

UNIVERSITÉ JEAN MONNET
FACULTÉ DES SCIENCES ET TECHNIQUES

Développement d'un jeu de Société : Le Scrabble

RESPONSABLES :

NADÈGE HOTELLIER
ADRIEN LONGÈRE
ÉMILIE MORVANT



Table des matières

Résumé	2
Introduction	3
1 Présentation du projet	4
1.1 Sujet	4
1.2 Cahier des charges	4
1.3 Limites	5
2 Etude de conception	6
2.1 UML 1.0	6
3 Développement du jeu	7
3.1 Parties Client-Serveur et Interfaces	7
3.2 Partie Dictionnaire et Recherche	8
3.3 La gestion des parties en cours	9
3.4 Partie Joueurs et Intelligence Artificielle	10
3.5 ANT : Exécuteur de tâches	13
Conclusion	14
Bibliographie - Webgraphie	15

Résumé - Abstract

🇫🇷 Français

Il s'agit de concevoir le célèbre jeu de Scrabble en intégrant un certain nombre de fonctionnalités supplémentaires. Le dictionnaire officiel, autrement appelé OSD¹, est intégré à l'application. Les points forts étant la gestion du jeu en réseau ainsi que la mise en place d'une intelligence artificielle ayant plusieurs niveaux de jeu.

🇬🇧 Anglais

It's the conception of the famous Scrabble game adding some features. The official dictionary, otherwise called OSD, is integrated into the application. The main points are the network management of the game as well as the implementation of an artificial intelligence having several levels of game.

¹OSD : Official Scrabble Dictionary

Introduction

Après réflexion et mise en commun de nos choix respectifs, avantages et inconvénients de chaque sujet proposé, nous avons retenu celui de la création du jeu de Scrabble. Effectivement, la création d'un jeu de mots croisés, avec insertion d'une intelligence artificielle, nous a paru être une très bonne expérience et nous permettait de mettre en pratique certaines théories apprises.

De plus, ce projet, comme tout autre, nous permettait d'apprendre. Nous allions pouvoir explorer et implémenter une IA, un dictionnaire permettant une recherche rapide des quelques milliers de mots existants et la mise en place d'une application en mode multi-joueurs par le biais d'un serveur.

Le choix du langage JAVA a été essentiellement motivé pour sa conception objet et sa facilité d'implémentation d'interfaces graphiques.

Nous allons commencer par présenter le sujet de manière à exprimer les besoins ressortissants. Nous poursuivrons avec une analyse de conception du logiciel permettant de mener à notre dernière partie qu'est le jeu en lui-même.

Chapitre 1

Présentation du projet

1.1 Sujet

L'objectif de ce projet est de programmer un jeu de Scrabble. Les joueurs auront la possibilité de jouer à plusieurs et contre la machine.

On utilisera un dictionnaire en ligne (par exemple [mediadico](#)) pour vérifier les mots proposés. On pourra par ailleurs utiliser un corpus pour fournir au programme un ensemble de mots possibles.

1.2 Cahier des charges

Adrien

- Afin de pouvoir permettre la communication entre les différents clients, il faudra mettre en place une architecture 2-tiers ou 3-tiers Client/Serveur.
- Etant donné qu'il s'agit de la conception d'un jeu de société, se basant essentiellement sur l'aspect visuel, toute interface et tout design devront être agencés de manière à permettre une utilisation simple de l'application.

Toute implémentation favorisant la simplicité est la bienvenue : *skins, drag'n'drop*

Émilie

- Faisant partie de la famille des jeux de mots croisés, le Scrabble a besoin d'un dictionnaire de base afin de vérifier les mots proposés par les joueurs. Ce dictionnaire se présente sous forme d'une liste officielle de mots acceptés. Le jeu permet également de faire vérifier le mot sur internet via un dictionnaire en ligne.
- Pour les débutants, une fonctionnalité de complétion d'un mot sera ajoutée.
- Ce jeu multijoueurs et multipartie doit être équipé d'interfaces correspondantes. Elles permettront ainsi de créer ou rejoindre une partie.

Nadège

- Afin de permettre une utilisation de la plateforme de jeu, les joueurs doivent avoir en leur possession un certains nombres d'actions possibles. En effet, un joueur doit pouvoir proposer un mot, piocher de nouvelles lettres, passer son tour ...
- Le sujet de création de ce Scrabble propose l'implémentation d'une intelligence artificielle. Cet IA¹ est un joueur 'automatisé'. Muni de différents niveau, un joueur 'humain' pourra donc jouer seul contre la machine.

