IMAC2 2014-2015

Thibault Fievet
Elena Loyatho
Laetitia Nanni

Introduction

Ce projet_s'inscrit dans le cadre de notre deuxième année à l'IMAC à l'Université Paris-Est Marne-la-Vallée. Il a pour but de mettre en pratique les connaissances acquises pendant les cours de java dans un projet concret, le Tetraword.

Ce rapport présente notre travail effectué dans le développement du Tetraword. D'abord, nous allons rappeler le fonctionnement du jeu. Puis, nous allons présenter l'architecture du projet et comment nous nous sommes pris pour développer le jeu. Enfin, nous allons faire un point sur la gestion de projet.

1. Présentation du jeu

1.1. Présentation

Le Tetraword est un jeu dérivé du célèbre jeu Tetris. Le tétris est un jeu qui se présente sous la forme d'un plateau vertical sur lequel glissent des éléments de formes diverses composées de briques qui s'empilent les unes sur les autres. L'objectif est de former des lignes entières qui peuvent ainsi disparaitre.

Le Tétraword rajoute une dimension de jeu sur les mots. Chaque briques comporte une lettre. Lorsque l'on passe au mode anagramme ou worddle, on a la possibilité de supprimer des briques en composant des mots.

C'est un jeu multi-joueur, nous avons la possibilité de jouer seul, contre l'ordinateur ou même jusqu'à 4 joueurs. Le jeu de base fonctionne avec des tétraminos, mais nous avons également la possibilité de paramétrer le jeu et de jouer avec des pentominos ou encore des hexominos.

1.2. Modes de fonctionnement

Le jeu comporte 3 modes de fonctionnement :

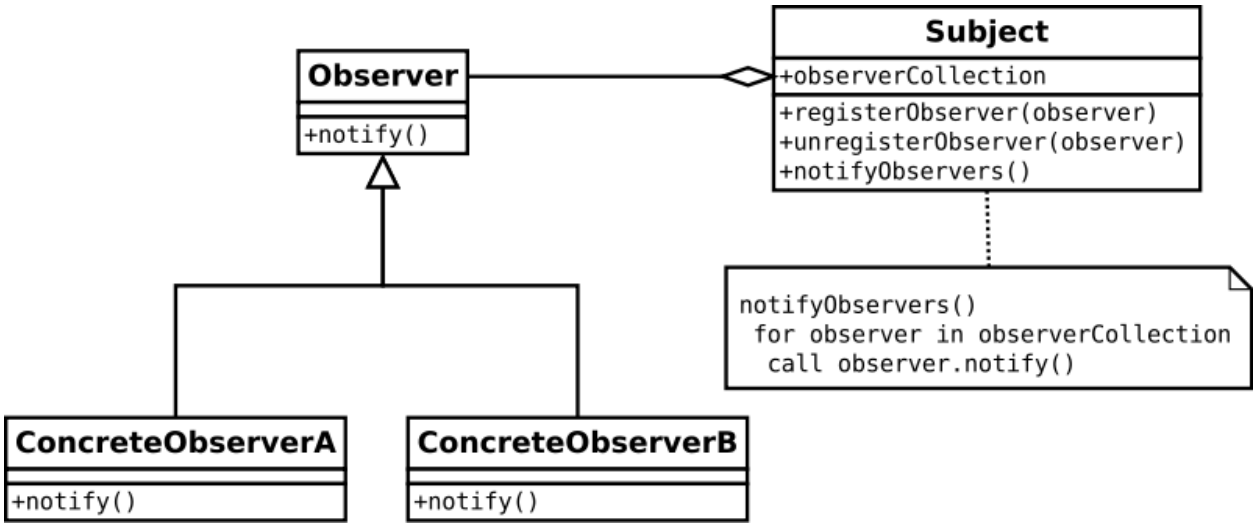
- **mode tetris** : les éléments de formes diverses composées de briques s'empilent les unes sur les autres. A chaque tour, une forme apparaît au milieu et en haut de l'écran et descend. Le but est de créer des lignes complètes qui se supprimeront.
- **mode worddle** : Le joueur doit trouver un maximum de mots valides à l'aide des briques présent sur le plateau
- **mode anagramme** : ce mode est activé quand au moins une ligne complète apparaît sur le plateau. Il faut alors trouver pour chacune des lignes le mot du dictionnaire utilisant le maximum de lettres de la ligne.

2. Architecture globale du projet

Dans cette partie, nous allons présenter l'architecture globale du projet. Nous avons divisé en 4 paquetages distincts le projet. Les classes sont reliées entre elle grâce au pattern Observateur. Nous allons d'abord faire un rappel de fonctionnement de ce patron de conception, car nous le retrouvons régulièrement dans notre implémentation du jeu. Puis, nous allons présenter la structure générale du projet.

