux-ci sont toujours exprimé en % des points totaux possibles sur une grille.

Variantes de compétition

Il serait possible d'avoir plusieurs modes :

- Mode "Entrainement" (avec ou sans limite de temps, sans enregistrement des points, les solutions sont disponibles à volonté)
- Mode "Compétition" (une grille est libérée par le serveur pour tous les joueurs en même temps, qui s'affrontent en temps-réel de manière synchronisée, selon les mêmes conditions)
- Mode "Challenge" (un joueur essaie de battre de manière désynchronisée des joueurs ayant précédemment joué sur la même grille et selon les mêmes conditions)
- A noter: les utilisateurs non-authentifiés (anonymes) ne peuvent accéder qu'au mode "Entrainement"
- A noter: hors-ligne, il n'est possible que de jouer en mode "Entrainement" (afin d'éviter qu'un utilisateur puisse disposer des conditions différentes, telles que du temps supplémentaire).

2.2. Partage des responsabilités entre le client et le serveur

Tâches du serveur

- Gestion du dictionnaire (différentes langues + mots spéciaux)
- Générateur aléatoire de grille selon les fréquences des lettres
- Vérificateur de fréquence et normalisateur (pré-validation)
- Testeur et solveur de grille (validation)
- Gestion de grilles déjà générée (p.ex. pour le mode "Challenge")
- Gestion des utilisateurs (authentification, scores, statistiques)
- Gestion des compétitions (et lutte contre la fraude)

Tâches du client (en mode utilisateur)

- Démarrage d'un jeu (avec choix du mode)
- Affichage de la grille
- Affichage des statistiques de la grille (nombre de mots à trouver/score maximal), du score en cours, du timer (chronomètre / minuteur).
- Prise en compte des mouvements utilisateurs (saisie tactile des mots).
- Affichages des scores précédents, des statistiques de l'utilisateur.
- Affichage des scores à la fin d'une compétition.
- Affichage des solutions d'un jeu terminé.
- Stockage de jeux futurs pour utilisation hors-ligne.
- Enregistrement (sign-on) et authentification (sign-in) des utilisateurs.

Tâches du client (en mode administrateur)

- toutes les tâches du client mode utilisateur, et en plus,
- démarrage manuel de compétitions inter-joueurs selon les paramètres choisis (déclenchement du mode "Compétition")
- (*optionnellement*: jeu en mode "triche" de façon contrôlée afin de mettre à l'épreuve nos stratégies de défense anti-triche).

Possibilités de tricherie et lutte contre la fraude

Problématique: côté client, on aimerait pouvoir valider les mots trouvés au fur-et-à-mesure du jeu (il faut donc connaître dès le début si le mot "jeu" est un mot valide ou non), mais afin d'éviter toute tentative de triche le serveur ne peut pas envoyer directement les solutions au client (à savoir les mots à trouver dans la grille).

Solution: les solutions sont envoyées sous forme "hachée" avec un sel différent pour chaque grille (en effet, sans sel, les mots du dictionnaire seraient trop faciles à trouver par inversion du hash). Ainsi, on envoie avec la grille la liste des hash des mots à trouver ainsi que le sel utilisé. Lorsque le client essaie un mot, on calcule le hash de ce mot avec le sel. S'il est présent dans la liste, alors le client sait que le mot est valide. Pour valider ses solutions côté serveur, le client devra toutefois envoyer les mots "en clair" afin d'empêcher toute tricherie. Un client qui enverrait des mots qui n'ont pas été validés par le processus décrit se verrait bannir l'accès, puisqu'il tente de contrer la sécurité mise en place.

Problématique: le client peut toujours utiliser un calculateur informatisé pour tester les solutions, ou pour générer toutes les solutions possibles et les tester. La seule solution possible serait alors de faire valider par le serveur toute tentative de mot (et de "détecter" sans précision absolue les utilisateurs qui abusent de ce système avec trop "d'erreurs", de mots invalides). Là encore, un client habile pourrait user de son propre dictionnaire afin de tester que les mots qu'il sait "probablement valides" (si les dictionnaires sont similaires). Une dernière approche pourrait être d'accepter une fréquence maximale de test de mot (par exemple 10 par seconde), qu'un humain ne peut raisonnablement pas dépasser. Toutefois encore, un client habile pourrait se débrouiller pour respecter cette fréquence avec son programme de brute-force.

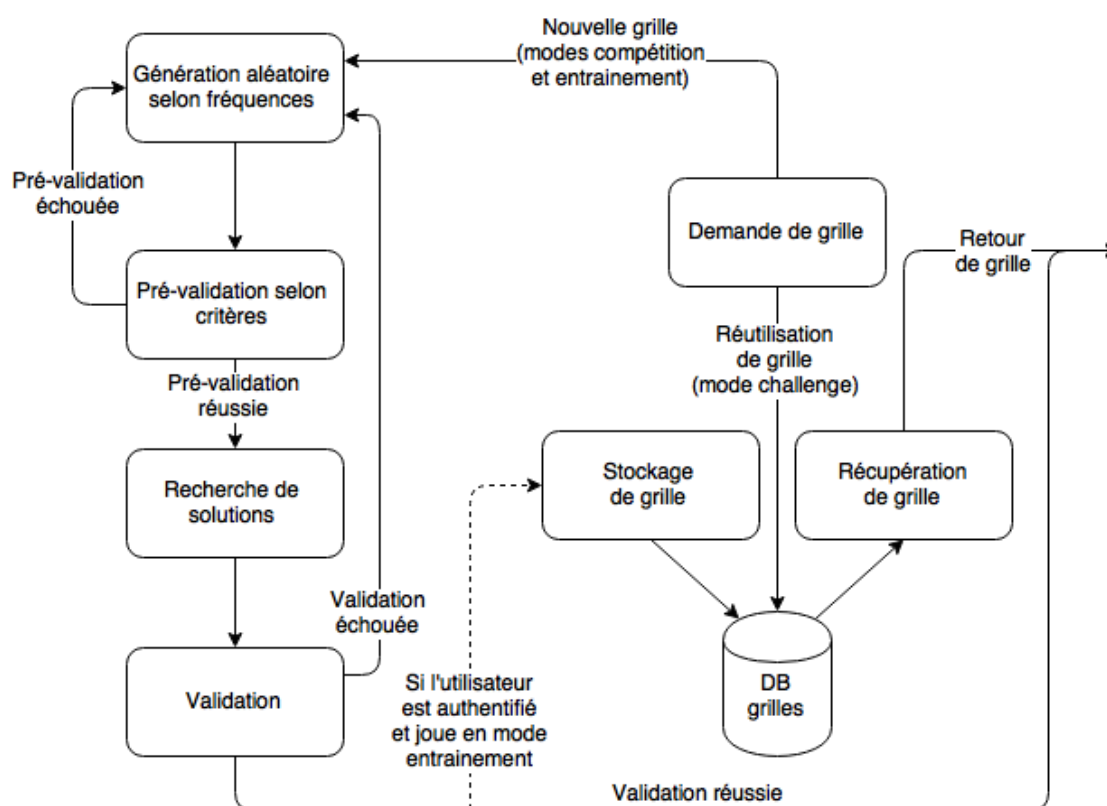
Trade-off (compromis): ce logiciel étant à pur but ludique, et ses membres exclusivement de filière "informatique-logiciel" (IL), nous avons décidé de laisser de côté les considérations trop techniques de sécurité. Nous n'implémenterons donc que la plus simple (et la moins risquée en terme d'erreurs) des méthodes de protection, à savoir le hachage avec sel, en étant conscients des risques possibles de triche. Les solutions plus élaborées sont laissées à réflexion pour nos collègues de "télécommunications - sécurité" (TS).

🚦 Précisions sur la génération des grilles

La génération des grilles se fait en plusieurs étapes côté serveur.

1. Génération: Une grille est générée aléatoirement, avec pour chaque case la probabilité qu'une lettre apparaisse égale à sa fréquence dans la langue désirée
2. Pré-validation: globalement, les probabilités d'apparition des lettres dans la grille doivent être comprises dans des fourchettes proches des probabilités standards. Par exemple, une grille sans "e" sera sûrement refusée, de même qu'une grille avec trois "z". De la même façon s'il est censé avoir 40% de voyelles, seule une grille avec 25 à 55% de voyelles sera acceptée, sinon refusée. En cas de refus, retour en 1.
3. Validation: la grille est résolue et le serveur détermine tous les mots qu'il est possible de trouver. Ce nombre doit être assez grand.
4. Stockage de la grille et de ses solutions pour un usage futur (mode challenge et/ou économie de ressources: une même grille peut être envoyée à plusieurs utilisateurs, même en "entraînement").

Un schéma résumant cette marche à suivre est disponible.



Précisions sur l'algorithme de résolution d'une grille.

Plusieurs solutions s'offrent à nous:

- tester toutes les combinaisons possibles dans la grille et vérifier si elles sont présentes dans le dictionnaire
- tester tous les mots du dictionnaire s'ils sont présents dans la grille
- utiliser un algorithme plus avancé de "Ternary Search Trie" tel qu'étudié en ASD2, ce qui demande une implémentation plus complexe.

Nous déterminerons à l'usage laquelle des solutions offre le ratio complexité-efficacité le plus adapté.

2.3. Diagramme d'activité général

Le diagramme d'activité ayant une taille conséquente, il a été fourni en annexe à la fin du rapport afin qu'il soit lisible.

2.4. Cas d'utilisation

Attention à la distinction client != acteur/utilisateur. L'acteur/utilisateur est l'humain qui va utiliser le programme. Le client est l'application utilisée par l'utilisateur, qui va se connecter à un serveur distant.

Acteurs:

- acteur principal: l'utilisateur "joueur". Celui-ci peut être de type de "anonyme" ou "authenticé". L'utilisateur anonyme peut créer un compte (enregistrement), se connecter (connexion) et jouer en mode "Entraînement" (cf scénarios appropriés). L'utilisateur authenticé peut se déconnecter et jouer dans tous les modes.
- acteur principal: l'utilisateur "administrateur". Celui-ci est nécessairement de type "authenticé".

Scénario de jeu "Entraînement"

1. L'utilisateur démarre l'App et clique sur "Entraînement"
 2. Le client demande une grille en mode "Entraînement"
 - 2.1 La grille est produite cf. cas d'utilisation *Génération de la grille*
 3. Le serveur renvoie la grille cf. cas d'utilisation *Téléchargement de la grille*
 4. Le serveur renvoie les solutions cf. cas d'utilisation *Résolution de la grille*
 5. Le client affiche la grille
 6. L'utilisateur joue (éventuellement en dépassant le time s'il le souhaite)
 7. L'utilisateur peut choisir d'afficher les solutions en tout temps
- A noter que l'utilisateur peut être connecté mais cela n'a aucune importance.*

Scénario de "demande de grille hors-ligne" et scénario de "jeu hors-

ligne": Ensemble, sont identiques au mode de jeu "Entraînement", mais en deux étapes distinctes.

