

Table des matières

Table des matières

**Introduction2**

**Règles du jeu 3**

Tetris3

Boggle3

TetraWord4

Déroulement du jeu4

Les bonus4

**Conception du jeu4**

Conception5

User-case5

UML6

Ergonomie et design6

Developpement7

Architecture7

Architecture du projet7

Architecture JAVA8

**Les problèmes rencontrées et les solutions apportées13**

**Extensions15**

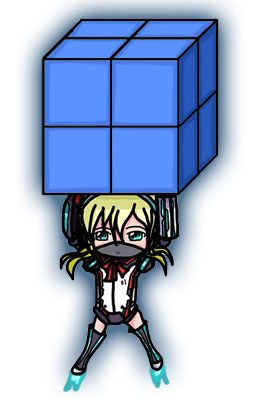
**Le résultat16**

**Conclusion19**

Introduction

Dans le cadre de notre formation, nous sommes amenés à réaliser un jeu réaliser dans le langage JAVA appelé TetraWord. Ce jeu est un mélange entre Tetris et Boggle. En effet, chaque cube comprend une lettre, ce qui permet ainsi de former des mots.

Dans une première partie, nous allons rappeler les règles du jeu. Dans une seconde partie, nous allons parler de la conception et développement du jeu. Enfin, dans une dernière partie, nous allons présenter l’extension et le résultat obtenu.



Règle du jeu

# Tétris

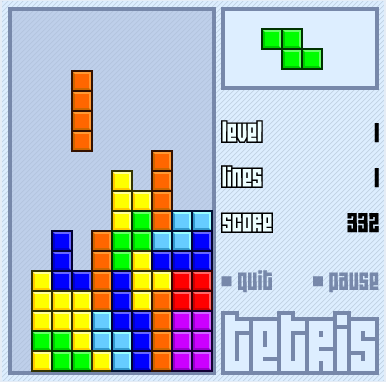


Figure 1 : Tetris

Tetris est un puzzle game. En effet, le but du jeu est de former des lignes à l’aide des pièces de couleurs et des formes qui tombent. Il est possible de tourner ces formes de 90 degrés la forme. Ces dernières sont formées de quatre pièces. Lorsqu’une ligne est complète, elle disparait. Le jeu ce termine lorsque l’écran est remplit.

# Boggle



Figure 2 : Boggle

Le but du Boggle est de trouver un maximum de mot dans une grille de lettre, pouvant être formés à partir de lettre adjacentes (horizontalement, verticalement ou en diagonale) dans un temps à partie.

# TetraWord



Figure 3 : Le logo

## Déroulement du jeu

La partie commence comme un jeu de Tetris classique. Cependant, chaque pièce de la forme contient une lettre. Une fois qu’une ligne est remplie, le mode anagramme s’active. Le joueur doit trouver un mot dans la ligne qui comporte au moins 70 % du meilleur anagramme. Si cette condition est remplie, la ligne est détruite. Tout comme Tetris, lorsque la grille du joueur est remplie, ce dernier a perdu.

## Les bonus

Le jeu contient également des bonus qu’il est possible d’activer :

* Le bonus **échange** : Ce bonus permet d’échanger les plateaux des deux joueurs.
* Le bonus **retournement** : Le plateau du joueur d’adverse se retourne. Les formes vont donc de bas en haut.
* Le bonus **malus** qui basse les points du joueur adverse
* Le **bonus** qui ajoute des points au score

Conception du jeu

Nous avons départagé le projet en plusieurs étapes :

* L’étape de **conception** : elle consiste à la réflexion sur l’architecture, les classes.
* L’étape de **design** : elle consiste à la réflexion sur l’ergonomie et la réalisation du design pour le jeu.
* L’étape de **développement** du jeu : elle consiste en la réalisation du jeu. Donc coder tous le jeu en lui-même.

# Conception

Avant de commencer le développement du jeu en lui-même. Nous avons choisi d’avoir une étape de conception. De cette façon, nous partons tous de la même base. Ainsi, nous savons tous, les fonctionnalités et variables utilisés et implémentés de chaque classe. De cette façon, il est plus facile de coder en simultané puisque nous connaissons les liens entre chaque classe. De plus, cette étape permet de gagner du temps au moment du développement. Pour cela, nous avons d’abord réalisé un user-case avant de faire un diagramme UML.