Ce cahier des charges a dû être remodelé au cours du développement de l'application dû notamment au fait que chaque partie est en liaison avec une autre. Le dictionnaire a donc été créé de manière simple, avec toutes les fonctionnalités que le dictionnaire requierait pour permettre aux membres du groupe d'avancer.

Au fur et à mesure de l'avancé, il s'est avéré être efficace bien qu'un peut lourd. Ce changement sera expliqué dans ce rapport.

¹IA : Intelligence Artificielle

1.3 Limites

Du fait de la contrainte de temps particulièrement courte et de la complexité du projet, nous nous sommes fixé des limites :

- Un joueur ne pourra s'identifier plus d'une fois.
- Un joueur ne pourra jouer plus d'une partie simultanément.
- Aucune gestion de profil utilisateur bien que le développement offre la possibilité de l'implémenter par la suite.

Chapitre 2

Etude de conception

2.1 UML 1.0

Pour l'analyse de ce présent projet, nous établirons les diagrammes sous UML 1.0.

2.1.1 Diagramme de classes

L'application possédant un nombre trop important de classes, nous avons volontairement choisis de représenter les liaisons entre les différents paquetages de l'application logicielle. Ainsi, nous pouvons avoir une première idée sur le volume et la complexité de celle-ci.

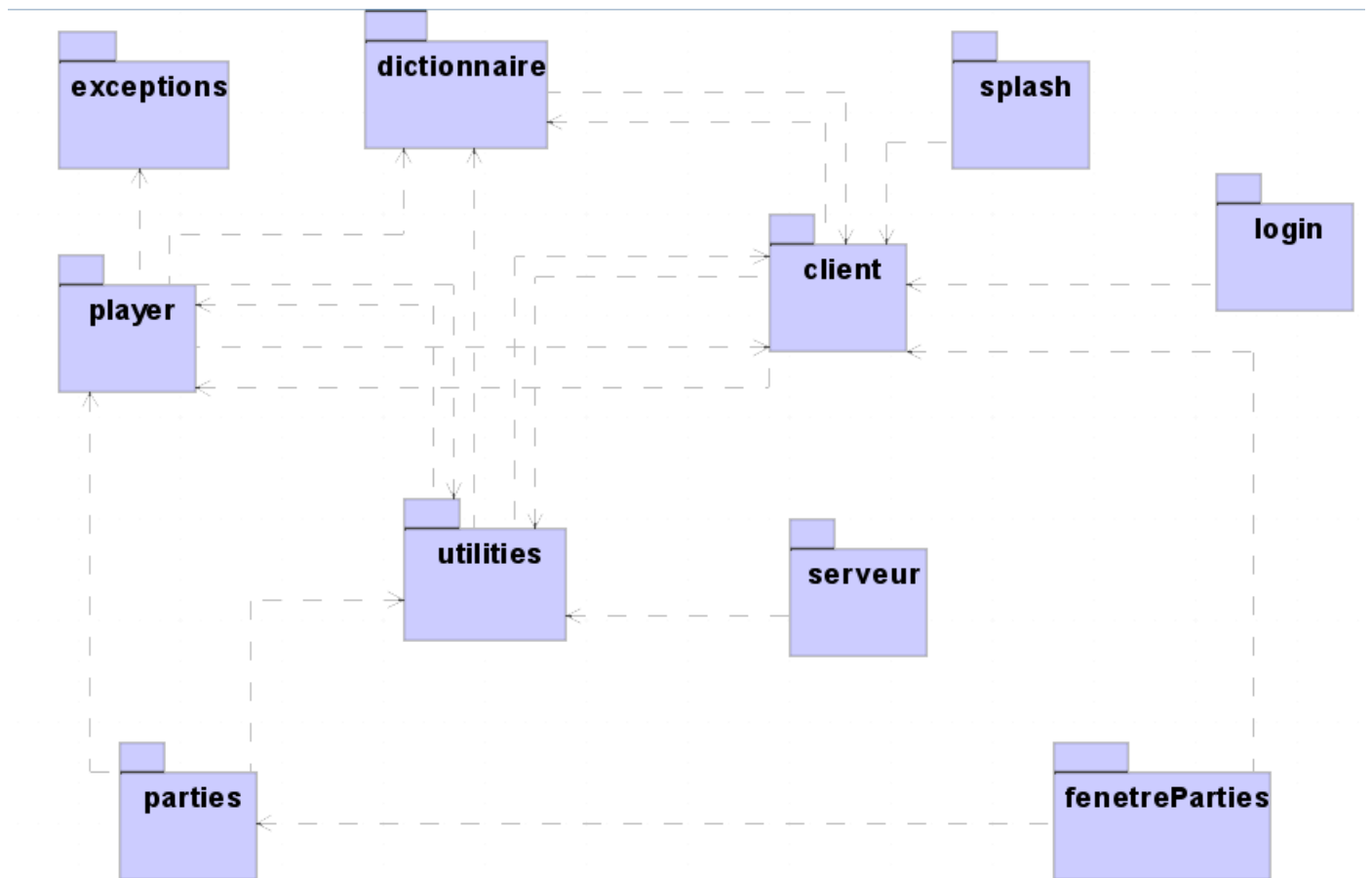


FIG. 2.1 – Diagramme de packages

Chapitre 3

Développement du jeu

3.1 Parties Client-Serveur et Interfaces

L'interface graphique du client se compose de différentes parties :

- splashscreen affiché pour le chargement des images
- login pour la connexion d'un utilisateur avec des sous parties
- client principal avec affichage du jeu et composantes diverses

La langue de l'interface est directement paramétrable depuis la fenêtre de login, les différentes chaînes affichées étant initialement contenues dans un document XML, classées par langue et par identifiant.

Une fenêtre permet également de configurer le proxy (pas vraiment fonctionnel à l'heure d'aujourd'hui).

La création d'un nouvel utilisateur se fait par une extension de la fenêtre principale de login.

Le déplacement des cases s'effectue par drag'n drop : on clique sur une case et on relâche sur une autre. Le curseur prend l'apparence de la case sélectionnée durant le déplacement pour accentuer l'effet visuel.

