

Run Away



FABRE Maxime
LEPOT Florian

2015

Sommaire

Introduction.....	2
I. Analyse fonctionnelle générale	3
A. But du Jeu	3
B. Objectif	3
C. Moyen mis en œuvre	3
II. Description de l'application.....	5
A. Diagramme UML des cas d'utilisation	5
B. Diagramme UML de classes.....	5
III. Réalisation de l'application	6
A. Liste des fonctions développées	6
1. Interface graphique	6
2. Lancer le jeu	6
3. La carte	6
4. Les ennemis	7
5. Les bonus	7
6. Les PNJ	8
7. Les musiques et sons	8
8. Pause	8
9. Gagner et perdre	8
B. Tests automatiques effectués	9
C. Difficultés rencontrées	9
Conclusion	10
Annexe.....	0

Introduction

Pour ce premier projet de Java, nous n'avions pas de consignes précises, le projet était libre. Nous avons directement pensé à faire un jeu, considérant que c'est la façon la plus facile et la plus intéressante d'explorer les possibilités du Java. Nous nous sommes donc lancés dans un jeu simple avec une difficulté intéressante, en gardant un principe de jeu réellement simple.

Nous avons donc fait un jeu où l'on incarne un personnage, devant s'enfuir en esquivant des ennemis qui ne doivent pas nous voir. Pour pouvoir développer ce jeu, nous avons utilisé la librairie 2D Slick.

Ce projet nous a beaucoup appris sur la manière à réfléchir et visualiser la structure d'un programme avant de se lancer dans le code, ceci dans le but d'éviter d'obtenir un programme trop lourd, non optimisé, et donc, lent. L'enjeu était bien évidemment de se familiariser avec la POO et d'apprendre à structurer un programme de la sorte.

I. Analyse fonctionnelle générale

A. But du Jeu

Le but du jeu est simple : retrouver la sortie dans la carte. La carte possède 25 salles, chacune contient une sortie. Cependant, seulement une de ces sorties est la bonne, attention à ne pas se tromper.

Chaque salle possède un ennemi, s'il vous voit ça sera terminé pour vous, sans arme inutile d'aller au corps à corps, il va donc falloir être patient et stratégique. Plusieurs bonus sont disposés dans la carte, bien qu'ils soient appelés bonus, certains peuvent vous désavantager.

Il faudra encore une fois de plus, jouer stratégiquement et réfléchir si prendre le risque de tenter le bonus peut vous être utile.

B. Objectif

Etant donné que le sujet de ce projet en programmation JAVA était libre, nous avons donc décidé de concevoir un jeu. La réalisation d'un jeu était pour nous l'idée la plus motivante et la manière la plus concrète d'approfondir les connaissances du langage JAVA vu en cours.

Un projet sous forme de jeu était intéressant, mais le fond restait pour nous le plus important. Nous voulions concevoir un jeu original, agréable à jouer et d'une durée de vie la plus longue possible. Le jeu ne devait être ni trop simple ni trop dur, une partie ne devait pas durer éternellement, et chaque partie devait être différente de la précédente.

Lors de la lecture du sujet et du temps imparti qu'il nous était donné, nous ne savions pas si nous aurions le temps de finir ce que nous voulions faire. Notre objectif principal était de rendre un jeu fonctionnel comportant les éléments de bases essentiels, à savoir : des ennemis, une sortie, une carte, et des cachettes. Si le temps était favorable, la bande sonore du jeu et le système de bonus seraient générés ; ce qui a été le cas.

Nos objectifs ont donc été réalisés, que ce soit techniquement ou au niveau du Gameplay. En effet d'une part les bonus aléatoires présents dans le jeu enlèvent tout aspect répétitif du jeu, d'une autre la lenteur voulue du personnage et l'absence d'armes pour se défendre rajoute un effet de tension recherché dans l'idée du jeu.

C. Moyen mis en œuvre

Avant même de commencer le projet nous avons longtemps réfléchi, quant à la structure de notre code et de la manière dont nous allons travailler. Après plusieurs recherches nous avons découvert la librairie Slick2D. Cette librairie était conseillée par de nombreux forums pour la réalisation de jeu, ce qui tombait à merveille.