## User-case

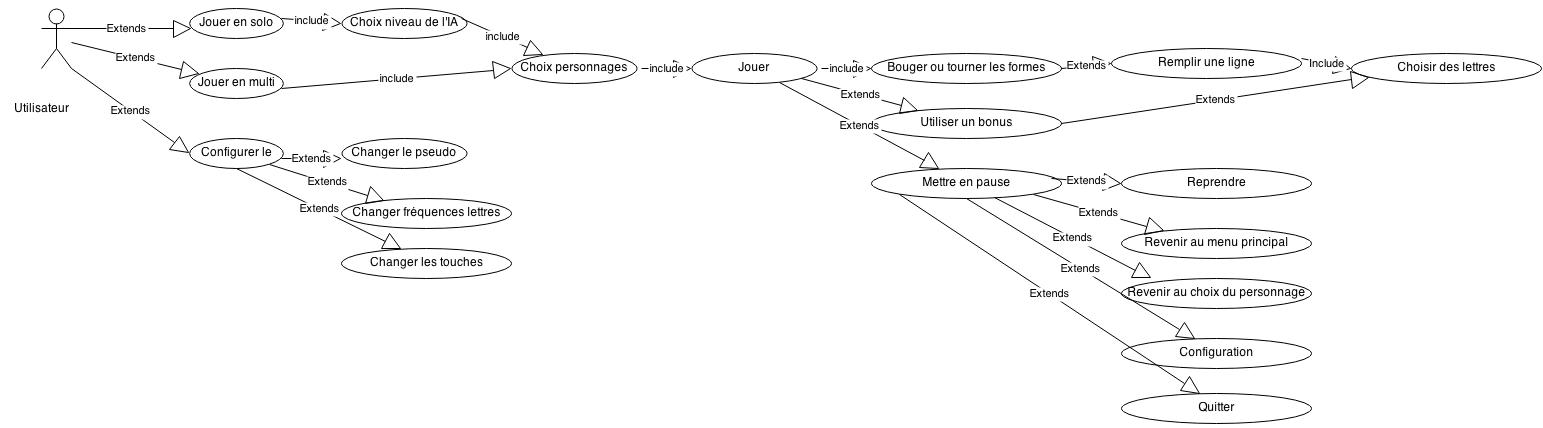


Figure 4 : User-case Utilisateur

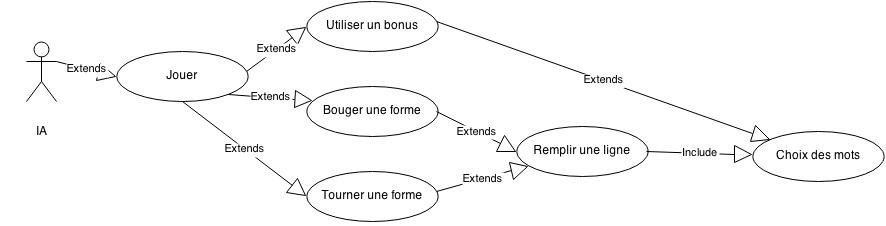


Figure 5 : User-case IA

## UML



Figure 6 : UML simplifié

# Ergonomie et design

En ce qui concerne l’ergonomie et le design, nous avons principalement utilisés l’user-case pour définir les différents types d’affichage à réaliser. Nous avons également fait des recherches sur l’existant afin garder les points forts au niveau ergonomie.

Nous avons choisi de refaire le design de zéro, c’est-à-dire sans reprendre des éléments existants. De cette façon, nous pouvons affirmer que nous avons entièrement réalisé le jeu. Nous avons donc fait différents test avant de se mettre d’accord sur un design particulier.