Le plateau est composé d'une matrice de cases autonomes ayant toute leurs attributs.

Le 'joker' est géré de façon à ce qu'un clic fasse apparaître sur ce dernier un champ dans lequel on ne peut saisir qu'une seule lettre qui sera vérifiée.

Un fenêtre annexe permet de rechercher dans le dictionnaire des mots de manière incrémentale : plus on saisit de lettres plus on réduit le nombre de mots résultants.

Les communications s'opèrent autour d'un serveur qui centralise les parties.

Toute communication est faite avec ce serveur qui met en relation des clients autour de parties.

Les communications se font en TCP/IP choisi pour sa simplicité et sa fiabilité, la performance du réseau n'étant pas ici critique.

Les paquets envoyés sont tous au format XML, ce qui permet une création et une analyse structurée et simple. Chaque requête/réponse fait l'objet d'une partie spécifique dans un message XML standard. On peut ainsi cumuler des informations dans un même message.

Dans l'idée, le serveur stocke dans une table chaque client qui se connecte et les associe à une partie lorsqu'ils en font la demande. Pour vérifier l'authenticité des utilisateurs, le serveur dispose encore une fois d'un fichier XML dans lequel il stocke les logins et mot de passe des utilisateurs inscrits. Dans le cas présent seuls deux utilisateurs sont inscrits dans ce fichier et peuvent se loguer pour jouer une partie, la gestion étant assez complexe. Cependant, on peut également se connecter avec un autre login et mot de passe. Le seul bémol est que ce nouvel utilisateur ne sera pas encore inscrit dans le fichier xml.

3.2 Partie Dictionnaire et Recherche

Afin de pouvoir jouer au scrabble, un des outils les plus importants est la “liste” des mots autorisés. C’est pourquoi, il nous a fallu définir une structure correspondant au dictionnaire des mots.

Nous disposons de la liste des mots classés par ordre alphabétique, dans un fichier de type texte, auquel nous avons donné l’extension `.dic`, contenant un mot par ligne.

Exemple : Extrait du fichier `francais.dic` :

```
AA
AAS
ABACA
ABACAS
ABACOST
ABACOSTS
ABACULE
ABACULES
ABAISSA
ABAISSABLE
...
```

Bien entendu, nous n’avons pas construit ce fichier nous même. Pour la version française, il s’agit de la version 5 du dictionnaire officiel du ScrabbleTM (ODS 5).