🚩 Scénario de jeu "Challenge"

1. L'utilisateur démarre l'App et clique sur connexion, entre ses identifiants cf. cas d'utilisation *Authentification*
2. Le client demande une grille en mode "Challenge" cf. cas d'utilisation *Téléchargement de la grille*
3. Le serveur renvoie une grille cf. cas d'utilisation *Téléchargement de la grille*
4. Le serveur renvoie les statistiques liées à la grille (des précédents joueurs)
5. Le client affiche la grille et les statistiques
6. L'utilisateur joue, jusqu'à la fin du "timer" (minuteur).
7. Le client soumet les solutions (avant la fin du timeout du server)
8. Le serveur valide les solutions proposées, enregistre le score et renvoie la totalité des solutions cf. cas d'utilisation *Vérification des solutions*
9. Le client affiche les solutions ainsi que le score relatif aux précédents joueurs.

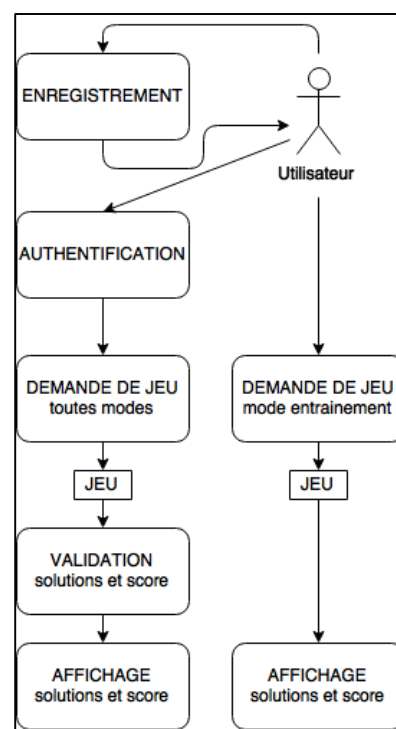
🚩 Scénario de jeu en mode "Compétition"

Identique au mode "Challenge" sauf qu'il n'y pas de statistiques précédentes, et que les statistiques finales sont en rapport aux joueurs simultanés.

A noter que ce mode n'est disponible que lorsqu'il est activé (automatiquement ou par action de l'administrateur).

Une ébauche de diagramme complet des cas d'utilisation est ci-contre. Celui-ci est appelé à être complété par des scénarios alternatifs (d'échec p.ex.). Pour simplifier le rapport intermédiaire, seule une version est complète, les autres en sont dérivées (le rapport final comprendra séparément les différents échanges possibles).

Attention, le nom des cas d'utilisation, des échanges et des modèles de domaine doivent être identiques. Nous avons tenté de respecter au mieux, une vérification complète pour le rapport final sera nécessaire.



🚩 Cas d'utilisation « Paramétrage d'une partie compétitive »

1. L'administrateur entre les données nécessaires pour une partie compétitive (temps, nombre de grilles à jouer, type de comptabilisation des points)
2. L'administrateur débute la partie
3. Une fois la partie terminée les points sont comptabilisés cf. cas d'utilisation *Vérification des solutions*

🚩 Cas d'utilisation « Vérification des solutions »

1. Une fois une partie terminée le serveur compare les solutions proposées par le client aux solutions générées cf. cas d'utilisation *Résolution de la grille*
 - 1.1. Les points sont comptabilisés selon le mode de comptabilisation des points cf. cas d'utilisation *Paramétrage d'une partie compétitive* point 1.

Cas d'utilisation « Validation de la grille »

1. Le serveur se base sur les solutions de la grille afin de déterminer si elle est jouable (par exemple nombre de mots possible) cf. cas d'utilisation *Résolution de la grille*

Cas d'utilisation « Résolution de la grille »

1. Le serveur crée une liste de tous les mots valides dans une grille pré validée, à l'aide du dictionnaire cf. cas d'utilisation *Pré validation de la grille*

Cas d'utilisation « Pré validation de la grille »

1. Vérification de la validité d'une grille générée (cf. cas d'utilisation *Génération de la grille*) en tenant compte du rapport voyelles/consonnes, du nombre d'apparition d'une lettre (par exemple une grille ne comprenant que des A et des Z ne pourra être pré validée)

Cas d'utilisation « Génération de la grille »

1. Une grille est créée à partir d'une génération aléatoire de lettres

Cas d'utilisation « Stockage de la grille »

1. Une grille validée est stockée dans la base de données pour usage ultérieur

Cas d'utilisation « Authentification »

1. Un joueur enregistré dans la base de données (cf. cas d'utilisation *Enregistrement*), peut s'authentifier en cliquant sur connexion.
2. Le joueur entre ses identifiants.
 - 2.1. Vérification que l'utilisateur soit bien enregistré cf. cas d'utilisation *Vérification que l'utilisateur ne soit pas déjà enregistré*
3. Le serveur répond par une approbation cf. cas d'utilisation *Approbation de la connexion*, ou par un échec cf. cas d'utilisation *Rejet de la connexion*

Cas d'utilisation « Approbation de la connexion »

1. Comme les identifiants entrés par l'utilisateur correspondent aux identifiants de la base de données, le serveur renvoie un token au client spécifiant que ce dernier est authentifié cf. cas d'utilisation *Délivraison du token*

Cas d'utilisation « Rejet de la connexion »

1. Le serveur demande à nouveau les identifiants du joueur cf. cas d'utilisation *Authentification*

Cas d'utilisation « Délivraison du token »

1. Un token est délivré à l'utilisateur spécifiant que ce dernier est authentifié

Cas d'utilisation « Téléchargement de la grille »

1. Envoi au client d'une grille stockée cf. cas d'utilisation *Stockage de la grille* si possible. Dans le cas contraire une grille sera générée cf. cas d'utilisation *Génération de la grille*

Cas d'utilisation « Enregistrement »

1. Si le client n'est pas déjà enregistré cf. cas d'utilisation *Vérification que l'utilisateur ne soit pas déjà enregistré*, il est ajouté dans la base de donnée, cf. *Création de l'utilisateur*

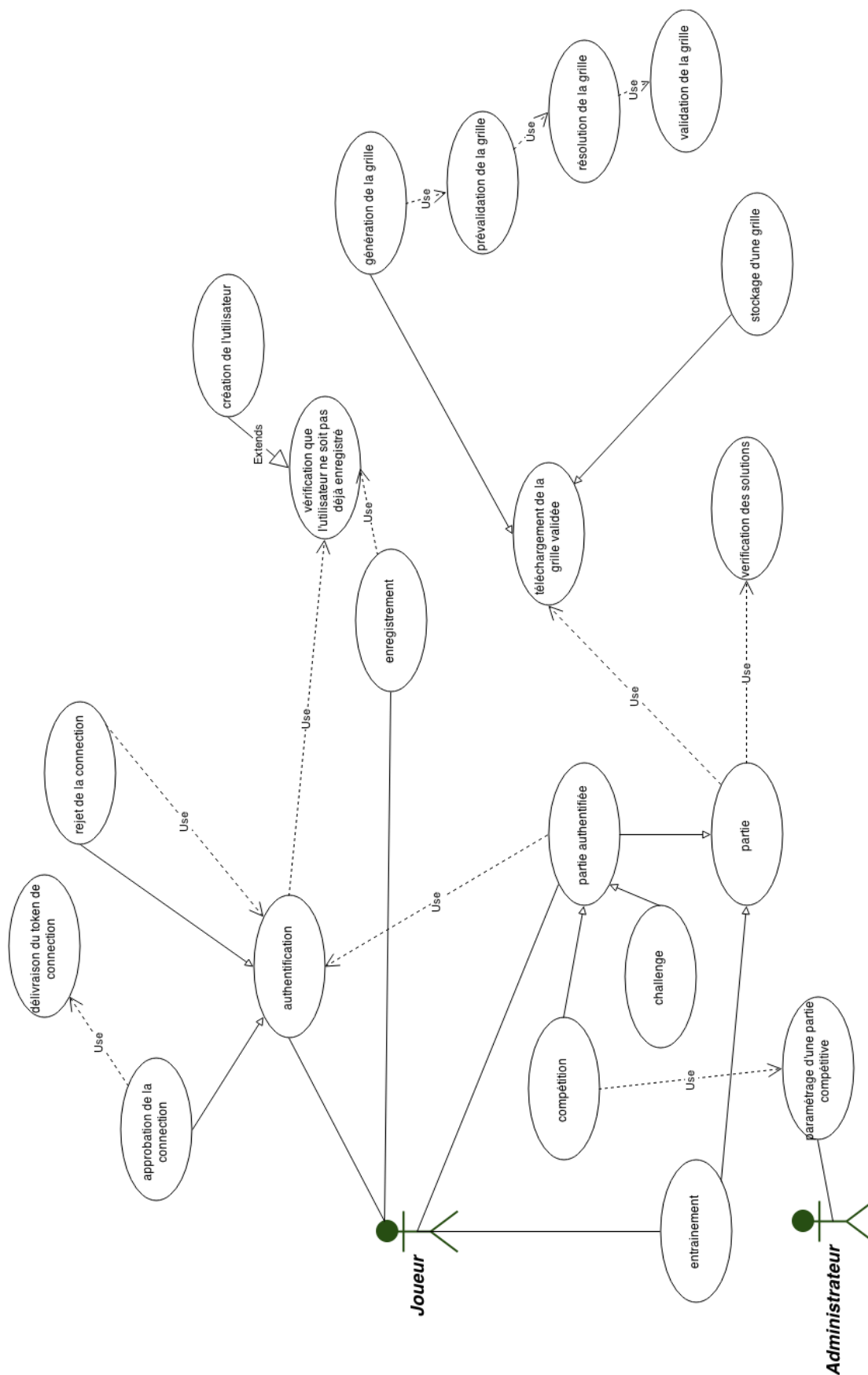
Cas d'utilisation « Vérification que l'utilisateur ne soit pas déjà enregistré »

1. Vérification si l'adresse e-mail n'est pas déjà présente dans la base de données

Cas d'utilisation « Création de l'utilisateur »