Nous sommes ainsi arrivés au résultat suivant :



Figure 7 : Design TetraWord

# Développement

## Architecture

### L’architecture du dossier :

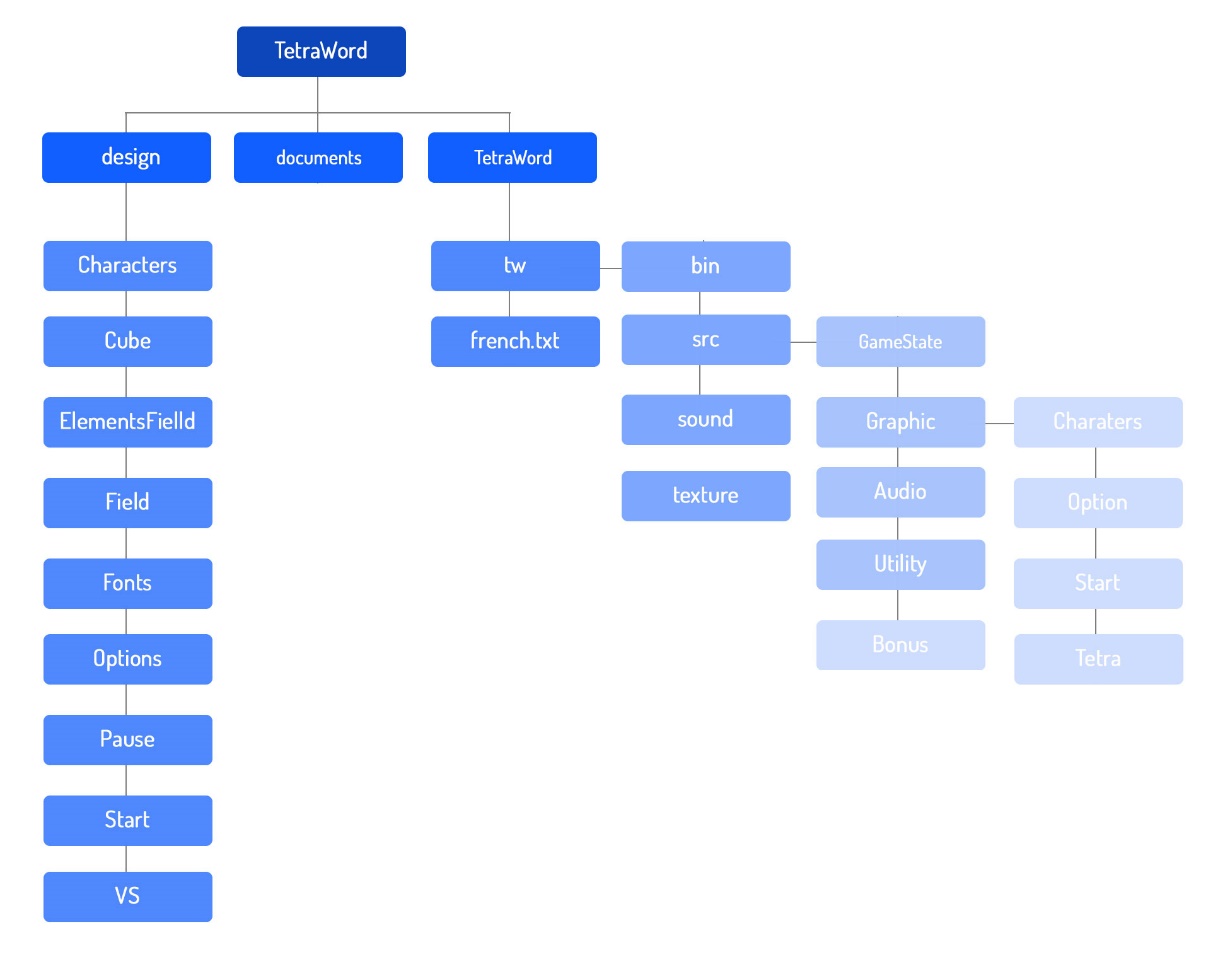


Figure : Architecture du projet

Notre projet est rangés dans un dossier « TetraWord » dans lequel nous avons les dossiers suivant :