La question qui s’est donc posée était : *“Comment, à partir de cette liste, construire un dictionnaire accessible rapidement pour le simple joueur mais aussi pour les robots ?”*

3.2.1 La structure

3.2.1.1 Choix de départ

Notre premier choix fut de représenter ce dictionnaire sous forme d’un arbre, afin de faciliter la recherche de mot(s). Néanmoins, après quelques semaines de recherches et de tests infructueux, nous nous sommes aperçus que cette piste était loin d’être optimale.

En effet, la structure que nous avons retenue était un arbre binaire de la forme :

```
private Noeud pere;      //permettant de parcourir l'arbre en sens inverse
private Noeud racine;
private Branche brancheDroite;
private Branche brancheGauche;
private Arbre filsGauche;
private Arbre filsDroit;
```

Chaque nœud possède la partie du mot courant, chaque feuille contient, quant à elle, un mot.

Néanmoins, pour représenter un alphabet de 26 lettres sous la forme d’un arbre binaire, la solution choisie était de placer une étiquette représentant un ensemble de lettres sur les branches. Cet ensemble étant représenté par deux caractères : la première lettre de l’ensemble, la seconde la dernière lettre, à savoir que si les deux lettres ne sont pas identiques, alors l’ensemble est partagé en deux pour créer deux nouvelles branches ; si elles le sont, alors on ajoute cette lettre au mot du nœud et on recommence.

Cette structure s’est avérée prendre beaucoup trop d’espace mémoire ; il était impossible de créer l’arbre comportant tous les mots. Ce problème a été détecté beaucoup trop tard. En effet, nous avons fait l’erreur de tester cette structure sur une toute petite liste de mot, et lors du test sur l’ensemble du dictionnaire, le résultat fut tel que nous ne pouvions stocker uniquement les mots allant de ‘a’ à ‘b’.

Étant donné l’avancement dans le temps et cette réflexion infructueuse, nous avons décidé de ne pas nous replonger au début de la réflexion et nous avons préféré réfléchir à d’autres fonctionnalités du jeu, sachant que le dictionnaire provisoire donnait de bons résultats et était suffisant pour l’utilité que nous en avons.

3.2.1.2 Choix final

Finalement, notre structure provisoire qui représentait le dictionnaire fut la version finale. Il s'agit d'une simple liste de mots. Le langage JAVA nous fournissant des fonctionnalités adaptées à notre besoin concernant ce dictionnaire, nous avons pu faire en sorte de créer des fonctions de complétion et de validation de mots.

3.2.2 Faut-il rendre le dictionnaire évolutif ?

Une des méthodes définies dans le dictionnaire permet de vérifier si un mot existe en ligne¹. Ceci pourrait permettre une évolution constante du dictionnaire courant. La question à se poser serait *“Où placer les limites de cette fonctionnalité ?”*

- Faut-il que ce soit accessible à tout joueur : en vérifiant si un mot existe, l'enregistrer ?
- Faut-il que seul certains utilisateurs en aient le droit ?

Après différents débats entre nous, il a été décidé de ne pas encore rajouter cette fonctionnalité en vu des différentes questions qu'elle soulève, mais la fonction permettant de vérifier si le mot existe est prête à être utilisée : `public boolean existWord(String mot)`.

3.3 La gestion des parties en cours

Cette fonctionnalité n'a pas pu être ajoutée, par manque de temps, mais voici sa description comme si elle aurait pu être implémentée.

Tout utilisateur doit pouvoir rentrer dans une salle afin de jouer avec un autre utilisateur ou un robot. Il peut aussi, à partir de l'interface, devenir spectateur d'une partie ou d'une démonstration.

Deux packages différents sont importants pour ceci : `scrabble.parties` et `scrabble.fenetreParties`. La première gérant les parties à proprement dites et la seconde, l'interface graphique.

3.3.1 Les parties

- Une partie : `Partie` `protected int langue;`
`protected int idPartie;` // chaque partie possi $\frac{1}{2}$ de un identifiant
`protected Player player1;` // pour pouvoir jouer il faut i $\frac{1}{2}$ tre 2
`protected Player player2;`
`protected Bag bag;` // le sac de lettre
`protected Spectateurs spectateurs;` // L'ensemble des spectateurs
`protected boolean privee;` // si i $\frac{1}{2}$ gale i $\frac{1}{2}$ i $\frac{1}{2}$ true la partie est privi $\frac{1}{2}$ e
// et n'acceptera pas de spectateurs
`protected boolean enCours;` // si i $\frac{1}{2}$ gale i $\frac{1}{2}$ true la partie i $\frac{1}{2}$ d'i $\frac{1}{2}$ marrer

Cette classe permet de créer, démarrer, rejoindre ou quitter des parties. Un joueur pourra aussi s'asseoir ou se lever durant une partie, cela correspond respectivement à passer de l'état joueur à l'état spectateur et inversement.