1. Les informations de l'utilisateur sont enregistrées dans la base de données

Diagramme Cas d'utilisation



3. Conception du projet

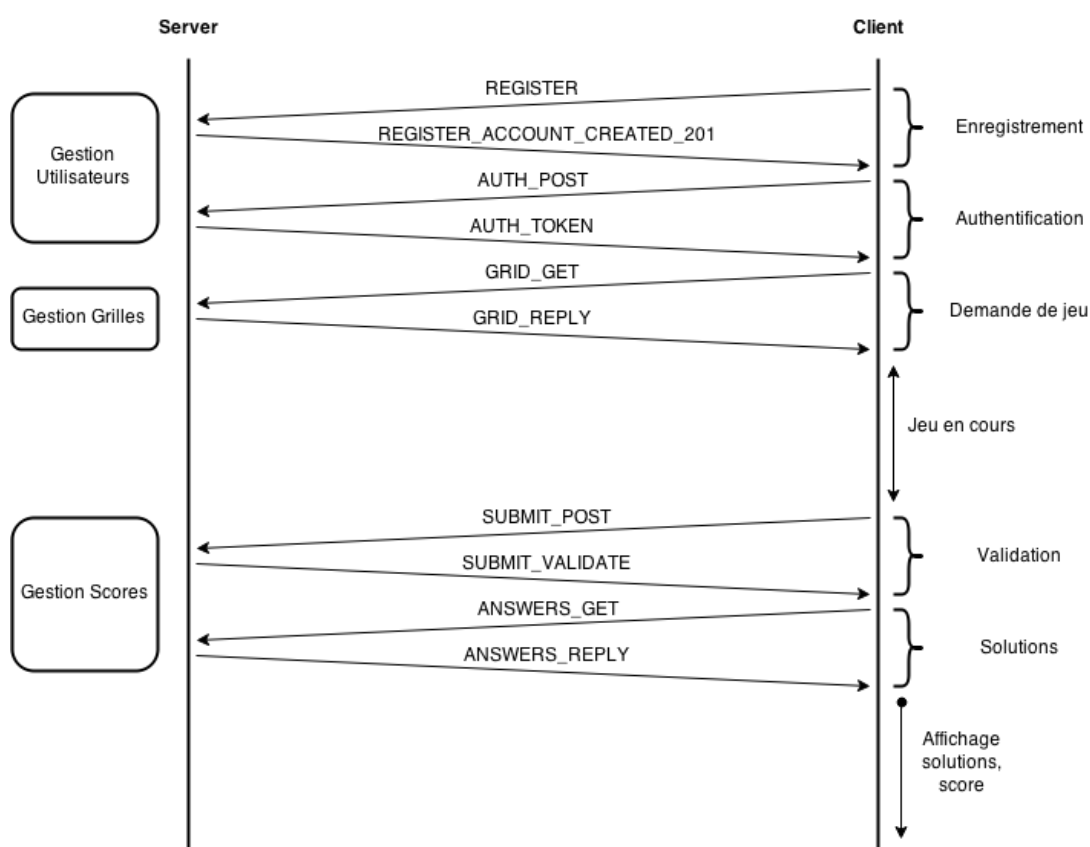
3.1. Protocole d'échange entre le client et le serveur

Les données échangées sont sérialisées en JSON. Les deux plates-formes sont en java, ce qui simplifie la mise en œuvre.

Voir également le schéma en annexe pour les différents échanges (diagramme de séquence, comprenant les différents échanges client-serveur pour le cas en mode challenge ou compétition). Le cas entraînement est basé sur le celui-là, plus simple.

Chaque message est composé d'un en-tête (header) et d'un contenu (content) correspondant à cette en-tête. On voit sur le schéma le "type" de message échangé (header). Le contenu est défini en détail dans les classes s'y rapportant.

Les messages d'échecs ne sont pas représentés. Il peut s'agir des suivants: bad request (400 - mauvais utilisation côté client), server error (500 - problème côté server), bad credentials (les données d'authentification ne sont pas reconnues), authentication required ().

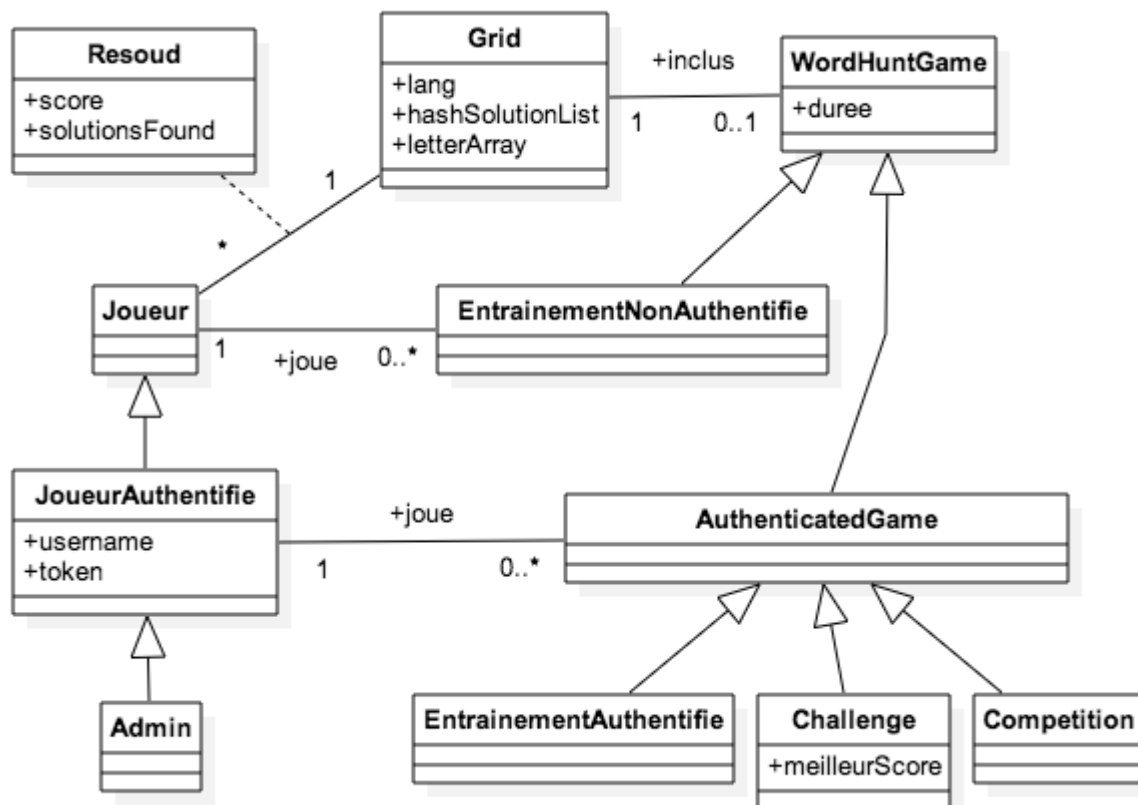


3.2. Modèle de domaine

Le modèle de domaine du client:

Un jeu WordHunt peut être selon le choix du joueur en mode entraînement, compétition, ou challenge. Chaque mode comporte leur propre caractéristique telles que décrites dans section 1 de ce document. Pour chaque résolution de la grille un score est assigné à la grille et au joueur.

L'entité admin a pour vocation de gérer certains modes de jeu tels que la durée d'une partie compétitive.

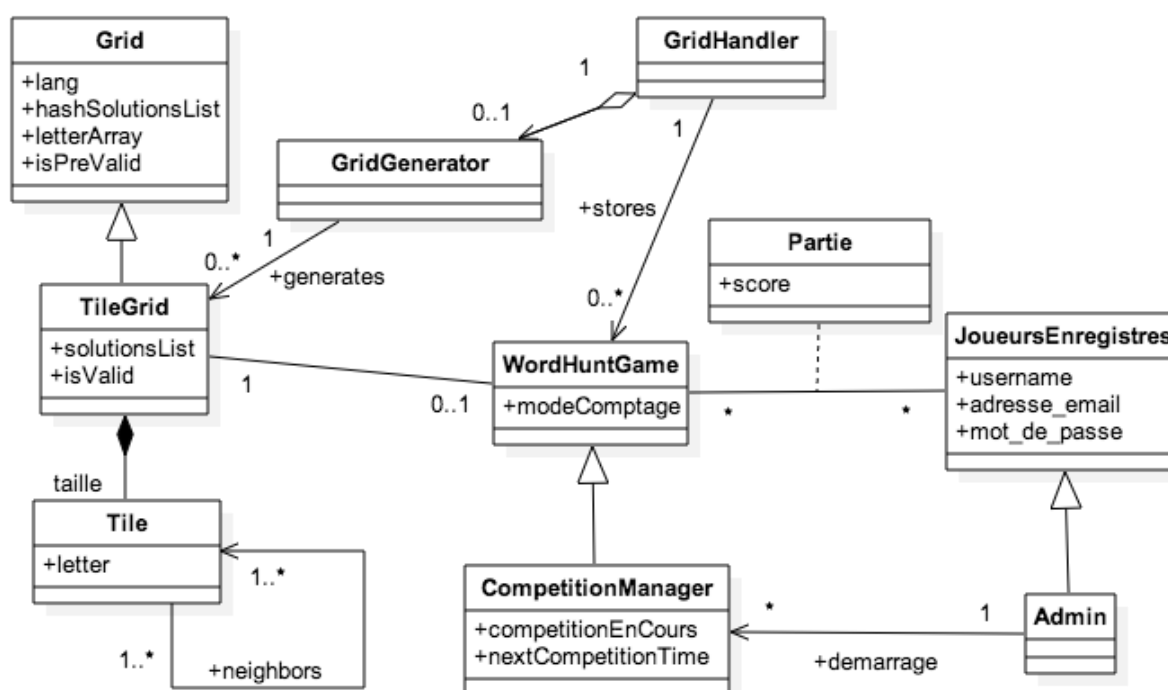


🚦 Le modèle de domaine du serveur :

Du point de vue du serveur, une grille est composée de lettres. Après la pré-validation de cette grille (vérification si le rapport voyelle/consonne est intéressant, nombre de répétition d'une lettre dans la grille), cette dernière devient une grille pré validée, puis une grille validée si le nombre de mot à trouver dans cette grille est suffisant pour être joué. Il faut donc, afin de valider une grille, la résoudre. Cette résolution est composée d'une liste de mots.

Du côté du joueur nous avons différents cas où l'utilisateur peut être enregistré dans la base de données et donc il peut profiter du mode compétitif du jeu.

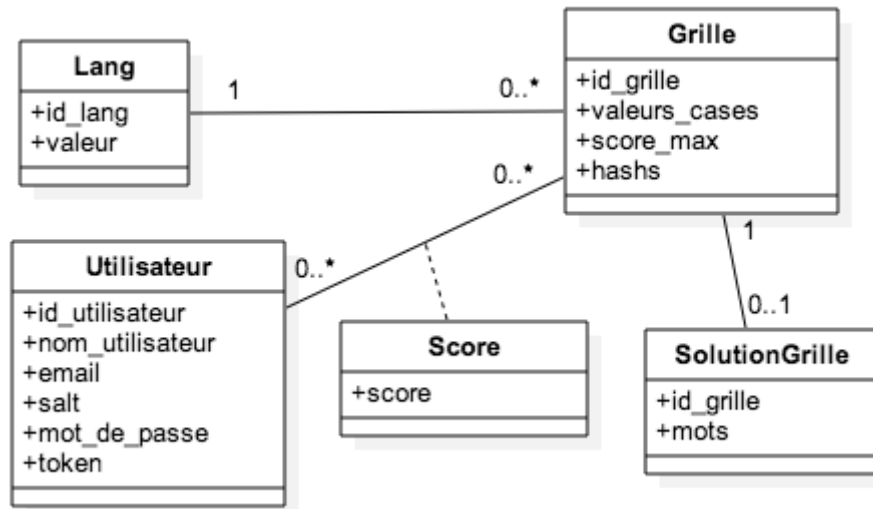
Et comme vu plus haut l'admin peut par exemple démarrer une partie compétitive (les autres types de jeu ne sont pas significatifs ici - donc pas représentés).



3.3. Base de données : Modèle conceptuel et relationnel

La base de données doit stocker toutes les informations persistantes liées à l'application (les utilisateurs, les grilles, les solutions, les scores, etc.).

Schéma Entité - Associations.