* **Design** : contient l’ensemble des éléments du design avant d’être intégré.
  + Characters : contient les images personnages
  + Cube : Contient à la fois les images les cube et les bonus
  + Elements field : Contient les images des éléments du terrain (la grille par exemple)
  + Field : Contient les images des fonds des terrains
  + Fonts : Contient la font utilisé
  + Options : Contient les images concernant la partie configuration du jeu. Ce dossier est aussi divisé en sous dossier correspondant aux différentes parties de la configuration
  + Pause : Contient les images concernant l’affichage du menu de pause
  + Start : Contient les images concernant l’affichage du menu principal
  + VS : Contient les images du VS dans le tetraword.
* **Documents** : contient l’ensemble des documents (rapports, diagramme, etc)
* **Tetraword** : contient les élements java.
  + Tw: contient tous les élements java
    - Bin : contient les exécutables
    - Src : contient les class java
      * Graphic : contient toute la partie affichage. Ce dernier est également séparer par des dossiers correspondant au différent affichage du jeu.
      * Game state : contient toute la partie GameState
      * Utility : contient toutes la partie fonctionnalités du jeu
    - Sound : Contient les bruitages et son en wav.
    - Textures : contient toutes les images des textures. Ce dossier contient également des sous-dossiers correspondant aux différents affichages (menu principal, tetraword, configuration, etc) qui sont eux même séparer selon les catégories que l’on retrouve dans design.
  + french.txt : il s’agit du dictionnaire français

Nous avons choisi de séparer le design de la texture. En effet, le dossier texture contient seulement les différentes textures en images alors que dans le fichier design, nous avons les différents essaies ainsi que le fichier d’origine, c’est-à-dire un fichier photoshop (psd). En effet, de cette façon, nous pouvons modifier le design du jeu et récupérer les anciennes versions. De plus, lorsqu’une personne souhaite seulement jouer au jeu, il n’a pas besoin de tout le design. Il a seulement besoin des textures. Ainsi, il utilisera seulement le dossier TetraWord qui est donc beaucoup plus léger.

### L’architecture JAVA :



Figure : Architecture JAVA

Nous avons 11 packages :

* **Package par défaut** : Contient l’Engine. C’est ce que l’on compile pour faire fonctionner tout le jeu. Il créer ainsi les instances.
* **Audio**: Permet de créer des fichiers audio et de les lire.
* **Game State** : contient les différents Game States, c’est-à-dire les différents états dans lesquels peut se trouver le jeu
  + Anagramme : Il s’agit du GameState pour Anagramme
  + Game : Cette classe contient la pile des GameState. Ainsi lorsqu’on change de State, on empile ou dépile la pile.
  + Game State : Il s’agit de la class mère dont tous les GameState hérite. Ainsi elle détermine les variables et fonction générique des GameState.
  + Tetris : Il s’agit du GameState pour le jeu en lui-même
  + Worddle : Il s’agit du GameState pour le mode Worddle
* **Graphic** : contient les différents éléments qui permettent d’afficher les choses récurrentes. Nous avons choisi d’utiliser des classes car elles permettent d’afficher plus rapidement les éléments de bases (par exemple les boutons avec des images). De plus, il est plus simple de récupérer les tailles des éléments de cette façon.
  + ButtonTetra: Elle contient les variables et fonctions pour afficher un bouton dans notre frame et hérite de JButton
  + Frame : Permet de créer la frame général qui contient tous les autres éléments graphiques. Elle hérite de JFrame.
  + PanelBase : Elle contient les variables et fonctions pour afficher un panel dans notre frame et hérite de JPanel
  + Test : Elle a été utilisée lors de la phase de développement pour des tests graphiques.
  + TetraComponent : Elle contient les variables et fonctions pour afficher un label dans notre frame et hérite de JLabel
* **Graphic.character** : contient les éléments pour afficher la partie concernant le choix du personnage.
  + CharacterLabel : Permet de créer les labels qui permettent d’afficher les personnages de selection dans la frame. Elle hérite de TetraComponent.
  + KeyListenerCharacter : contient les keyListener qui permettent de savoir sur quel bouton du clavier on a appuyé et d’effectuer une action en fonction de cela. Elle hérite de KeyListener. Ici la selection du personnage.
  + KeyListenerSpace : contient les keyListener qui permettent de savoir sur quel bouton du clavier on a appuyé et d’effectuer une action en fonction de cela. Elle hérite de KeyListener. Ici le changement de state.
  + PanelCharacter : Permet d’afficher l’ensemble de la partie choix du personnage. Elle hérite de PanelBase et contient les différents éléments (PresentationPlayerComponent, etc) de l’affichage.

Ici nous avons choisi de séparer les deux Listener car chaque Listener est attaché à un JComponent et ces deux Listener appartiennent à deux JComponent différents.