- L'ensemble des spectateurs : `Spectateurs` - Il représente une liste de joueurs (`List<HumanPlayer> spectateurs`) à laquelle on peut tout simplement ajouter ou supprimer des éléments.
- L'ensemble des parties : `Parties` - Il est représenté sous la forme d'un vecteur de `Partie` : `Vector<Partie> partiesEnCours`.

Cette classe permet de rajouter ou de supprimer des parties. De plus, un utilisateur pourra rejoindre une partie donnée dans l'ensemble.

3.3.2 Gestion graphique de l'ensemble

- La fenêtre comportant toutes les parties : `FenetreParties` - Ici est affiché la liste des parties en cours sous la forme d'une `JList` (à l'intérieur d'un `JScrollPane`, afin d'avoir une barre de défilement). L'avantage d'une `JList` est qu'elle permet d'afficher une liste d'objets à partir d'un tableau ou d'un vecteur. Dans

¹url du type : http://dictionnaire.mediadico.com/traduction/dictionnaire.asp/definition/mot_a_chercher/2008

notre cas, nous utilisons un vecteur de parties ; chaque parties étant un composant de la liste. Ainsi, il a fallut définir la partie graphique d'un composant : `UnePartieRenderer`.

De plus, cette interface contient un bouton pour créer une nouvelle partie. Lorsque l'on clique dessus, le programme ouvre une nouvelle fenêtre spécifique à la création d'une nouvelle partie : `FenetreNouvellePartie`.

- La partie graphique d'une partie : `UnePartieRenderer` - Pour une partie, il est plus judicieux d'afficher différentes données :
 - le numéro de la salle correspondant à l'identifiant de la partie
 - les deux joueurs (humain ou robot)
 - le nombre de spectateurs²
 - la langue de la partie
 - si la partie est privée ou publique
 - si la partie est en cours ou non
 - un bouton permettant de rejoindre la partie sélectionnée

Il est important de souligner que la `JList` "absorbe" les actions effectuées par l'utilisateur, ce qui n'a pas été sans poser de problème. En effet, lorsque l'on clique sur un composant, il se retrouve simplement sélectionné (il est alors affiché en grisé dans notre cas). Ainsi, le clic sur un bouton à l'intérieur d'un composant n'est pas accessible. Il faut donc redispacher les actions sur les différents composants. Pour cela, lors de la création de la `JList` (dans `FenetreParties`), il faut lui ajouter un écouteur avec un redispacheur des événements de la souris : `RedispatchMouseListenerListe`.

- Redispacher les événements : `RedispatchMouseListener` - Cette classe consiste à récupérer les différents événements effectués à la souris, puis en fonction de l'endroit cliqué, à envoyer l'événement dans le bon composant graphique.

Dans notre cas, l'événement qui nous intéresse est "cliquer sur un `JButton`". Afin de déterminer si le clic a bien eu lieu sur un bouton. Il faut donc une méthode `Component getComponentAt(int x, int y)` dans la classe `UnePartieRenderer` qui renvoie le composant cliqué. Cependant, nous avons dû faire face à un petit problème : il s'avère assez compliqué de récupérer le composant `JButton`.

Après différentes tentatives infructueuses et afin de ne pas perdre plus de temps sur ce petit détail, nous en renvoyons le composant en fonction des coordonnées du clic : si elles se trouvent dans le bouton, alors cette fonction le renvoi.

Il suffit ensuite d'affecter un écouteur de souris au bouton avec l'action "rejoindre la partie".

- La fenêtre de création d'une nouvelle partie : `FenetreNouvellePartie` - Cette fenêtre s'ouvre si l'utilisateur a cliqué sur le bouton de création d'une nouvelle partie. C'est ici que l'on spécifie les différentes options de cette dernière. Nous avons pris parti de créer automatique une partie dans la langue du joueur.

3.3.3 Que peut faire un utilisateur ?

- Un utilisateur peut accéder à une partie (en tant que joueur ou spectateur) ou en créer une nouvelle, seulement s'il n'est pas déjà sur une table de jeu.
- Il ne peut accéder qu'aux parties dans sa langue³.
- Une partie de type privée n'accepte que deux utilisateurs au maximum.