2.1. Fonctionnement du pattern Observateur

L'observable, dans le schéma ci-dessous Observer, envoie des notifications aux objets qui l'observent dès qu'un certain événement se produit. Ensuite, les observateurs déclenchent les actions adéquates. Par exemple, un bouton sera l'observable, dès qu'un clic se produit, il prévient les objets qui le suivent pour que eux fasse l'action appropriée.



2.2. Architecture du jeu

Nous avons créé une architecture de jeu sous forme de paquetages de cette façon :

```
.
├── MANIFEST.MF
├── README.md
├── TetraWord.jar
├── bin
│   ├── *.classe
├── build.num
├── build.xml
├── data
│   ├── conf
│   │   ├── hexaminos.xml
│   │   ├── letters.xml
│   │   ├── letters_simple.xml
│   │   ├── liste_francais.txt
│   │   ├── liste_francais_simple.txt
│   │   ├── pentominos.xml
│   │   ├── tetrominos.xml
│   │   ├── triominos.xml
│   │   ├── words.txt
│   ├── font
│   │   └── Minecraftia.ttf
│   ├── img
│   │   └── *.jpg
│   ├── lib
│   │   └── jdom.jar
│   └── sounds
│       └── *.wav
└── doc
    └── rapport.pdf
    └── LISEZMOI.txt
└── src
    ├── Main.java
    ├── control
    │   ├── AnagramListener.java
    │   ├── Bot.java
    │   ├── Keyboard.java
    │   ├── Player.java
    │   ├── SoundEffect.java
    │   └── WordleListener.java
    ├── menus
    │   ├── GameScreen.java
    │   ├── Gameboard.java
    │   ├── Interface.java
    │   ├── Menu.java
    │   ├── MenuInGame.java
    │   ├── ObservableButton.java
    │   ├── Options.java
    │   └── TFont.java
    ├── modificators
    │   ├── Modificator.java
    │   └── Reserve.java
```

```
└── polyominos
    ├── Brick.java
    ├── Letter.java
    ├── Polyomino.java
    ├── PolyominoList.java
    └── PolyominoPattern.java
```

Dans le fichier src/ il y a 4 *paquages*. Le paquage “control” gère les interactions de l’utilisateur avec le jeu, c’est-à-dire qu’il a les mécaniques de jeu. Le paquage “menus” comporte toutes les vues de l’application. Enfin, les paquets “modificators” et “polyominos” comporte respectivement toutes les fonctions necessaire à la création et à la vie des modificateurs et des polyominos.

3. Développement du jeu : stratégies et difficultés

Dans cette partie nous allons présenter notre démarche de développement dans chacune des parties du projet, en expliquant les difficultés rencontrés et les solutions que nous avons apporté.

Interface, style du jeu

La mise en place, à échelle d’un projet, de la libraire **swing**, extrêmement difficile à appréhender au début. Cette étape a été réalisée indépendamment du reste du projet dans un premier temps, puis le tout a été fusionné.

Chaque type de vue correspond à une classe et un système d’encapsulation permet de rendre les éléments dans leur ensemble. Selon les besoins de placement des éléments, nous avons utilisé plusieurs *layout managers* dont les plus importants sont le BoxLayout et le BorderLayout.

Le style de l’application peut se changer facilement. Il est possible de modifier les images de fond des JPanel et l’importation de nouvelles font.

Les effets sonores sont regroupés dans une énumération dans la classe SoudEffect. Les fichiers .wav sont gérés grâce à la class Clip fourni par java. Il est possible d’agir sur le volume et de joueur le son en boucle ou non selon les besoins.

Tetra….

Les **polyominos** sont générés à partir de **fichiers xml** dans lesquels se trouvent les coordonnées x et y de chaque brique qui les composent. On y trouve également leur couleur.

Ces paramètres, une fois importée dans le jeu, servent à fabriquer aléatoirement des polyominos à la demande du joueur lorsque celui-ci en a besoin. Une liste commune à tous les joueurs est ainsi générée pour garantir un ordre précis de défilement des pièces entre les joueurs, par soucis d’équité.

La première difficulté concernant les polyominos vient de leur placement, puisqu'ils doivent parfaitement s'imbriquer les uns aux autres. Au final, ce soucis nous a poussé à réaliser une **intelligence artificielle** pour l'adverssaire plutôt décevante, les contraintes étant trop nombreuses pour obtenir un résultat dans des délais aussi courts.

La plus grosse difficulté réside dans la **suppression** des pièces, car il faut d'abord tester si une ou plusieurs lignes ont été formées, les stocker pour ensuite les supprimer. Enfin, il ne faut pas oublier de faire descendre les pièces du dessus tout en gardant la cohérence physique du monde de tetris.

...Word

Le **dictionnaire de mots** est assez important, il a donc été décidé de l'importer dans le jeu dès son démarrage, pour éviter des latences lors du commencement d'une partie. D'abord implémenter selon un algorithme de dichotomie, la recherche de mots a finalement laissé place à une table de hashage optimisée proposée par java, plus simple et performante.