* + PersoSelectComponent : Permet de créer les components qui permettent d’afficher le carré de sélection qui permet de s’avoir quel personnage a été sélectionner. Elle hérite de TetraComponent.
  + PresentationComponent : Permet de créer les components qui permettent d’afficher le grand carré de présentation du personnage a été sélectionner. Elle hérite de SelecteurExtends .
  + PresentationPlayerComponent : Elle permet de mettre à jour l’affichage en fonction du personnage choisi.
  + SelecteurComponent :
  + SelecteurExtends : Il s’agit du carré qui permet d’indiquer quel personnage le joueur a choisi.

Nous avons choisi de séparer SelecteurComponent et SelecteurExtends car de cette façon, il est plus facile de dupliquer un selecteur une fois que le premier est créé.

* **Graphic.option** : contient les éléments qui permettent d’afficher la partie concernant la configuration du jeu.
  + PanelOption : Permet d’afficher l’ensemble de la partie configuration de base donc ce qui est commun à l’affichage des différentes parties de la configuration. Elle hérite de PanelBase.
  + RoundTextIU : Permet de créer des JTextField à bord rond.
  + TetraComboBox : Permet de créer les menus déroulants.
  + TextFieldTetra : Permet de créer des TextField et de savoir si le texte à l’intérieur a été changé ou non.
* **Graphic.option.letters** : contient les éléments qui permettent d’afficher la partie configuration particulièrement la partie concernant la configuration de la fréquence des lettres.
  + LetterArray : Permet de créer tous les components avec toutes les lettres de l’alphabet. Elle contient donc 26 LetterComponent.
  + LetterComponent : Permet de créer des component qui contiennent une lettre et un TextField qui va avec.
  + LetterUnderPanel : Permet d’afficher l’ensemble de la partie configuration de la fréquence d’apparition des lettres. Elle hérite de PanelBase.
* **Graphic.option.option** : contient les éléments qui permettent d’afficher la partie configuration particulièrement la partie concernant la configuration générale du jeu.
  + JoypadComponent : Permet de créer les éléments d’affichage pour la configuration des touches. Elle permet également de mettre à jour les touches. Elle hérite de TetraComponent et contient des KeyComponent.
  + KeyComponent : Permet de créer un élément qui permet la configuration des touches (La touche et l’input qui va avec).
  + NameComponent : Permet de créer les élements d’affichage pour les autres éléments de la configuration, notamment le changement de pseudo. Elle permet également de mettre à jour le nom.
  + OptionUnderPanel : Permet d’afficher l’ensemble de la partie configuration de base (la configuration des touches, le choix du pseudo, etc). Elle hérite de PanelBase.
* **Graphic.start** : contient PanelAccueil  qui contient elle-même les éléments qui permettent d’afficher la partie concernant le menu principal.
* **Graphic.tetra** : contient les éléments qui permettent d’afficher la partie concernant le jeu, le tetraword.
  + BonusArrayComponent : Permet de créer les éléments d’affichage de la liste des bonus à afficher. Elle contient un tableau de BonusComponent.
  + BonusComponent : Permet de créer les éléments d’affichage d’un bonus.
  + FieldComponent : Permet de créer une grille dans lesquelles on retrouve les formes (shapes) et cases. Elle permet de gérer l’affichage des briques.
  + KeyListenerTetra : contient le listerner et la fonction pour mettre les jeux en pause.
  + MouseListenerTetra : contient le listener qui permet de
  + MouseMotionListenerTetra : C’est une classe test pour tester le mouseMotionListener que nous n’avons pas utilisé.
  + PanelTetraWord : Permet d’afficher l’ensemble du jeu lui-même.
  + PauseComponent : Permet de créer les éléments d’affichage et les listeners pour l’affichage de l’écran de pause.
  + ScoreComponent : Permet de créer les éléments d’affichage pour le score.
  + VersusComponent : Permet de créer les éléments d’affichage pour la partie VS.
* **Utility**: contient toutes les classes qui permettent au jeu de fonctionner.
  + AlphaTree : correspond à l’arbre ternaire lexicographique qui est rempli au début du programme
  + Anagramme : Cette classe n’est pas utilisée
  + Board : Permet de créer un terrain et contient la taille, les cases du terrain. Chaque joueur a son terrain.
  + Case : Cette classe permet de créer une case et de connaitre ces propriétés (la couleur, sa position, etc)
  + Chrono : Cette classe permet de gérer un chrono notamment pour avoir un temps de latence pour l’apparition d’une forme.
  + Configuration : Contient les configurations du jeu tel que la vitesse, la fréquence d’apparition des bonus, etc.
  + Dictionary : Contient le dictionnaire avec la liste des lettres, la liste des lettres pondérés et l’arbre ternaire lexicographique.
  + laTetris : IA
  + Letter : Cette classe permet de créer une lettre et connaitre ces propriétés (fréquence)
  + Player : Permet de créer un player et de connaitre ces propriétés (nom, etc).
  + Shape : Permet de créer une forme ou brique.
  + Square : Permet de savoir si une case à des liens avec d’a
* **Utility.bonus** : contient les bonus.
  + BonusScore : contient les éléments nécessaires pour le bonus qui permet d’ajouter des points à son score.
  + BonusTetra : contient la structure et les fonctions génériques de chaque bonus. Il s’agit du parent de tous les bonus.
  + ExchangePlateau : contient les éléments nécessaires pour le bonus qui permet d’échanger les plateaux des deux joueurs.
  + MalusScore : contient les éléments nécessaires pour le bonus qui permet de retirer des points au score de l’adversaire.
  + ReturnMalus : contient les éléments nécessaires pour le bonus de retournement de plateau.