3.4 Partie Joueurs et Intelligence Artificielle

Nous allons ici décrire le mécanisme de fonctionnement des joueurs (humain et robot) ainsi que les classes nécessaires utilisées.

Après avoir évoqué les relations entre un joueur et les différents 'objets' qui lui sont propres ou dont il se sert, nous commencerons par détailler le joueur humain qui est moins complexe que le robot.

²Afin de gérer le "pluriel" d'un mot, nous avons créé une méthode `public static String Pluriel(int i)` qui permet de rajouter un 's' à un mot si $i > 1$.

³Ceci est un parti pris, nous pourrions modifier cette option assez facilement.

3.4.1 Préparation

Afin de pouvoir effectuer des tests unitaires pour intégrer par la suite les joueurs, certaines classes ont été créées.

Nous avons la classe `Lettre` qui, comme son nom l'indique représente une lettre avec ses caractéristiques dans son contexte, à savoir le jeu de Scrabble. Ainsi, elle a pour attributs : un caractère (la lettre), une position et un type.

Il est évident que cette classe est similaire voire la copie d'un objet de type `Case`, mais comme il a été dit, ce ne sont que des classes provisoires avant l'intégration ou propres aux joueurs.

Chaque joueur dispose d'un ratelier qui contient les lettres qu'il peut jouer à un moment donné. Cet objet est modélisé par la classe `Rack` (N.B Ratelier) qui ne comporte qu'une seule chose : une liste de '`Lettre`' sur laquelle des opérations sont effectuées (changement, ajout, permutation ...).

Les classes suivantes ne sont utilisées que par le robot qui est représente l'ordinateur contre lequel un humain peut jouer.

Ainsi, la classe `Word` (N.B Mot) représente un mot. Tout comme le ratelier, elle ne possède qu'un attribut qui n'est autre qu'une liste de '`Lettre`'.

Cette classe est par la suite étendue par `RobotWord` qui représente le mot qu'un robot peut jouer. Elle contient le mot sous forme d'une chaîne simple et une variable booléenne indiquant la direction du mot. La variable, `isVertDirection()`, vaudra 'vrai' si le mot est vertical et faux dans le cas contraire.

C'est cette classe qui contiendra la méthode permettant de positionner le mot sur le plateau de jeu.

Cette dernière, nommée `isWordCorrectToPlace()` fonctionne de la manière suivante :

- On parcourt le mot à la recherche d'une lettre appartenant au plateau.
Si l'une d'elle est effectivement présente sur le plateau, alors on sort de la boucle et l'indice qui permettait le parcours du mot est inférieur à la taille du mot en question. Dans le cas contraire, cet indice vaut la taille du mot.
- Si aucune lettre n'appartient au plateau, la première lettre du mot est placée au centre du plateau.
- Selon que le mot est vertical ou horizontal, chacune des autres lettres composant le mot vont se voir affecter une position.

3.4.2 Les différents joueurs

3.4.2.1 Interface Player

Cette interface définit les principales méthodes des joueurs. Elle est surtout utile pour l'une d'entre elles, à savoir la méthode `equals` qui est redéfini.

En effet, dans ce jeu, les joueurs humain et machine n'ont pas grand chose en commun. L'un possède quelques méthodes lui permettant de jouer ou valider un mot, tandis que l'autre doit composer un mot en fonction d'un niveau de jeu et d'un plateau.

C'est pourquoi certaines fonctions apparaissant dans cette interface ne sont pas redéfinies dans la classe du joueur machine car il n'en a pas besoin.

3.4.2.2 Joueur Humain

La classe `HumanPlayer` implémente donc l'interface précédente. Elle possède les caractéristiques principales d'un joueur, soit un nom, un niveau, un score et un ratelier. Les actions associées sont définies :

- valider un mot
- demander de l'aide pour créer un mot
- mélanger des lettres
- changer des lettres

3.4.2.3 Joueur Machine

Comme son nom l'indique, cette dernière classe représente le robot. Implémentant l'interface, elle possède un nom, un niveau, une langue, une liste des lettres du plateau, une liste des mots qui ont été refusés, un ratelier et un arbre de décision temporaire.

Par défaut, lorsqu'un joueur de ce type est créé, il est initialisé avec le niveau 'facile' Son nom est composé du mot Robot suivi d'un numéro aléatoire.