🚦 Schéma relationnel:

Lang(`id_lang`, `valeur`)

Grille(`id_grille`, `id_lang`, `valeurs_cases`, `score_max`, `hashs`)

clé étrangère `id_lang` référence Lang.`id_lang`

Utilisateur (`id_utilisateur`, `nom_utilisateur`, `email`, `salt`, `mot_de_passe`, `token`)

unique (`nom_utilisateur`)

unique (`email`)

Score(`id_grille`, `id_utilisateur`, `score`)

clé étrangère `id_grille` référence Grille.`id_grille`

clé étrangère `id_utilisateur` référence Utilisateur.`id_utilisateur`

SolutionGrille (`id_grille`, `mots`)

clé étrangère `id_grille` référence Grille.`id_grille`

3.4. Interface utilisateur

L'interface utilisateur d'ouverture devrait proposer les menus suivants: Entraînement, Charger des parties Hors-Ligne, Connexion (login/signa-in), Enregistrement (sign-on), ainsi qu'afficher le nombre de parties hors-ligne encore disponibles.

L'interface d'enregistrement possèdera les champs suivants: nom d'utilisateur, email, mot de passe.

L'interface de connexion possèdera les champs suivants: nom d'utilisateur, mot de passe, une check-box "rester connecté". Une fois connecté, un bouton doit permettre la déconnexion.

Pour un utilisateur connecté, les menus suivants sont disponibles: Entraînement, Charger des parties Hors-Ligne, Challenge, Compétition, mes scores.

Dans le cas d'entraînement, le joueur peut choisir ses paramètres de jeu (la langue, le mode de comptage de points, contre la montre ou non), alors qu'en mode challenge et compétition, il est forcé d'utiliser ceux qui lui sont proposés par la partie disponible respectivement en cours (afin d'éviter la multiplication des instances de parties).

L'interface de jeu en elle-même possède une grille où chaque case est cliquable (plus exactement "slidable" pour sélectionner tactilement un mot). Une fois que l'utilisateur a commencé à "slider", seules les cases contiguës sont slidables à leur tour. Lorsque l'utilisateur relâche, le mot choisi est testé. L'interface comprend en outre le timer, le score actuel, le score total possible sur la partie ainsi que le nombre de mot de chaque longueur à trouver.

4. Implémentation du projet

4.1. Technologies et langages utilisés

Pour l'implémentation du serveur nous avons utilisé exclusivement le langage Java sans framework particulier. Côté client, nous avons par contre utilisé Android Studio, afin de pouvoir gérer facilement l'interface graphique.

4.2. Problèmes rencontrés

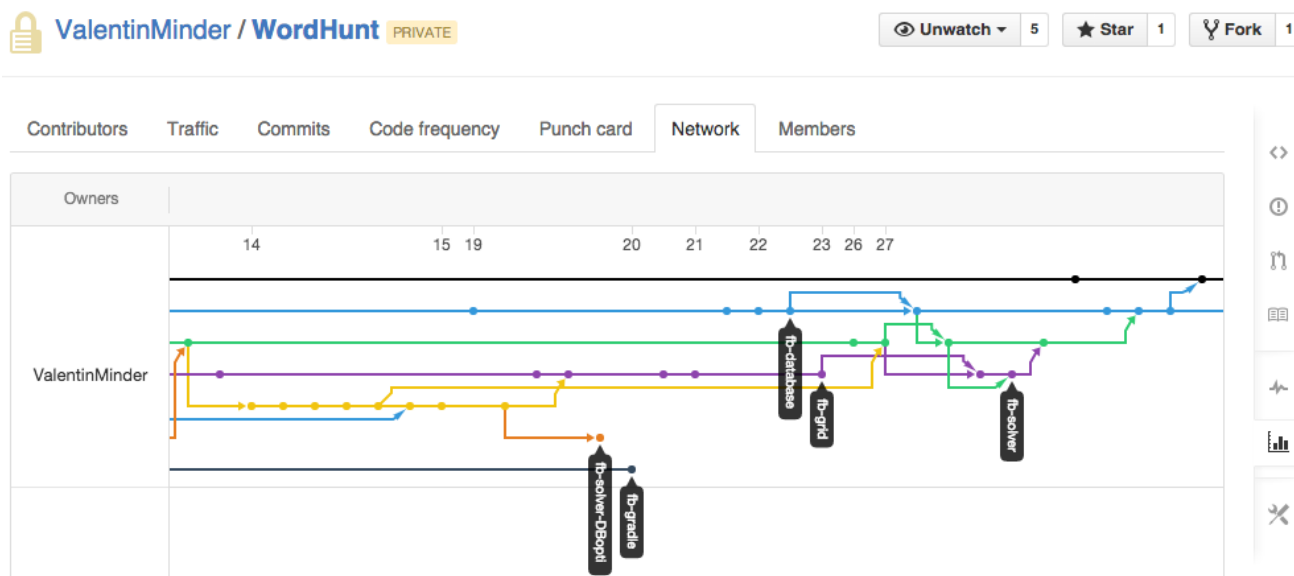
Retard durant l'itération 4 :

Nous savions que l'itération 4 était une itération charnière de notre projet car elle constitue vraiment le cœur de la logique de notre application, à savoir la résolution des grilles. Nous avons d'ailleurs prévu une charge de travail très élevée, ce qui a été le cas. Après avoir implémenté la logique, nous avons décidé de reporter les petites tâches restantes à l'itération suivante. Ces tâches ont bien été faites à l'itération suivante, sans créer un nouveau retard.

Synchronisation de Github :

Nous nous sommes rendus compte que la synchronisation du travail pouvait vite s'avérer être une tâche bloquante fastidieuse. La répartition était nette entre les membres des groupes, typiquement une partie client et une partie serveur. Chaque partie était implémentable de façon bien distincte, jusqu'au moment où nous étions obligé de rassembler toutes les pièces. Nous ne pensions pas que

cette tâche allait nous prendre autant de temps. Par conséquent, nous n'avons pas pu avancer sur l'itération courante, qu'il fallait alors effectuer exclusivement durant notre temps libre.



DB & TreeSet :

Remarques sur la structure de stockage des mots.

Lors de la conception du projet, nous avions dans l'idée de stocker les mots dans la base de données (database) SQL. Lors de la résolution d'une grille, l'idée était de générer tous les mots possibles dans la grille, puis de vérifier leur présence dans la base de données. Nous avons remarqué par la suite que dans une grille 4x4, nous pouvons trouver jusqu'à 1.6 millions de mots de longueur 3 à 10, s'ils sont tous générés (voir ci-dessous). Il serait donc utile d'arrêter de générer des mots plus longs dès qu'une sous-chaine (p.ex. GHC) n'est plus préfixe d'un mot existant. Toutefois, une telle approche nécessiterait un aller-retour constant entre la structure de mots et la structure de résolution de la grille. Toutefois, une seule requête vers la base prends plusieurs centaines de millisecondes, donc avec de nombreux aller-retour vers la base de données, cette approche est totalement impossible avec une base de données.

Dans un deuxième temps, nous avons tenté d'inverser le problème, par une approche récupération de tous les mots existants puis tests de présence dans la grille, on a tous les mots du dictionnaire dont il faut tester la présence. Un test de présence consiste à chercher successivement la présence consécutive des lettres d'un mot à tester. Le petit Robert contient environ 60 000 mots. Le dictionnaire du jeu type scrabble contient TOUTES les déclinaisons et variations (pluriels, féminins, accords, conjugaisons, etc.) donc plus de 328'000 entrées actuellement. Cette approche inversée, bien meilleure que la première, ne s'est pas avérée efficace (il faut tester chaque mot du dictionnaire, soit 328000 tests).

Il a donc été décidé de changer encore d'approche. Depuis chaque case, on visite tous les voisins. A chaque lettre ajoutée, on teste si la chaine générée jusque là est un mot. On continue de visiter les voisins de cette case à condition que la chaine générée soit une sous-chaine d'un autre mot, c'est-à-dire qu'on ait encore une possibilité de trouver un mot (grâce à la recherche de la "clé supérieure"). Par exemple, si on a la chaine "ABC", il existe le mot "ABCES", donc on continue à chercher dans les

voisins de C. Par contre, avec la chaîne "GDF", il n'y a pas de mot qui commence par GDF, donc on s'arrête. Si on effectue le premier test de continuation à 3 lettres, on n'a généré que 408 "mots", et seuls les intéressants seront continués. Au final, on aura environ un nombre de tests égal au nombre de mots trouvés fois leur longueur, soit environ 500 mots fois longueur 10 au max, soit au max 5000 tests, ce qui est nettement mieux que précédemment.

En java, il existe une structure qui permet l'accès rapides à la clé courante, et à la clé supérieure. Nous avons utilisé la structure TreeSet, qui est un arbre de recherche binaire (left-leaning red-black tree), qui nous permet de garantir d'opération de "vérification de clé" et "clé supérieure" en une complexité $O(\log n)$. Sachant que le nombre de mots en langue française est de l'ordre de 3280'000, la complexité $\ln(n) = 12.7$, soit 13x le temps constant, ce qui est très acceptable. Grâce aux nombreuses optimisations et utilisation d'un "TreeSet", une grille est solutionnée en 1 à 3 ms. Une grille valide est obtenue en moins de 5 ms. La structure a généré aux chargements du dictionnaire depuis un fichier texte (environ 4MB) prend environ 100 ms et occupe 10MB de RAM, ce qui reste très raisonnable.

Note: dans une grille 4x4 sans réutilisation, on trouve des mots de longueur

L - présent - cumulatif		
1 -	16 -	16
2 -	84 -	100
3 -	408 -	508
4 -	1764 -	2272
5 -	6712 -	8984
6 -	22672 -	31656
7 -	68272 -	99928
8 -	183472 -	283400
9 -	436984 -	720384
10 -	905776 -	1626160
11 -	1594648 -	3220808
12 -	2310264 -	5531072
13 -	2644520 -	8175592
14 -	2250192 -	10425784
15 -	1260672 -	11686456
16 -	343184 -	12029640

Problème Android Studio VS Eclipse ADT :

Android Studio et Eclipse ADT utilisent tous les deux un émulateur pour appareil Android. Cependant, la structure globale des projets est radicalement différente. En raison des dépendances (le project «

shared » est partagé entre le projet « client » et le projet « server ») il n'était pas possible de travailler en parallèle sur les deux projets. Ceci nous a empêchés de travailler en même temps sur le serveur et le client. Nous devons à chaque fois atteindre un état terminal dans notre code avant de le partager avec l'autre partie, ce qui a pour principale conséquence un temps de développement plus lent.

5. Gestion du projet

5.1. Rôles des participants

Rôle	Personnes
Chef de projet	Valentin (suppléant David)
Représentant des utilisateurs	Karim (suppléant Eléonore)
Analyste	David (suppléant Karim)
Architecte	Eléonore (suppléant Valentin)
Programmeur	tous
Responsable tests	Paul (suppléant David)
Responsable config	collectivement

5.2. Plan d'itérations initial

La durée d'une itération est systématiquement d'une semaine. Le mini-projet est lancé le 22 avril 2015 (semaine académique #8, semaine civile #17) quand **commence l'itération 1**.

Le mini-projet est à rendre et présenter le 16 juin 2015 (semaine académique #16, semaine civile #25) où **se termine l'itération 8**.