On peut remarquer que la structure a quelque peu changé. En effet, nous n’avions pas pris en compte SWING. Cependant, on retrouve le GameState ainsi que les différentes classes qui se trouve dans le package Utility.

Les problèmes rencontrées et solutions apportés

Nous avons rencontrées des problèmes lors de la phase de développement. En effet, nous avions eu un problème concernant la rotation car la façon dont sont codées les formes font que lorsqu’on tournait l’élément, ce dernier s’arrêtait.

De plus, lorsqu’un élément tournait, nous choisissons un pivot qui correspond à une case autour duquel il va tourner. Cependant, le carré doit tourner à partir du centre.

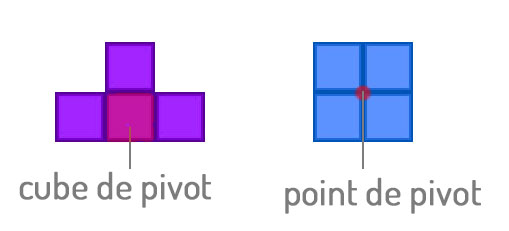


Figure : Schéma pivot

Solution :

Nous avons choisi de simplement faire tourner les lettres, donc changer les lettres dans les cubes car le cube ne change pas de forme lorsqu’on le tourne.

Nous avons également eu des problèmes lors de l’utilisation de SWING. En effet, lors de l’affichage des éléments, ce dernier prend en compte des bordures.

Solution :

Nous avons donc pris en compte les bordures lors du positionnement des textures en faisant des calculs simples.

Pour le projet, nous avons utilisé git pour centraliser les données. Ce qui nous permet ainsi de travailler simultanément sur le projet sans gêner les autres de l’équipe. Cependant, nous avons également eu quelques soucis au moment d’envoyer les données car elles peuvent entrer en conflit. De plus, certains membres de l’équipe utilisent pour la pour la première fois git.



Solution :

Pour que le projet ne soit pas « détruit », nous avons choisi de travailler chacun sur sa branche et faire des « merges » ou fusion en cas de besoin. De plus, nous avons choisi de d’envoyer les données ou « push » à chaque fois que l’on fait des changements pour pouvoir récupérer plus facilement les anciennes versions en cas d’erreur.

Extensions

Nous avons apportés quelques améliorations au programme afin d’augmenter le côté fun du jeu. Ainsi, nous avons ajouté des personnages au jeu. Au début de la partie, le joueur doit choisir son avatar (un panda (don’t say no to panda), un robot, un ninja ou un démon). Chaque avatar est caractérisé par un terrain, et par une attaque spéciale.