Les méthodes définies sont :

- createWord : elle permet de créer une liste de mots possibles et retourne celui qui a été choisi
- selectResultWord : elle permet de sélectionner un mot parmi ceux qui sont probables selon le niveau
- checkWordsPosition : elle permet de vérifier la position des mots. Lorsqu'un mot ne peut être placé, il est supprimé de la liste des mots possibles.
- makeWords : elle parcourt l'arbre temporaire et crée autant de mots qu'il y a de parcours.
- getAllPositionOfALetter : elle permet de récupérer un tableau de lettre en fonction du caractère recherché.

Les méthodes createWord et makeWords étant les principales et les plus complexes, seul le principe de fonctionnement du joueur machine va être exposé dans sa globalité à l'aide d'un exemple.

Nous avons connaissance de trois composantes :

1. les lettres de la main du robot
2. les lettres du plateau
3. un dictionnaire de mots autorisés

Le robot va parcourir le dictionnaire. Pour chaque mot, il va créer une liste de déplacements possibles selon son niveau (expliqué ci-après). Lorsque la fin du dictionnaire sera atteinte, il va sélectionner un mot dans cette liste en fonction de son niveau de jeu :

- En mode facile, il va simplement effectuer un tirage aléatoire sur la liste de mots. Cependant, la liste ne contient que des mots dont la longueur est inférieure à six lettres.
- En mode normal, il sélectionne le plus long mot dans la liste. Néanmoins, cette liste ne contient pas des mots dont la longueur est supérieure à huit lettres.
- En mode expert, il sélectionne le mot qui rapporte le plus de points.

Pour atteindre ceci, le robot crée un arbre temporaire des déplacements, horizontaux ou verticaux, autorisés.

Voyons ceci, sur un exemple.

Si nous avons le plateau de jeu suivant :

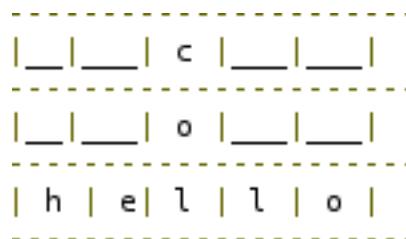


FIG. 3.1 – Plateau de jeu

Le robot a la main suivante : 'abcdo' et le dictionnaire contiendrait : 'aback, doc'...

Nous allons commencer par créer une liste de toutes les lettres pour le premier caractère du mot : 'a'. Il n'y a qu'une seule lettre et elle se trouve dans la main du robot. Cela signifie que nous pouvons en théorie créer le mot. Nous allons poursuivre en créant un arbre temporaire ayant pour racine cette lettre soit 'a'. Nous prenons le caractère suivant qui est 'b', nous avons qu'une seule lettre et elle est également dans la main du robot. Notre arbre ressemble donc à ceci : Maintenant, on passe au caractère suivant : 'a'. Puisque nous n'avons pas de lettre

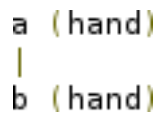


FIG. 3.2 – Arbre temporaire :aback

(celle de la main du robot est déjà utilisée), nous arrêtons la recherche et passons au mot suivant. Nous prenons le mot *doc*. Le premier caractère est dans la main du robot. Nous avons trois caractère 'o' : sur le plateau en position 23 (ligne 2, colonne 3) et en position 35 (ligne 3, colonne 5). Nous avons également deux caractères 'c' : une sur le plateau en position 13 et l'autre dans la main du robot. Notre arbre ressemble donc à ceci :

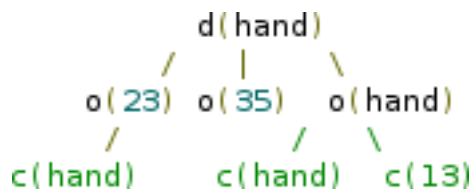


FIG. 3.3 – Arbre temporaire : doc

Après, nous pouvons sélectionner les positions valides du mot 'doc' : d(hand) o(23) c(hand) ; d(hand) o(hand) c(13). Le mot d(hand) o(hand) c(hand) n'est pas valide car nous devons utiliser au moins un caractère situé sur le plateau de jeu.

A noter que durant la création de l'arbre temporaire, nous vérifions également que la position du mot est correct. Lorsque le caractère suivant est sur le plateau, nous devons vérifier que tous les caractères précédents peuvent être placé sur le plateau sans conflits.