Evaluation du temps consacré.

Non compté bilans et team linking (répartition, séances de synchronisation, etc.).

Faible: moins de 3h/personne (aka moins de 15h)

Moyen: de 3 à 5h par personne (aka de 15 à 24h)

Elevé: plus de 5h/personne (aka plus de 25h)

Nous listons ci-dessous le plan d'itérations initial :

1. 22.04 - 28.04 : effort global prévu: faible.

- Installation des outils de développement (android-studio, adb, git, MySQL, etc)
 - Gestion / Développement de l'infrastructure
 - Réalisé en fin d'itération.
- Le client et le serveur peuvent se pinger mutuellement
 - Développement de l'infrastructure
 - Réalisé en fin d'itération.
- Schéma de l'interface utilisateur
 - Gestion
 - Réalisé en fin d'itération.
- Mise à jour du rapport et des itérations en fonctions de remarques ELS/Greppin
 - Gestion
 - Réalisé en fin d'itération.

2. 29.04 - 05.05 : effort global prévu: élevé

- Le serveur peut générer une grille aléatoire (sans critères de validation ni réponses)
 - Développement des fonctionnalités
 1. Cas d'utilisation: "génération de la grille"
 2. Réalisation partielle
 3. Développement à compléter
 - Réalisé en fin d'itération.
- La base de données contient un premier dictionnaire utilisable, contenant les fréquences de lettres associées
 - Développement de l'infrastructure
 - Schéma relationnel, construction de la DB
 - Parsing et insertion d'un dictionnaire
 - Récupération des fréquences de lettres
 - Réalisé en fin d'itération.
- Définition de la première version des protocoles d'échanges, incluant le contenu des JSON échangés ainsi qu'un diagramme de séquence
 - Gestion / Développement de l'infrastructure
 - Le protocole est défini avec un diagramme de séquence complet Le contenu de chaque message est prédéfini dans un fichier texte, qui indique quel sera le contenu probable des messages, à confirmer lors de leur implémentation/utilisation
 - Sérialisation / désérialisation des messages, caractérisés par un header et un contenu en json
 - Réalisé en fin d'itération

3. 06.05 - 12.05 : effort global prévu: très élevé.

- **On peut s'attendre à ne pas avoir de version démontrable à cette étape car la plupart des réalisations sont complétées à l'itération 4.**
- Les fonctionnalités de base du client sont implémentées:
 - On peut recevoir une grille depuis le serveur
 - On peut l'afficher
 - On peut saisir des solutions dessus.
- Les solutions ne sont pas envoyées au serveur, le client calcule les points.
 - Développement des fonctionnalités
 1. Cas d'utilisation: "partie"
 2. Réalisation partielle
 3. Développement à compléter
- L'algorithme de validation des grilles (côté serveur) est partiellement implémenté, en parallèle avec le solveur de grille. Il sera complété à l'itération suivante
 - Développement des fonctionnalités
 1. Cas d'utilisation: "pré validation de la grille", "résolution de la grille", "validation de la grille"
 2. Réalisation partielle
 3. Développement à compléter

4. 13.05 - 19.05 effort global prévu : très élevé

- Le serveur peut valider des grilles avant de les envoyer au client avec les solutions hashées
 - Développement des fonctionnalités
 1. Cas d'utilisation: "pré validation de la grille", "résolution de la grille", "validation de la grille"
 2. Réalisation entière
 3. Développement qui sera amélioré au fil des prochaines itérations si besoin
- Le client peut envoyer les solutions à sa grille au serveur
 - Développement des fonctionnalités
 1. Cas d'utilisation: "challenge", "compétition"
 2. Réalisation partielle
 3. Développement définitif
- Le serveur peut vérifier si les solutions envoyées par le client sont correctes
 - Développement des fonctionnalités
 1. Cas d'utilisation: "vérification des solutions"
 2. Réalisation entière
 3. Développement définitif

5. 20.05 - 26.05 effort global prévu : moyen

- Le système de gestion des utilisateurs est implémenté côté client:
- On peut enregistrer un nouveau compte depuis le client
- On peut se connecter ou se déconnecter depuis le client
 - Développement des fonctionnalités
 1. Cas d'utilisation: "authentification", "enregistrement"
 2. Réalisation partielle
 3. Développement définitif
- Le système de gestion des utilisateurs est implémenté côté serveur:
- On peut stocker les utilisateurs dans la BDD
- On peut enregistrer les statistiques de jeu des utilisateurs
 - Développement des fonctionnalités
 1. Cas d'utilisation: "authentification", "enregistrement", "rejet de la connexion", "approbation de la connexion", "délivraison du token de connexion", "vérification que l'utilisateur ne soit pas déjà enregistré", "création de l'utilisateur"
 2. Réalisation entière
 3. Développement définitif

6. 27.05 - 02.06 effort global prévu : élevé

- Un client peut rejoindre une compétition en cours
 - Développement des fonctionnalités
 1. Cas d'utilisation: "compétition"
 2. Réalisation entière
 3. Développement définitif
- Un nouvel utilisateur de type admin est implémenté, qui peut lancer les compétitions manuellement
 - Développement des fonctionnalités
 1. Cas d'utilisation: "paramétrage d'une partie compétitive"
 2. Réalisation entière
 3. Développement définitif

7. 03.06 - 09.06 effort global prévu : moyen

- Le client peut demander une grille précédente pour la jouer en mode challenge ou entraînement, à choix
 - Développement des fonctionnalités
 1. Cas d'utilisation: "partie", "challenge", "entraînement"
 2. Réalisation entière
 3. Développement définitif

8. 10.06 - 16.06 effort global prévu : faible

- Essai de l'application par des utilisateurs lambdas (alpha-testeurs), debugging en cas de découverte de bug.
- Marge de manœuvre si des itérations précédentes ont été plus compliquées que planifié
- Préparation du bilan final

5.3. Suivi du projet

1. 22.04 - 28.04 : effort global: Prévu : faible. Réalisé: moyen (total: 24h)

- Installation des outils de développement (android-studio, adb, git, MySQL, etc)
 - Gestion / Développement de l'infrastructure
 - Auteurs: tous (env. 2h/pers)
 - Réalisé en fin d'itération.
- Le client et le serveur peuvent se pinger mutuellement
 - Développement de l'infrastructure
 - Auteurs: Karim (server), Valentin (client), total 3h ensemble.
 - Réalisé en fin d'itération.
 - Démo: *On voit qu'un client android, en appuyant sur un bouton, est détecté par le serveur distant qui, en réponse, fait un echo en console des infos reçues de l'android.*
 - La démo est validée, mais toutefois pas complètement stable.
 - Le manque d'une itération pour gérer le "multi-client" est relevé.
- Schéma de l'interface utilisateur
 - Gestion
 - Auteur: Paul (3h)
 - Réalisé en fin d'itération.
 - Démo: démonstration interactive d'un mock-up est effectuée avec succès.
- Mise à jour du rapport et des itérations en fonctions de remarques ELS/Greppin
 - Gestion
 - Auteurs: Eleonore et David (3h ensemble)
 - Réalisé en fin d'itération.
- Bilan global: Tout a été *réalisé comme prévu, pas de replanification à opérer*

2. 29.04 - 05.05 : effort global: élevé. Réalisé: moyen (16h).

- Correction rapport (David - 2h)
- Le serveur peut générer une grille aléatoire (sans critères de validation ni réponses)

- Développement des fonctionnalités
 1. Cas d'utilisation: "génération de la grille"
 2. Réalisation partielle
 3. Développement à compléter
 - Le serveur génère des grilles aléatoires en suivant les fréquences de lettre (Karim - 2h)
 - Cette partie n'est pas présentable aujourd'hui, pour cause de batterie manquante (Karim)
 - Réalisé en fin d'itération.
 - La base de données contient un premier dictionnaire utilisable, contenant les fréquences de lettres associées
 - Développement de l'infrastructure
 - Schéma relationnel, construction de la DB (Paul - 2h)
 - Parsing et insertion d'un dictionnaire (David - 2h) - Il est discuté de la pertinence d'un dictionnaire interne ou d'utiliser un service externe. Interne: problème de complétude. Externe: dépendance/accès. Partir d'un dictionnaire COMPLET avec toutes déclinaisons, etc.
 - Récupération des fréquences de lettres (Karim)
 - Réalisé en fin d'itération.
 - La première version des protocoles d'échanges est définie, incluant le contenu des JSON échangés ainsi qu'un diagramme de séquence
 - Gestion / Développement de l'infrastructure
 - Le protocole est défini avec un diagramme de séquence complet (Eleonore - 1h)
 - Le contenu de chaque message est prédéfini dans un fichier texte, qui indique quel sera le contenu probable des messages, à confirmer lors de leur implémentation/utilisation (Eleonore - 1h)
 - Sérialisation / désérialisation des messages, caractérisés par un header et un contenu en json (Valentin / Eleonore - 3h chacun)
 - Réalisé en fin d'itération
- Bilan global: Tout a été *réalisé comme prévu, pas de re planification à opérer*

3. 06.05 - 12.05 : effort global prévu: très élevé. Réalisé: TRES élevé (43h).

- **Début d'itération: on peut s'attendre à ne pas avoir de version démontrable à cette étape car la plupart des réalisations sont complétées à l'itération 4.**
- **Fin d'itération: les composantes développées dans leur première version sont démontrables séparément (pas intégrées/mergées dans le projet global).**
- Les fonctionnalités de base du client sont implémentées (Paul)
 - On peut recevoir une grille depuis le serveur (protocole/serial): ok.
 - On peut l'afficher sur le client: ok.
 - On peut saisir des solutions dessus (click): ok.
 1. Temps réalisé 2h (plus 5h apprentissage Android)
- Les solutions ne sont pas envoyées au serveur, le client calcule les points (avec continuité dictionnaire)
 - Développement des fonctionnalités: Cas d'utilisation: "partie", Réalisation partielle, Développement à compléter

- Réalisé par David: 8h
 1. Interface d'accès au dictionnaire : ok (validation d'un mot, récupération de tous les mots. A compléter par récupération de mot correspondant à une condition: lettres autorisées, sous-chaine de départ, ..)
 2. Algorithme de calcul des points: ok. (uniquement version simple: par le nombre de lettre, à compléter)
- Réalisé en fin d'itération (pas de démo possible, absence de David)
- L'algorithme de validation des grilles (côté serveur) est partiellement implémenté, en parallèle avec le solveur de grille. Il sera complété à l'itération suivante
 - Karim: grille, génération, pré-validation (8h)
 1. Algo de génération aléatoire de grille avec les fréquences de la langue française ok.
 2. Algo de pré-validation avec certaines conditions (par exemple les fréquences max de voyelles): ok.
 3. Tout est stocké sous forme de propriétés éditables.
 - Eleonore/Valentin: solveur exhaustif de grille (temps: 10h chacun)
 1. La grille est représentée comme un "graphe" composé de tuiles (Tiles) adjacentes.
 2. première version fonctionnelle (confrontation de la grille au dictionnaire) : tous les mots possibles dans la grille sont testés (DFS). Pour une grille 4x4, jusqu'à 1.5 MILLIONS de chaines à tester.
 3. versions améliorée en cours de développement (confrontation du dictionnaire à la grille, ainsi qu'optimisation des requêtes et des sets de calculs, p.ex. avec les sous-chaine de départ, les lettres composantes/interdites)
 - Développement des fonctionnalités
 1. Cas d'utilisation: "pré validation de la grille", "résolution de la grille", "validation de la grille"
 2. Réalisation partielle
 3. Développement à compléter
 - Réalisé en fin d'itération