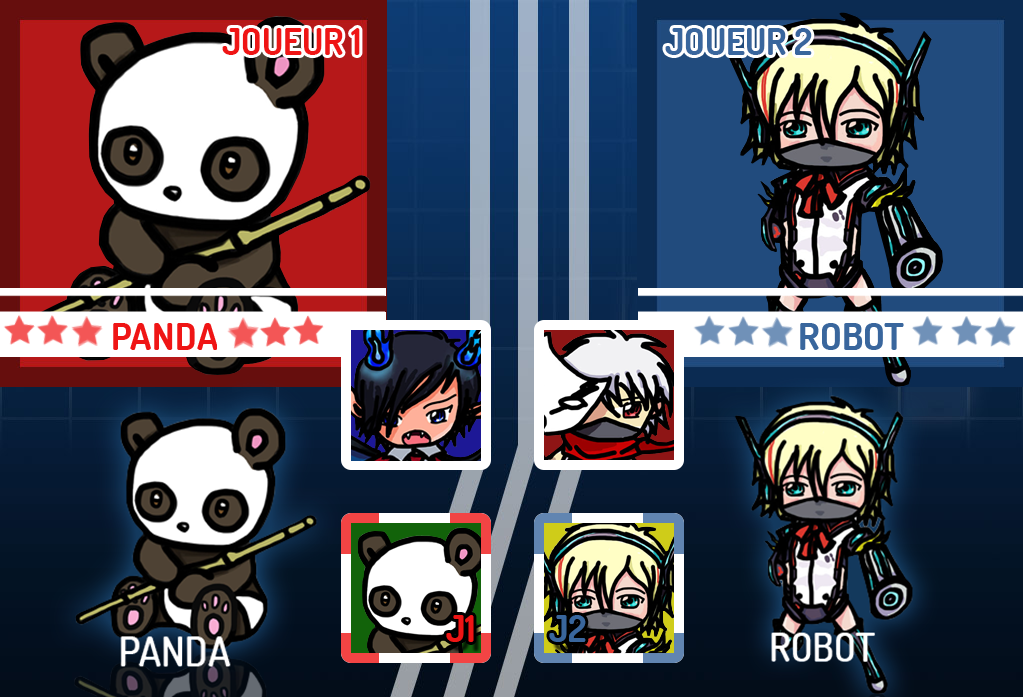


Figure 11 : Partie choix du personnage

Lorsque la partie est lancée, le choix du terrain se fait donc en fonction des personnages choisi. Ainsi, une fonction fait le choix aléatoire entre un terrain par défaut et les terrains liés aux personnages choisis.

Une fois dans le jeu, à chaque fois que le joueur gagne des points, un jauge se rempli. Il s’agit de la jauge spéciale du personnage. Lorsqu’elle est remplie, le joueur peut activer l’attaque spéciale du personnage. Cette attaque peut correspondre à un bonus. Lors de l’activation de l’attaque, une bande au milieu de l’écran apparait avec la tête du personnage en grand avant l’activation de son attaque spéciale. Nous n’avons malheureusement pas eu le temps d’implémenter cette partie.