3.4.2.4 Problèmes rencontrés

Lors de l'intégration, le plus gros problème rencontré fut la vérification du positionnement des lettres de la liste des mots que le robot pouvait jouer. En effet, la matrice contenant le plateau, débutait ses indices à 1. De plus, le robot, utilisant le plateau sous forme d'un tableau à une seule dimension pour permettre une recherche plus rapide d'une lettre à une position donné. Pourtant ce ne fut pas le cas, ce fut même assez laborieux.

Initialement, le robot a un niveau facile, les niveaux normal et expert sont implémenté, bien que ce dernier ne le sois qu'à moitié : il manque le calcul de points d'un mot en fonction de son placement probable sur le plateau.

De plus, lorsque le robot cherche à placer un mot, il ne regarde que la ligne ou la colonne, selon que le mot sera positionné horizontalement ou verticalement, où est la lettre appartenant à la fois au mot et au plateau.

3.5 ANT : Exécuteur de tâches

ANT est l'équivalent du makefile en C. A travers diverses cibles, on peut effectuer différentes actions. Regroupées dans le fichier `build.xml`, elles sont définies en tant que tâches.

Ainsi, nous avons actuellement trois cibles importantes :

- `make` : phase de compilation avec création d'un journal et d'une archive `.jar`
- `run` : phase de lancement d'un serveur et de deux clients. Utile pour une démonstration entre deux joueurs humains
- `clean` ou `cleanAll` : phase permettant d'effacer les dossiers créer

Pour connaître les différentes possibilités offertes, tapez la commande suivante dans un terminal ouvert dans le dossier contenant le fichier `build.xml` : `ant`. Pour plus d'informations, se référer à au fichier.

Conclusion

Bien qu'intéressant, de par ses multiples aspects ludiques, ce projet s'est révélé être d'une complexité que nous avions pas soupçonné. Le sujet d'un jeu de société est déjà en lui même un projet qui attire ainsi que le fait de pouvoir mettre en pratique nos enseignements d'une manière 'amusante'.

Le sujet étant assez vaste : allant de l'interface graphique à la gestion d'une intelligence artificielle, en passant par le concept des applications 2/3-tiers avec notamment les clients-serveur, sans oublier une réflexion algorithmique importante pour l'implémentation de jeux de mots : ici pour le dictionnaire, comment le modéliser ? sous forme d'arbres, automates...

Un projet avec un large choix d'implémentations et néanmoins peu de temps, nous avons voulu trop vite étoffer toutes les fonctionnalités possibles et ainsi nous avons peu à peu dériver de notre planning initial.

Malgré tout cela et les problèmes, peut être secondaires, que nous avons rencontrés tel que l'aspect graphique ou encore la communication client-serveur, nous avons obtenu un résultat qui semble tout à fait correct au vu du temps disponible et de la complexité du projet en lui-même. Nous sommes conscients que beaucoup de travail reste à faire afin de l'améliorer

Pour respecter un planning établi, mieux vaut séparer le travail en petite modules que de se pencher tous sur un même problème..

Bibliographie

- [1] S.A GORDON, *A Faster Scrabble Move Generation Algorithm*
- [2] B. SHEPPARD, *World Championship Caliber Scrabble*
- [3] R. Hightower N. Lesiecki, *Java Tools for Extrem Programming*
- [4] V. MASSOL T. HUSTED *JUnit in Action*, Edition Manning
- [5] E.M BURKE B.M COYNER *Java Extrem Programming Cookbook*, Edition O'Reilly
- [6] J. TILLY E.M BURKE *ANT, The Definitive Guide*, Edition O'Reilly
- [7] Cours de LP Multimedia (ANT), Nice Sophia Antipolis (06), 2006/2007
- [8] Dictionnaire, www.fisf.net,
[Wikipedia : Officiel du jeu de scrabble](#)
- [9] Cours de Graphes de L2 et d'algorithmique de L3 (pour la recherche de mot dans le dictionnaire en particulier)
Université Jean Monnet - Saint Etienne (42) 2006 á 2008
- [10] Lettre et points associés [Wikipedia : Lettres de Scrabble](#)
- [11] Cours sur le Drag'n'Drop en Java [Cours de Paris](#)
[Javaworld : explication sur un exemple](#)
- [12] Image disponible sur www.jeuxvideo.com,
[Comment faire un SplashScreen](#)