Bilan global: Tout a été *réalisé comme prévu, pas de replanification à opérer*

4. 13.05 - 19.05 effort global prévu : très élevé Réalisé: élevé (23h)

- Cette itération comprenait 2 jours de congé. L'équipe était très fatiguée par les nombreuses échéances récentes dans les divers cours. Le chef a donc consciemment choisi de "négliger" les aspects moins importants de cette itération afin de conserver les ressources humaines disponibles et "en forme" pour la suite.
- Pas vraiment de replanification nécessaire.
 - La plus grosse partie du travail, et surtout la plus compliquée, a été terminée par toute l'équipe.
 - a. Paul: migration client Eclipse > Android Studio (5h)
 - b. Eleonore: test d'optimisation en confrontant le dictionnaire à la grille (chercher la présence des mots dans la grille). 3h
 - c. David: test d'optimisation DB avec les lettres composantes 3h

- d. Valentin: test d'optimisation avec chargement dans une structure Collection Java appropriée. Le TreeSet fait l'affaire. 7h
 - e. La version de Valentin est la plus simple, très efficace et est donc retenue.
 - f. Karim: validation des grilles (condition sur le nombre de mots et présence de mots longs) 5h
 - Les parties manquantes sont plus faciles.
 - Le protocole pour envoyer des solutions, valider des solutions doit être implémenté, ce qui peut être intégré à l'itération suivante (développement protocole enregistrement / connexion)
 - Le développement du client à l'itération suivante inclura l'envoi des solutions et affichage du score.
- Le serveur peut valider des grilles avant de les envoyer au client avec les solutions hashées
- Développement des fonctionnalités
 1. Cas d'utilisation: "pré validation de la grille", "résolution de la grille", "validation de la grille"
 2. Réalisation entière
 3. Développement qui sera amélioré au fil des prochaines itérations si besoin
 - Réalisé grâce aux nombreuses optimisations et utilisation d'un "TreeSet" (arbre de recherche binaire, left-leaning red-black tree), garanti d'opération de "vérification de clé" et "clé supérieure" en $O(\log n)$.
 - Une grille est solutionnée en 1 à 3 ms. Une grille valide est obtenue en moins de 5 ms.
 - Réalisé en fin d'itération.
- Le client peut envoyer les solutions à sa grille au serveur
- Développement des fonctionnalités
 1. Cas d'utilisation: "challenge", "compétition"
 2. Réalisation partielle
 3. Développement définitif
 - Pas réalisé, voir remarque ci-dessus.
- Le serveur peut vérifier si les solutions envoyées par le client sont correctes
- Développement des fonctionnalités
 1. Cas d'utilisation: "vérification des solutions"
 2. Réalisation entière
 3. Développement définitif
 - Pas réalisé, voir remarque ci-dessus.

5. 20.05 - 26.05 effort global prévu : moyen . Réalisé: élevé (25h, 5h/personne)

Rattrapage

- **Iter4:** Le client peut envoyer les solutions à sa grille au serveur
 - **Protocole de renvoi solutions: Eleonore: ok.**
- **Itér4:** Le serveur peut vérifier si les solutions envoyées par le client sont correctes
 - **Vérification solution côté server + acceptation ou avertissement: Eleonore: ok**

- **Itér4:** envoyer les grilles au client avec les solutions hashées
 - **Karim: gérer l'envoi des solutions hachées avec la grille, et la stocker dans la db.**
- **Lors d'une demande de grille, les solutions hachées sont envoyées.**

Développement des fonctionnalités

- Cas d'utilisation: "authentification", "enregistrement", "rejet de la connexion", "approbation de la connexion", "délivraison du token de connexion", "vérification que l'utilisateur ne soit pas déjà enregistré", "création de l'utilisateur"
- Réalisation complet
- Développement définitif
- **Protocole: pour interaction users (register&login): Valentin: ok.**
- Le système de gestion des utilisateurs est implémenté côté client:
- On peut enregistrer un nouveau compte depuis le client
- On peut se connecter ou se déconnecter depuis le client
 - **Client android pour interaction users (register&login): Paul: ok.**
- Le système de gestion des utilisateurs est implémenté côté serveur:
- On peut stocker les utilisateurs dans la BDD
 - **Server database users: insertion enregistrement / check connexion / délivraison du token: David: ok**
- Le client submit ses solutions à la fin du temps et pour la validation.
 - **Le merge final du travail réalisé en parallélisme par l'équipe n'est pas démontrable cette semaine pour contrainte de temps. Toutes les parties individuelles sont montrées et fonctionnelles.**
- On peut enregistrer les statistiques de jeu des utilisateurs
 - **Cette étape, peu important pour la globalité du logiciel, est remise à plus tard, sans planification.**
 - Développement des fonctionnalités
 1. Cas d'utilisation: "stockage des statistiques"
 2. Réalisation entière
 3. Développement définitif

6. 27.05 - 02.06 effort global prévu : élevé. Réalisé: faible à moyen

- *Nota: cette semaine il a été donné congé à une partie de l'équipe (pour cause de surcharge de travail, PRO - projet de semestre, David et Paul). Le reste du travail a été réalisé par Eleonore et Valentin en équipe, Karim jouait le rôle du code review et du représentant client (commentaires, respect des scénarios)*
- Un client peut rejoindre une compétition en cours
 - Développement des fonctionnalités
 1. Cas d'utilisation: "compétition"
 2. Réalisation entière

- 3. Développement définitif
 - Il y a 3 états: pas de compétition (not running), compétition prévue plus tard (scheduled), compétition en cours (running, joignable)
 - Réalisé en fin d'itération
- Un nouvel utilisateur de type admin est implémenté, qui peut lancer les compétitions manuellement
 - Développement des fonctionnalités
 1. Cas d'utilisation: "paramétrage d'une partie compétitive"
 2. Réalisation entière
 3. Développement définitif
 - Evidemment, seul l'admin peut lancer des compétition
 - Actuellement, seulement une compétition peut être prévue à la fois.
 - Réalisé en fin d'itération
- Le client Android n'est pas terminé pour ces cas d'utilisation.... mais un client simplifié java peut gérer correctement les interactions (programme de test)
- Une batterie de tests automatisés (JUnit) garantit le fonctionnement correct du programme jusque là.
 - Testé actuellement: Ping (vérification connexion), Grille d'entraînement, Login (connexion), Register (enregistrement), ScheduleCompet (planification d'une compétition)
 - A tester: recevoir une grille de compétition, vérification correcte des solutions.
- Bilan global: Tout a été *réalisé comme prévu, pas de re planification à opérer*

7. 03.06 - 09.06 effort global prévu : moyen. Réalisé: élevé (32h)

- Le client peut demander une grille précédente pour la jouer en mode challenge ou entraînement, à choix
 - Il est possible de demander une grille en mode entraînement en étant authentifié: une nouvelle grille sera créée et stockée pour futur usage (alors qu'en étant anonyme, on reçoit une grille temporaire).
 - Les participations des utilisateurs en mode challenge et entraînement sont toujours enregistrées.
 - Mode challenge: on demande une grille existante pour laquelle on n'a pas encore joué. Trois options:
 1. Selon un utilisateur (une grille que l'utilisateur X a déjà joué, mais pas moi - à condition qu'il y en ait une).
 2. Selon l'identifiant de la grille (la grille X - à condition que je ne l'ai pas déjà jouée)
 3. Sans critères: n'importe quelle grille ayant été jouée par quelqu'un d'autre (et pas par moi - à condition qu'elle existe)
-> très pratique pour tous car on ne connaît pas forcément les grilles, ni

Dans les 3 cas, il est possible qu'il ne soit pas possible de satisfaire à la demande, l'utilisateur reçoit un message d'erreur et est invité à s'entraîner d'abord et à revenir plus tard.

- Développement des fonctionnalités
 1. Cas d'utilisation: "partie", "challenge", "entraînement"

- 2. Réalisation entière
- 3. Développement définitif
- **Réalisé en fin d'itération (Valentin - 9h)**
- On peut enregistrer les statistiques de jeu des utilisateurs
 - **Ré agendé: les statistiques de jeu sont stockées en db (Valentin, 2h).**
 - Modification complète de la structure de la db: il n'y a plus de "mot" ni de "dictionnaire" -> chaque grille stocke ses solutions dans une table séparée, il y a une association grille-utilisateur pour les scores.
 - Développement des fonctionnalités
 - 1. Cas d'utilisation: "stockage des statistiques"
 - 2. Réalisation entière
 - 3. Développement définitif
 - **Réalisé en fin d'itération.**
- Finir la GUI sur Android :
 - David (6h): enregistrement/connexion/déconnexion/stockage du token
 - Paul (15h): swipe pour saisir les solutions, compteur de temps, calcul du score, entraînement authentifié, modes compétition et challenge (3 sortes: par user)
 - **Réalisé en fin d'itération**
- **NEW: Les JUnit Tests sont implémentés pour toutes les fonctions**
 - **15 tests sont disponibles. Nouveautés:** joindre et recevoir une grille de compétition (yc l'échec), challenge avec les 3 modes (yc l'échec), vérification correcte des solutions (yc l'échec).
 - **Réalisé en fin d'itération**
- *Gestion de l'ambiance du groupe & motivation : Eléonore (hotline 24h/24)*

Bilan global: Tout a été *réalisé comme prévu, pas de replanification à opérer*

8. 10.06 - 16.06 effort global prévu : faible. Réalisé: très élevé (37h)

- Essai de l'application par des utilisateurs lambdas (alpha-testeurs), debugging en cas de découverte de bug. (David et Eléonore 1h chacun)
- Marge de manœuvre si des itérations précédentes ont été plus compliquées que planifié (Paul, amélioration du client : 10h)
- Préparation du bilan final (Karim : 9h, David, Eléonore, Paul, Valentin : 4h chacun)
- **Réalisé en fin d'itération**

6. État des lieux

Tout fonctionne correctement. Nous avons également implémenté 15 tests unitaires afin de vérifier notre code.

Nous pourrions imaginer facilement l'ajout de diverses fonctionnalités. En premier lieu, nous avons pensé à donner les solutions d'une grille une fois que le joueur la termine afin qu'il puisse s'améliorer. D'ailleurs, le schéma du protocole client-serveur au point 3.1 a été conçu de manière à pouvoir ajouter cette fonctionnalité (même si elle ne figurait pas dans le plan d'itération initial).

En profitant de notre conception modulaire, nous pourrions rajouter des langues afin d'internationaliser notre application. Toutefois, nous devrions nous restreindre à l'alphabet latin.

Une troisième fonctionnalité serait de modifier le système de comptage de points, en apportant un bonus à tous les mots qui appartiennent à un certain thème. En tant que futurs ingénieurs en informatique, l'idée qui est sorti des discussions serait de rajouter un multiplicateur de points à tous les mots touchant à l'informatique.