Jusque-là toutes les librairies que nous avons utilisées dans nos projets précédant (SDL, Qt), possédaient une forte documentation francophone. En ce qui concerne Slick2d les tutoriaux francophone est quasi inexistante et les tutoriaux anglophone faible.

La documentation a donc été notre meilleur ami durant la conception du jeu, ce qui nous a permis de nous adapter plus facilement. La création de la carte a été créée grâce à Tiled, un éditeur de Map facilement intégrable dans Slick2D.

Grace à ce gestionnaire nous avons donc pu créer une grande Map et gérer les collisions plus facilement. Notre code est basé sur la version Java 6 comme demandé dans le sujet.

II. Description de l'application

A. Diagramme UML des cas d'utilisation

Voir Annexe.

B. Diagramme UML de classes

Voir Annexe.

III. Réalisation de l'application

A. Liste des fonctions développées

Comme on l'a expliqué précédemment, grâce à Slick, nous avons pu établir différents Etats, notamment le Menu de départ, le Jeu, l'Etat « Gagné » et pour finir le « Perdu ».

En lançant le programme, on arrive donc au menu de départ. Sur ce menu, on peut lancer le jeu ou quitter.

1. Interface graphique

Un élément sans lequel notre jeu n'aurait aucun intérêt, est l'interface graphique.

Grâce à Slick 2D, qui est une librairie graphique pour Java, nous avons pu mettre en place une interface avec différentes images et un certain dynamisme dans les déplacements de nos personnages.

2. Lancer le jeu

Dès lors que le joueur lance le jeu, le programme va générer tout ce dont on a besoin, à savoir :

- Un héros qui sera contrôlé par le joueur,
- La carte globale, ainsi que les « Tiles » bloquants,
- Une sortie à un certain emplacement,
- Toutes les musiques & sons,
- Deux PNJ qui vont offrir un indice sur l'emplacement de la sortie,
- Des ennemis : 1 par salle,
- Des bonus : 25 répartis aléatoirement sur toute la carte.

Le joueur peut alors commencer à se déplacer sur la carte avec les flèches du clavier.

3. La carte

La carte du jeu est fixe et est composée de 25 salles. Ces salles ont une ambiance différente pour que le joueur ne soit pas totalement perdu et puisse avoir quelques repères.

Ces salles sont composées de carrés, appelés « Tiles », pouvant être bloqués ou non. Si un Tile est bloqué, ni le joueur ni l'ennemi ne pourront le traverser. Par exemple les arbres, les rochers ou les barrières.

Le joueur peut changer de salle en suivant simplement le chemin dans le jeu.

Sur cette carte se trouve 24 sorties, dont seulement 1 permet de gagner. Les 23 autres sont piégées et vont mettre le joueur en difficultés.

4. Les ennemis

Dans les salles, il y a également au moins 1 ennemi qui peut repérer le joueur. Le champ de vision de cet ennemi est représenté par un rectangle blanc qui s'adapte automatiquement à la direction de l'ennemi ainsi qu'à sa position.

Si le joueur est derrière un bloc non traversable, il n'est pas repéré car protégé par ce même bloc.

De base, l'ennemi est plus rapide que le joueur, mais cela peut évoluer si ce dernier prend des bonus (ou malus) situées dans les salles également, sous forme de chopes de bière.

Les déplacements des ennemis sont gérés aléatoirement, ils ne disposent pas d'une IA avancée. Nous avons pris la décision de ne pas développer d'IA pour l'ennemi car la difficulté est suffisamment élevée sans cela.

5. Les bonus

Des bonus sont également disposés aléatoirement partout sur la carte. Ces bonus peuvent apporter différents avantages, ou désavantages, ceci étant défini lors de la création de la carte.

