IFT1025 - Travail Pratique 1

The Legend of Zoe

Concepts appliqués : Programmation orientée objet, algorithmie, hiérarchies de classes, développement d'application de taille moyenne

---

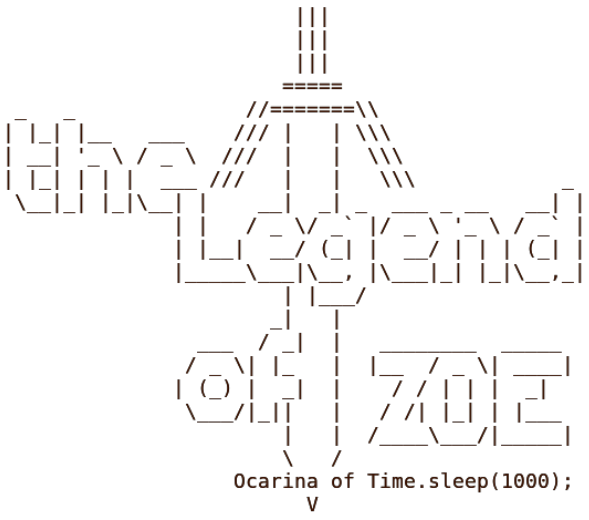


Figure 1: Écran d'intro du jeu

Contexte

Ce premier travail pratique consiste à programmer un petit jeu en ligne de commande inspiré de *The Legend of Zelda* et des jeux *Roguelike*.

Vous incarnez *Zoe*, la cousine de Zelda, qui doit partir à la recherche des pièces de l'*Hexaforce* pour rétablir l'ordre dans le monde.

Zoe avance à travers les 6 niveaux d'un donjon, chaque niveau cache une des pièces de l'Hexaforce.

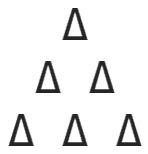


Figure 2: Les six pièces de l'Hexaforce

Description du jeu

Zoe se promène dans un donjon gardé par des monstres. Chaque niveau du donjon renferme quelques trésors, un certain nombre de monstres et une sortie.

Un niveau peut être affiché textuellement dans la console :

```
################   ####################
###############      ############    ###
######## ######      @########### $   ##
########   #####      #############  ###
########    #####     ############   ###
#########     #   $   ###  ######    ###
#########    ##      ###    ####    ####
#########    #      ###  &   ##    #####
######## @  ###   ####            #####
########   ####   ####    ##  E   ######
#######    ###     ##     ###    #######
####### x               ###  _  ######
########       #        #####    ######
###########  ##################  ######
```

Exemple de niveau

Chaque élément du jeu a une représentation en caractère textuel :

```
&: Zoe
#: mur du donjon
$: trésor fermé (cache un item)
_: trésor ouvert (l'item a déjà été pris)
@: monstre en vie
x: monstre battu par Zoe (0 vie)
E: sortie du niveau
```

Le jeu se joue *tour par tour* : *Zoe* effectue une action, puis les monstres font leur action, puis *Zoe* fait une action, etc. *Zoe* joue toujours en premier lorsqu'un nouveau niveau commence.

Zoe peut se déplacer dans 4 directions (haut/bas/gauche/droite), ouvrir un trésor à proximité et prendre l'item contenu, attaquer un monstre à proximité et creuser des trous dans les murs du donjon autour d'elle.

Les monstres peuvent se déplacer dans 8 directions (haut/bas/gauche/droite/diagonales) et attaquer *Zoe* lorsqu'elle est à proximité. Lorsqu'un monstre meurt, il libère un item prédéterminé.