Nous pourrions alors proposer la planification suivante:

17.06 - 24.06 effort global prévu : moyen

- o Les langues suivantes sont ajoutées: anglais, allemand, espagnol, italien
 - Développement des fonctionnalités
 1. Cas d'utilisation: "pré validation de la grille", "résolution de la grille", "validation de la grille"
 2. Réalisation entière
 3. Modification des listes de mots, et des critères de validation et de pré-validation par langue. Gérer les caractères spécifiques de chaque langue.
- o Le client peut voir les solutions une fois qu'il termine une grille
 - Développement des fonctionnalités
 1. Cas d'utilisation: "challenge", "compétition", "entraînement"
 2. Réalisation entière
 3. Développement définitif
- o Les mots liés à l'informatique comptent triple
 - Développement des fonctionnalités
 1. Cas d'utilisation: "vérification des solutions"
 2. Réalisation entière

7. Autocritique

En somme nous estimons que ce projet est une réussite, ayant implémentés chaque cas d'utilisation préalablement défini. Le travail a été facilité par la bonne entente au sein du groupe. Egalement, la répartition des tâches a été très bien faite.

Nous aurions dû prévoir plus de temps pour la synchronisation avec Github. Le fait d'ajouter de petites tâches autour du cœur projet (itération 4) n'était pas une bonne idée. Cette quatrième itération aurait dû exclusivement comprendre la résolution d'une grille. Ces problèmes mineurs (évoqués au point 4.1), n'ont pas eu de conséquence sur l'avancement du projet

Nous remarquons que le fait d'avoir laissé une petite itération en fin de projet a pu limiter les risques de retard. Nous pensons aussi que l'ampleur du projet nous a poussés à être plus rigoureux dans notre approche du problème.

8. Conclusion

En conclusion, nous sommes satisfaits du résultat de ce projet. Par contre, si nous avions eu plus de temps libre pour coder, même en dehors du cadre du cours, il aurait été intéressant de pousser le projet plus loin. Sur la fin il était aussi démotivant de travailler, en sachant que le projet ne comptait que pour une petite partie de la note. Nous avons tout de même pris plaisir à travailler en groupe, et le referons peut-être dans le futur.

Annexe

Installation :

Prérequis à l'installation server

- installer les outils mySQL et phpMyAdmin (p.ex: WAMP sur Windows, MAMP sur MacOS X)
- paramétrer les accès mySQL aux valeurs port = 3306, utilisateur = "root", password = ""
- lancer mySQL
- l'IP du server doit être atteignable depuis le client. En particulier, le pare-feu du server doit être désactivé, et les deux composants (clients et server) doivent se trouver dans le même réseau local.

Installation du server

- dans mySQL, importer le fichier de création de la base de données, se trouvant dans database/wordhunt-schema.sql
- lancer le projet en lançant java -jar server.jar dans un terminal (ou cliquant sur server.jar dans l'explorateur de fichier ou depuis un IDE, grâce au main se trouvant dans la classe wordhuntserver/src/server/WordHuntServer.java)
- lancer les test (projet WordHuntClientTest) afin de vérifier le bon fonctionnement du tout, les 15 tests doivent se dérouler correctement. Ces tests vont également créer un utilisateur admin, avec le mot de passe "adminPassword". Si cette étape est sautée, il est nécessaire de créer l'utilisateur admin manuellement par le client (afin d'éviter qu'un utilisateur le fasse à notre place).
- lors du lancement en IDE ou en ligne de commande, le serveur va exhiber les adresses IP auquel il peut être contacté. Repérer l'adresse IP local (10.x.y.z à la HEIG ou 192.168.x.y à domicile). A défaut, taper ifconfig ou ipconfig dans terminal pour récupérer l'IP.

Installation client (sur appareil android, version 4.0 ou supérieure)

- transférer le fichier client.apk sur l'appareil (p.ex. par dropbox ou email)
- dans "Paramètres">"Personnel">"Sécurité", cochez la case "Sources Inconnues Autorisées" pour permettre l'installation directe d'application, sans passer par Google Play Store.
- ouvrir le fichier client.apk et choisir "continuer avec installateur système" (et pas Google Play).
- ouvrir l'application. Dans "Settings", saisir l'IP du server (trouvée ci-avant).

Installation client (sur appareil android ou émulateur, p.ex. celui d'Android Studio)

- brancher le câble de l'appareil

- installer depuis Android Studio (déployer, choisir la cible)
- ouvrir l'application. Dans "Settings", saisir l'IP du server (trouvée ci-avant).

Utilisation :

Prérequis: avoir installé l'application comme décrit dans la section précédente.

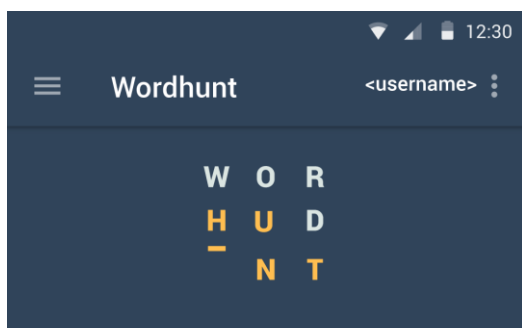
Démarrage de l'Application



Ouvrir l'Application WordHunt (chercher le logo ci-dessus dans le menu "Toutes Applications" ou "Applications" selon les versions Android).

Menu principal

Au démarrage, on se trouve dans le menu principal (home).



Partie hors ligne

Entrainement

Challenge

Competition

se connecter

s'inscrire sur Wordhunt

Important: changer les paramètres IP

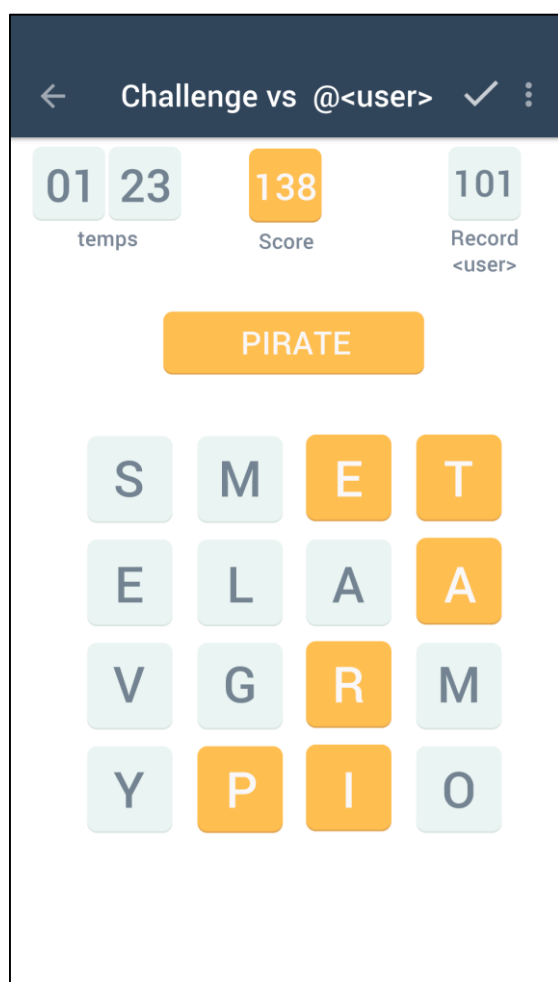
Une étape essentielle est de se rendre dans les paramètres (Settings, 3 points verticaux en haut à droite), pour y configurer correctement l'IP du server. En effet, si le server n'est pas déployé sur un domaine DNS ou une IP fixe, il est nécessaire de saisir l'IP manuellement.

Prêt à jouer en mode "Hors-Ligne" (Entrainement Non Authentifié)

Dès lors, vous êtes prêts à jouer! Mais attention, vous n'avez pas créé de compte, et vous n'êtes pas authentifié! Vous ne pouvez donc uniquement jouer en mode "Hors Ligne" (c'est-à-dire Entrainement Non Authentifié). Dans cette configuration, le serveur va vous donner une grille de jeu, qu'il oubliera directement. Il ne sera pas possible de participer à des challenge ou compétitions, ni d'enregistrer des scores.

Principe général du jeu (tous les modes)

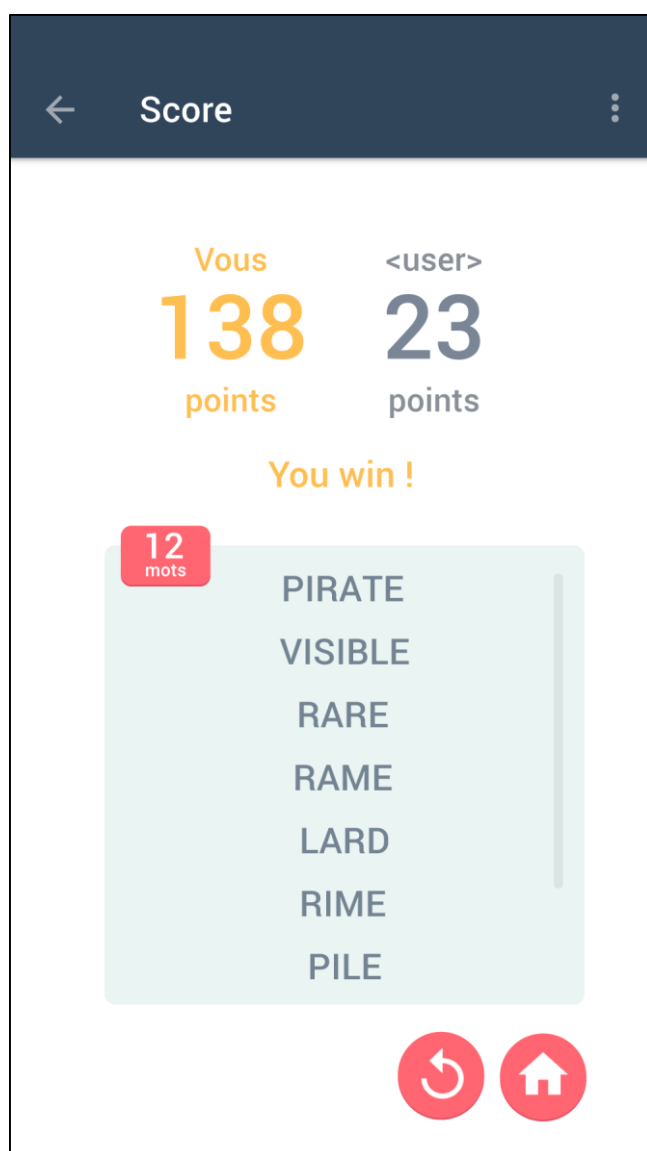
Une grille de lettres est affichée. En glissant sur les lettres contiguës (diagonale comprise, donc 8 directions possibles depuis chaque lettre), vous devrez composer le maximum de mots de la langue française en un temps limité. Sont acceptés tous les mots, déclinaisons, conjugaisons, formes, pluriel, féminins, etc. figurant dans le dictionnaire officiel du SCRABBLE®. Par exemple, dans la grille ci-dessous, on a sélectionné le mot "PIRATE" et il reste 1 minute 23 secondes pour jouer.