Les **lettres** sont configurées dans un fichier xml, associés à leur fréquence d'apparition dans la langue française, pour permettre la réalisation de mots plus facilement. Le choix des lettres accentuées s'est longtemps posé pour finalement être concervé, ajoutant une touche de rigueur et poussant le joueur à ne pas massacrer le français.

Fonctions communes

La gestion du clavier, bien que simple, s'avère chaotique lorsque 4 joueurs doivent se partager les touches.

La sauvegarde n'a pas été implémentée à cause de bugs et du temps de développement nécessaire pour les corriger.

Modes de jeu

Le mode **tetris** a présenté des difficultés de développement quant à son confort de jeu. Ainsi, les rotations des polyominos devaient s'exécuter depuis le milieu de la pièce et si celle ci rentrait alors dans le mur ou dans une autre pièce, il devenait préférable de décaler la pièce de quelques crans sur les côtés plutôt que d'annuler la rotation, pouvant frustrer le joueur.

Le mode **anagramme** fut difficile à penser, car les éléments cliquables ne le sont pas de prime abord (ce sont de simples rectangles dessinés à l'écran). Il fallait rendre ce mode ergonomique, deux boutons ont

ainsi été ajoutés permettant d'annuler ou de valider la saisie du mot. Les lettres sont également d'une couleur différente pour se repérer.

Enfin, le mode **worddle**, qui n'est activable que depuis un modificateur arrivant de manière aléatoire sur l'écran, fut plus facile à mettre en place, car proche de l'anagramme dans son fonctionnement. Beaucoup de bugs sont en revanche arrivés avec ce mode, car celui-ci est actionnable à presque n'importe quel moment, nous poussant à revoir certains bouts du code.

Modificateurs

Les **modificateurs** arrivent aléatoirement sur l'écran. Il faut alors les attrapper pour pouvoir les utiliser. Certains nous attribuent des bonus, d'autres des malus. Certains apparaissent à des endroits difficiles d'accès, poussant le joueur à se mettre en danger. Ils sont au nombre de 8 :

- renversement de l'écran adverse 
- tremblement du plateau 
- ralentissement des chutes de pièces 
- bonus au score 
- malus au score 
- vents latéraux droite et gauche 
- worddle 

Les modificateurs bénéficient d'un thread pour s'actualiser sans gêner le déroulement de la partie.

Conclusion du développement

Les **threads** nous ont permis de gérer des ressources communes à plusieurs processus de façon à ne pas avoir de conflit, un mode multijoueur où jusqu'à 4 joueurs s'échangent des informations de jeu après les avoir modifiées.

4. Gestion de projet

4.1. Organisation du projet

Nous avons organisé notre projet en plusieurs phases. Nous avons commencé par analyser le sujet pour le comprendre et imaginer une architecture adéquate. Nous avons ensuite séparé le développement : d'un côté nous avons mis en place l'interface, et de l'autre côté le jeu. Celui-ci a été ensuite développé comme expliqué précédemment dans la partie 3. Enfin, nous avons terminé par la correction des différents bugs et la rédaction des documents.

N'ayant pas le même niveau pour le développement, les tâches n'ont pas été faites de façon forcement équitable, en effet, Thibault qui est plus à l'aise faisait plusieurs tâches le temps que Laetitia et Elena en fasse une. Nous avons donc répartie les tâches de la façon suivante : Laetitia et Elena se sont occupées des parties plus faciles à implémenter ainsi que toutes la partie interface du jeu et Thibault s'est surtout concentré sur le développement plus difficile techniquement. Le travail préalable de conception du jeu et la partie concernant la rédaction des documents a, lui, était fait par l'ensemble du groupe. Nous avons néanmoins essayé de faire en sorte que tout le monde participe à chaque phase du projet et que son ensemble soit compris par tous les membres de l'équipe.

4.2. Fonctionnalités demandées

Fonctionalités	Implémentés
Mode tetris	oui
Mode anagramme	oui
Destruction des briques	oui
Modificateur de ralentissement	oui
Modificateur de tremblement	oui
Modificateur de tempête (vent latéral)	oui
Modificateur de retournement de plateau	oui
Modificateur d'échange de plateau	non
Modificateur de bonus ou malus de score	oui
Modificateur bombe	non
Modificateur de voyage temporel	non

Modificateur d'entrée dans le mode worddle	oui
Mode multi-joueur	oui
Intelligence artificielle	oui
Mode pause	oui
Sauvegarde et rechargement de parties	non
Configurabilité du jeu	oui

Conclusion

Ce projet nous a permis de mettre en pratique sur un projet concret ce que nous avons appris lors des cours et TDs de java de ce semestre. Il nous a également permis d'approfondir nos connaissances en nous poussant à chercher par nous-même via les différentes ressources sur internet et la documentation java.