Les différents bonus sont les suivants :

- Augmentation de la vitesse du joueur
- Diminution de la vitesse du joueur
- Invisibilité (Le joueur ne peut plus être repéré)
- Freeze joueur : Le joueur ne peut plus bouger
- Freeze ennemi : Les ennemis ne peuvent plus bouger
- Grand ralentissement joueur : Le joueur a une vitesse très petite
- Téléportation aléatoire du joueur
- Augmentation de la vitesse de l'ennemi
- Master Power : Invisibilité + Vitesse largement augmentée
- Ecran noir : Le joueur ne voit plus rien durant un certain temps
- Emplacement sortie : Affichage de l'emplacement de la sortie

Pour prendre un bonus, il suffit de passer sur une chope de bière. Un son différent pour chaque bonus se lancera, tous n'ayant pas un effet directement visible. Il n'y a aucun moyen pour le joueur de savoir quel bonus il obtiendra avant de le prendre.

6. Les PNJ

Comme dit précédemment, il y a deux PNJ situés sur la carte. Leur placement est aléatoire, ils peuvent donc se trouver n'importe où (y compris dans la salle de départ).

Ces PNJ sont très utiles car en leur parlant (Touche ESPACE à proximité de l'un d'eux), vous obtenez un indice sur la position de la sortie.

L'un des 2 vous donnera le nombre de salles à parcourir horizontalement pour accéder à la sortie, l'autre verticalement.

7. Les musiques et sons

Nous avons placé des musiques et des sons dans notre jeu. Il y a en effet des musiques d'ambiance différentes en fonction de l'Etat dans lequel on se trouve (Menu / Jeu / Gagné / Perdu).

Par ailleurs, comme dit précédemment, nous avons mis en place des sons s'activant lorsque le joueur prend des bonus, ceux-ci sont différents en fonction du bonus pour qu'il sache lequel s'est activé, tous n'étant pas forcément visibles au premier coup d'œil.

8. Pause

Nous avons mis en place une touche permettant de mettre le jeu en pause. Tout déplacement est alors impossible, pour le joueur comme pour les ennemis.

Dans ce menu, le joueur peut choisir de reprendre le jeu, retourner au menu principal ou quitter.

9. Gagner et perdre

Pour gagner, il faut que le joueur trouve la bonne sortie, et y entre. S'il se trompe de sortie, entre 1 et 5 ennemis apparaissent dans la salle, il ne faut donc pas prendre la mauvaise sortie, sous peine de grandes difficultés. Si le joueur réussit à entrer dans la sortie, il gagne et un nouvel écran apparaît lui signifiant qu'il a gagné, avec une nouvelle musique bonus.

Perdre est beaucoup plus facile, il suffit de se faire repérer par l'ennemi, donc d'être dans son champ de vision et non caché par un bloc. Dès que le joueur s'est fait repérer, un nouvel écran apparaît lui signifiant qu'il a perdu, avec une nouvelle musique.

B. Tests automatiques effectués

Nous effectuons de nombreux tests tout au long de notre programme, notamment :

- Sortie de la carte : Nous vérifions à chaque déplacement du joueur ou des ennemis que ceux-ci ne se dirigent pas en dehors de la carte.
- Blocage des Tiles non traversables : De même que pour la sortie de la carte, nous vérifions que lors d'un déplacement, le joueur et les ennemis ne puissent pas traverser un tile bloqué, ni le chevaucher.
- Gestion de la vitesse : Nous vérifions également qu'avec les bonus, le joueur ou les ennemis ne peuvent pas avoir une vitesse nulle ou inférieure à 0, le jeu serait bloqué dans le cas contraire, nous l'avons donc bloqué.
- Affichage des indices des PNJ : Avant d'afficher l'indice d'un PNJ, nous vérifions sa position. S'il est trop près d'un bord de l'écran et que l'information va être coupée, nous la décalons et l'affichons plus loin pour la voir entièrement.

Grâce à ces tests automatiques, on ne peut pas avoir d'overflow ou autres bugs liés.

C. Difficultés rencontrées

Lors de la conception de notre projet Run Away, nous avons rencontrées plusieurs difficultés. Par exemple lors de la conversion entre pixels, tiles, salles, etc... En effet, pour Slick les coordonnées (0 ; 0) sont situées en bas à gauche, alors que pour tout le reste, elles se situent en haut à gauche.