Le résultat

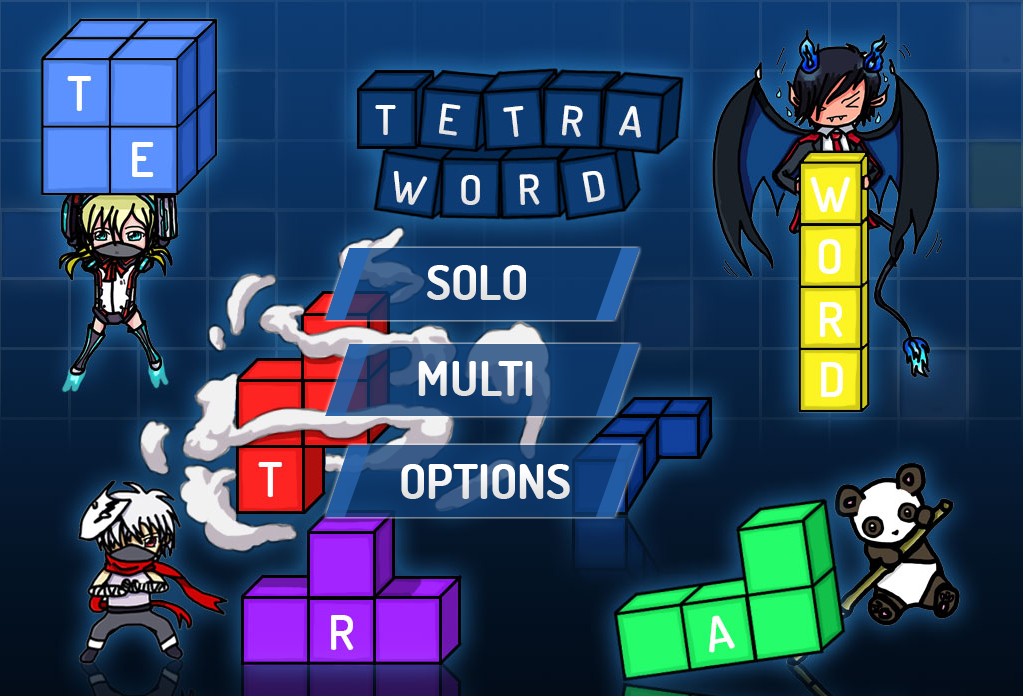


Figure 12 : Menu principal



Figure 13 : Menu de configuration



Figure 14 : Configuration de la fréquence d'apparition des lettres



Figure 15 : Choix du personnage

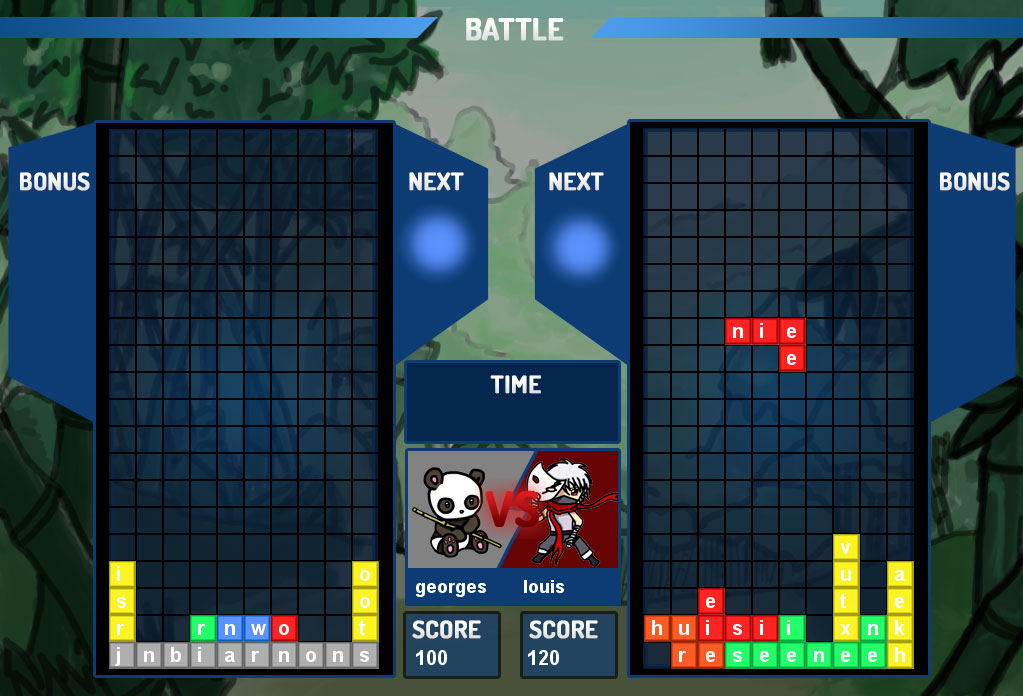


Figure 16 : TetraWord

Comme on peut le voir sur ce screen, nous n’avons pas eu le temps d’afficher le temps, les prochaines formes et les bonus, bien que la plupart d’entre eux soit implémenté.

Conclusion

Ce jeu fut très intéressant à concevoir et à développer. En effet, cela nous a permis de développer nos compétences en JAVA mais aussi dans la conception, réflexion avant le développement du jeu pour que la structure du jeu soit la plus optimale possible. Ainsi nous avons pu avoir une réflexion structurés tout au long du processus. Ce jeu constitue un plus dans notre portfolio.

Il est également intéressant d’observer les différents points sur lesquelles nous avons eu des problèmes, notamment sur l’UML, car en remarquant nos erreurs nos avons moins de chance de les reproduire.

De plus, ce projet nous a également permis de mettre en application les acquis du cours et expérimenter nos compétences.

Le langage JAVA est un langage structuré donc l’ensemble des classes fournis permet de créer simplement et rapidement des applications.