Note: la figure représente un challenge contre un autre utilisateur. Toutefois, tous les modes de jeux sont très similaires quand au but et fonctionnement du jeu.

Fin du jeu

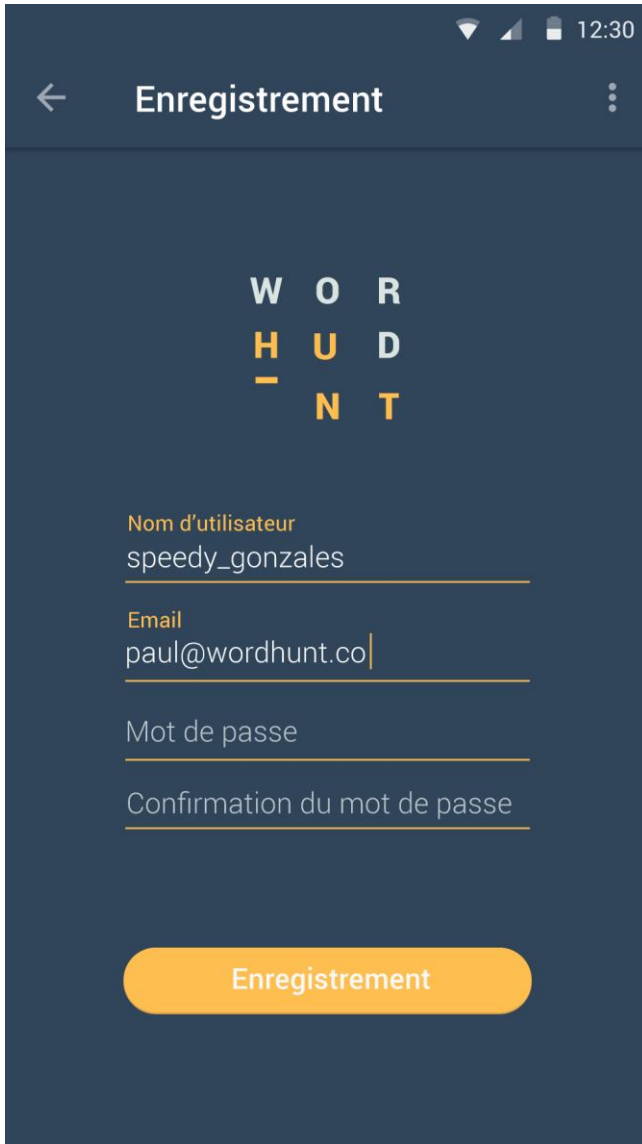
A la fin du temps, ou plus tôt si vous le souhaitez en cliquant sur la flèche (le vu/check mark ☑), le jeu se termine et affiche vos mots trouvés et le score. Dans toutes les parties authentifiées (voir plus loin), le score est automatiquement transmis. Dans une version future, les mots à trouver (disponibles et que vous n'avez pas trouvés) s'afficheront également.



Si vous souhaitez maintenant accéder aux fonctions plus avancées: il faudra créer un compte puis se connecter.

Création de compte sur WordHunt

Cliquez sur "s'inscrire sur wordhunt" et la page ci-dessus apparaîtra. Enregistrez-vous en choisissant un nom d'utilisateur (username) unique à WordHunt, et un mot de passe suffisamment protégé. Entrez également votre email, qui ne doit pas encore avoir servi pour l'enregistrement sur WordHunt.



W O R
H U D
- N T

Nom d'utilisateur
speedy_gonzales

Email
paul@wordhunt.co

Mot de passe

Confirmation du mot de passe

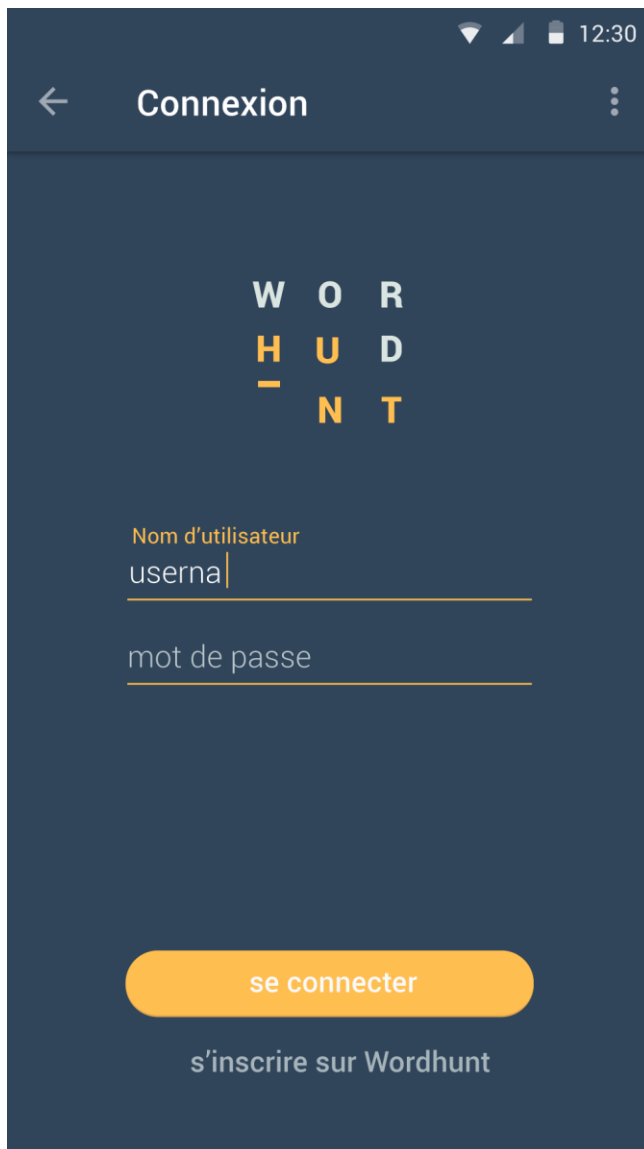
Enregistrement

Attention: n'oubliez pas votre mot de passe! Actuellement, il n'est pas possible de le changer, ni de le remettre à zéro. De plus, il n'est pas possible de le récupérer en clair, celui-ci étant stocké de façon cryptographiquement irréversible.

Après avoir cliqué sur "Enregistrement", vous recevrez une confirmation. Sinon, réessayez plus tard, ou quitter l'application et recommencez. Une fois enregistré correctement, vous pouvez passer à l'étape suivante: la connexion.

Connexion à WordHunt

Cliquez sur "Connexion" et connectez-vous en utilisant le nom d'utilisateur et le mot de passe définis lors de l'enregistrement. Si vous avez oublié votre mot de passe, contactez les administrateurs à support@wordhunt.org pour obtenir un nouveau mot de passe, ou créer un nouveau compte avec un autre username et email.



A la pression du bouton "se connecter", vous recevrez un message de confirmation, et votre appareil recevra un jeton de connexion valable. Actuellement, celui-ci est valable indéfiniment. Si vous recevez un message d'erreur, vérifiez les données, ou réessayez plus tard, ou quitter l'application et recommencez.

Jeu en mode "Authentifié"

Vous êtes maintenant connecté. Tous vos jeux se réalisent de manière authentifiée, et vos scores sont automatiquement transmis au serveur après chaque partie.

Jeu en mode Entraînement Authentifié

Au clic sur le bouton "Entraînement", vous recevrez une NOUVELLE grille, vous êtes les premiers à y participer, et celle-ci a été sauvegardée côté serveur, afin de permettre à d'autres de vous défier par la suite. A la fin de la partie, vos scores sont transmis et enregistrés.

Jeu en mode Challenge

Au clic du bouton "Challenge", vous essaierez de jouer une partie qu'un autre utilisateur a déjà jouée afin de le défier et tenter de battre son score. Il existe trois façon de se procurer une grille: sans spécifier de paramètres, en choisissant un nom d'utilisateur, en choisissant une grille.

Sans spécifier de paramètres, vous recevrez n'importe quelle grille jouée par quelqu'un d'autre. Ce mode est très pratique si vous ne connaissez personne, ou si les autres choix sont trop restrictifs. En choisissant un nom d'utilisateur, vous aurez la possibilité de jouer une grille déjà jouée par l'utilisateur choisi. En choisissant une grille (par son identifiant), vous jouerez cette grille précise.

Attention: il est tout à fait possible que votre recherche soit infructueuse, auquel cas un message d'erreur s'affichera. En particulier, il est possible qu'il n'y ait aucune grille à laquelle vous n'avez pas jouée, qui correspond à vos critères (revenez plus tard, attendez qu'un autre utilisateur joue). De plus, il faut que l'utilisateur ou la grille choisie existe déjà. Par ailleurs, dans le cas de choix de grille, vous ne devez pas avoir déjà joué cette grille. Et dans le cas de choix d'utilisateur, celui-ci doit avoir joué à une grille à laquelle vous n'avez pas encore joué.

Jeu en mode Compétition

En cliquant sur le bouton "Compétition", vous essaierez de rejoindre une compétition. Attention, s'il n'y pas de compétition en cours, ou que la prochaine est prévue mais pas encore disponible, vous recevrez un message d'erreur. Ceci est parfaitement normal. Réessayez lorsqu'une compétition sera programmée et en cours (disponible). Ceci est absolument dépendant des compétitions prévues par l'administrateur du jeu.

Le principe du jeu est exactement le même. Toutefois, plusieurs joueurs jouent simultanément sur la même grille. A la fin, les score sont soumis au serveur. Dans une version futures, les scores de tous les participant à la compétition seront affichés chez tous les utilisateurs (actuellement, seul l'administrateur peut les consulter dans la base de données).

Fonctionnalités Administrateur

Lorsque vous êtes connectés en tant qu'administrateur, vous avez accès directement depuis le client à des fonctions supplémentaires. Actuellement, seul l'utilisateur "admin" possède de tels droits. Actuellement, seul la planification de compétition est possible depuis le client (le reste des fonctions, tels que la consultation des données stockées, peut toujours se faire depuis l'interface MySQL / phpMyAdmin)

Planification de Compétition

En tant qu'administrateur, vous avez la possibilité de planifier des compétitions que les utilisateurs pourront joindre pendant la durée de celle-ci. Dans les paramètres (settings), cliquez sur "Planifier une compétition",



Nouvelle compétition

Délais [s]

Durée [s]

Lancer la compétition

Sur l'écran obtenu, spécifier le délai de lancement de la compétition et la durée de la compétition (fenêtre de temps pendant laquelle la compétition peut être jointe). Si laissés vides, les champs prendront les valeurs par défaut, à savoir un lancement immédiat (1 sec), et une compétition joignable pendant 5 minutes. Pressez sur "Lancer la compétition". Vous recevrez un message de confirmation. Sinon, un message d'erreur s'affichera (notamment si vos droits admin sont expirés ou s'il y a déjà une compétition en cours).

A partir de ce moment, les utilisateurs souhaitant joindre une compétition ne recevront plus le message "aucune compétition" mais à la place "prochaine compétition dans X secondes". Dès que le délai (delay) sera écoulé, les utilisateurs pourront joindre la compétition.

Diagramme d'activité général