D'autre part, lors de la conversion pixels vers Tiles, on doit arrondir le résultat, cela créait dans certains cas des erreurs ou des décalages, notamment pour vérifier si le Tile est bloquant ou non.

Nous avons cependant réussi à résoudre tous les problèmes que nous avons rencontrés en différenciant les différentes façons d'arrondir.

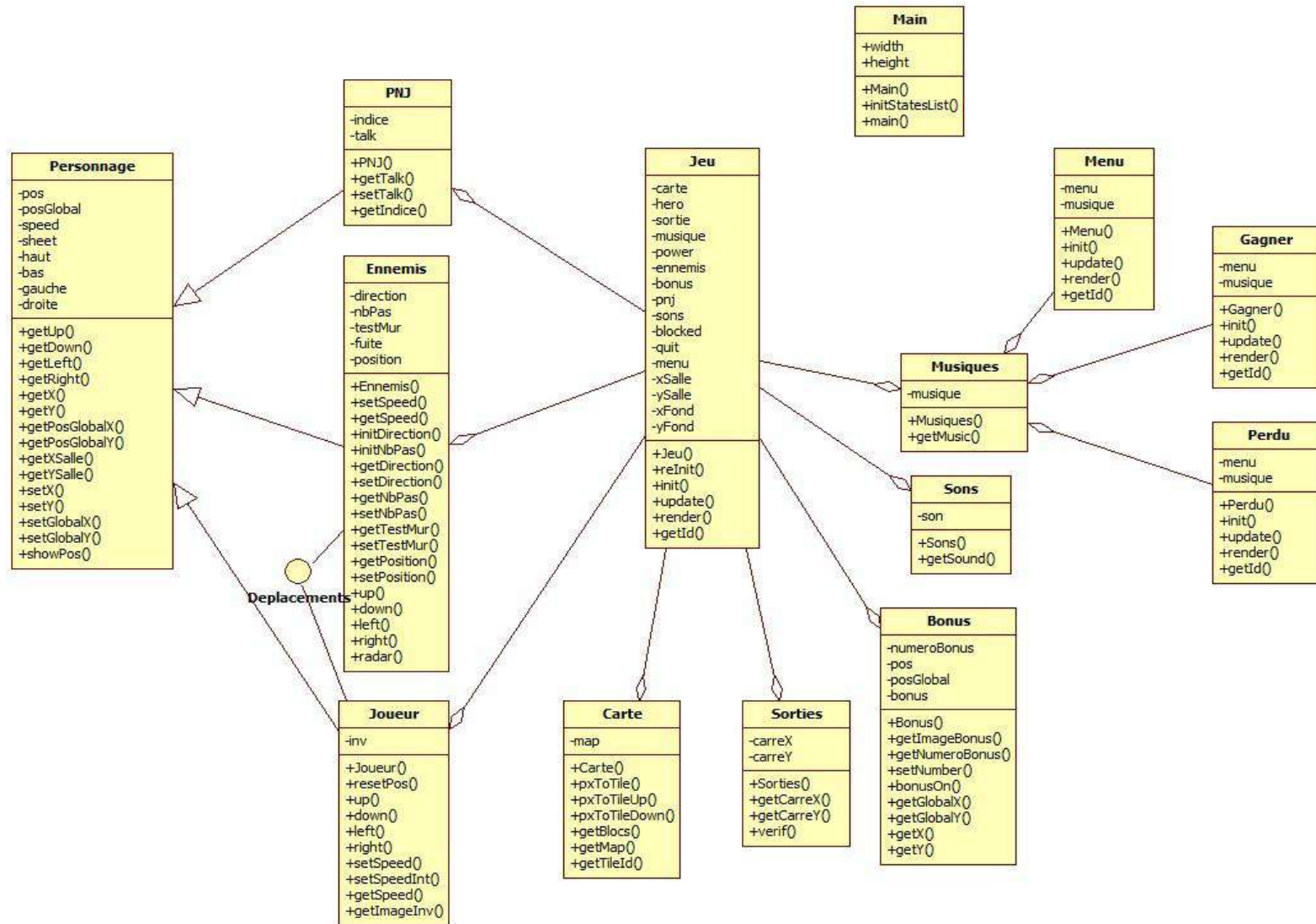
Conclusion

Pour finir sur ce projet, nous avons essayé de respecter au maximum le cahier des charges. En plus d'avoir réalisé nos objectifs de départ nous avons implémenté les améliorations que nous voulions, pour un résultat qui nous satisfait.

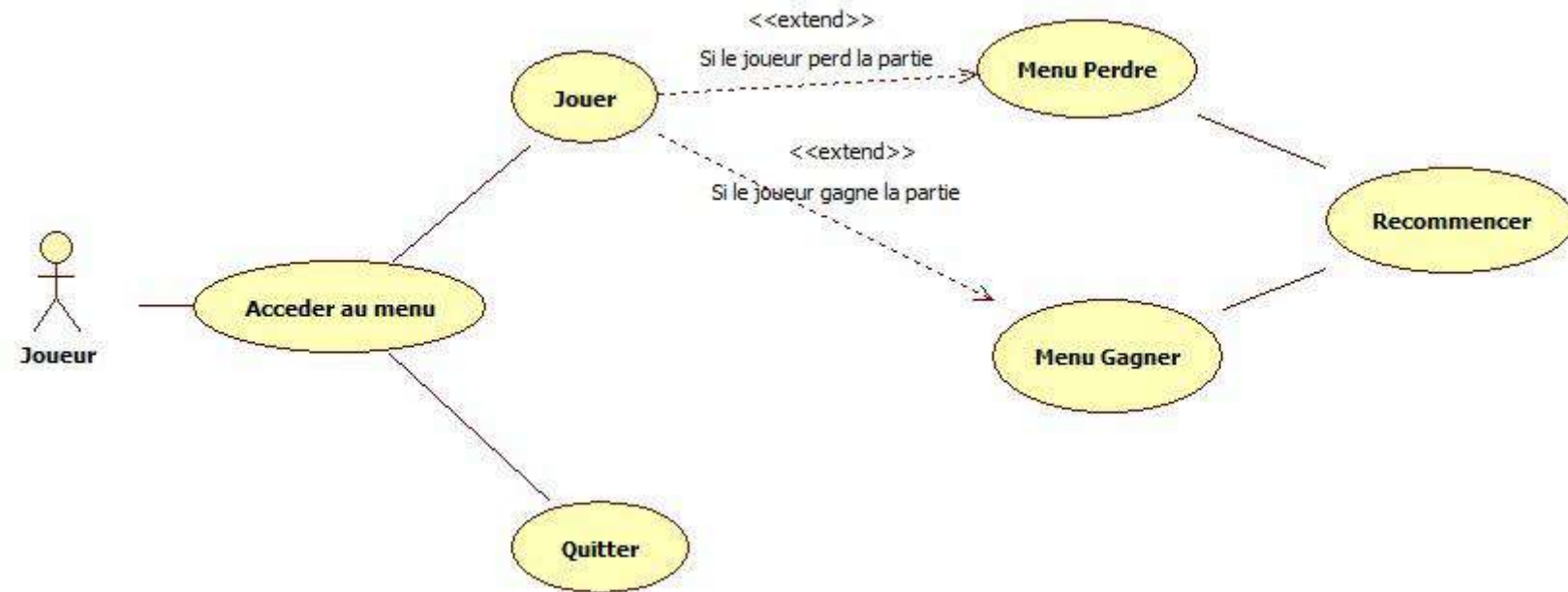
Nous avons aussi beaucoup appris quant à l'utilisation du langage JAVA et de la POO en général, et cela nous a montré que ce langage avait d'énormes possibilités.

Ce projet nous a également permis d'apprendre à utiliser Slick2D qui simplifie beaucoup d'éléments et qui nous a permis de concevoir un jeu plus performant.

Annexe



1. Diagramme des Classes



2. Diagramme des Cas d'utilisations