Un trésor peut seulement être ouvert lorsqu'on est à proximité

Zoe doit être sur l'une des 8 cases voisines pour pouvoir ouvrir un trésor

Lorsque Zoe ouvre un trésor, elle prend l'item à l'intérieur et elle l'utilise

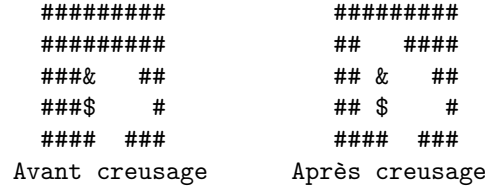
---

Un monstre peut seulement être attaqué lorsqu'on est à proximité

Un monstre attaque Zoe dès qu'elle se trouve dans son voisinage

Zoe peut attaquer les monstres dans son voisinage. Lorsqu'un monstre n'a plus de vie, il meurt et il libère un item.

Zoe peut creuser dans les murs du donjon autour d'elle



Avant creusage

Après creusage

Notez qu'on peut seulement creuser dans les murs, on ne peut pas creuser dans les trésors, monstres et autres.

Chaque niveau contient un certain nombre de trésors et de monstres. Les items contenus dans les trésors ou libérés par le monstres sont les suivants :

- Un coeur : redonne un point de vie à *Zoe*
- Une potion de vie : redonne toute la vie à *Zoe*
- Un morceau d'*Hexaforce* : il y a exactement un morceau d'*Hexaforce* par niveau, que *Zoe* doit ramasser avant de passer au niveau suivant

Il y a un nombre aléatoire de coeurs et de potions de vie dans chaque niveau.

Les monstres ont une force et un nombre de points de vie variable :

- Le nombre de points de vie initial d'un monstre au niveau N est égal à `(int)` $max(0.6N, 1)$
- La force des monstres (le nombre de points de vie que *Zoe* perd lorsqu'elle se fait attaquer) au niveau N est de `(int)` $max(0.4N, 1)$
- *Zoe* enlève 1 point de vie aux monstres par attaque

Notez que le nombre de points de vies et la force sont des nombres entiers. Notez aussi que les niveaux sont numérotés de 1 à 6.

Chaque déplacement de *Zoe* compte comme un tour. *Zoe* peut se déplacer sur toutes les cases du niveau qui **ne contiennent pas** un mur, un trésor ou la sortie.

Notez qu'il est **possible** que *Zoe* se retrouve sur la même case qu'un monstre (vivant ou mort). Deux ou plusieurs monstres peuvent se retrouver sur la même case.

Lorsque c'est le tour des monstres, chaque monstre joue. Le comportement des monstres est le suivant :

- Si *Zoe* est dans une des 9 cases avoisinantes (8 cases voisines + la case sur laquelle le monstre se trouve), attaquer *Zoe*
- Si *Zoe* est plus loin :
  - Regarder la coordonnée x du monstre et calculer le déplacement horizontal dx à faire pour se rapprocher de *Zoe*
    - Si *Zoe* est à droite, $dx = +1$
    - Si *Zoe* est à gauche, $dx = -1$
  - Regarder la coordonnée y et calculer le déplacement vertical dy similairement
  - Si la case $(x + dx, y + dy)$ est disponible, aller dessus, sinon, ne rien faire
- Par exemple : si *Zoe* est en position $(4, 7)$ dans le donjon et que le monstre A est en position $(10, 4)$, le monstre se déplace sur la case $(9, 5)$ si elle est disponible et reste sur place sinon

Zoe commence la partie avec 5 vies et 0 morceau d'Hexaforce. Le jeu se termine soit :

- Quand *Zoe* n'a plus de vies (partie perdue)
- Dès que *Zoe* a collecté le $6^{ème}$ morceau de l'*Hexaforce* (partie gagnée)

Design Orienté Objet

Ce sera à vous de choisir le découpage en classes optimal pour votre programme. Faites bon usage de l'orienté objet et de l'héritage lorque nécessaire.

Tous les designs décrits dans le cadre de l'exercice #3 et les commentaires donnés sont disponibles à tout le monde, n'hésitez pas à les consulter pour vous inspirer, mais gardez en tête que tout partage direct de code entre équipes constitue un plagiat et résultera en la note de zéro.

Application en ligne de commande

Au moment de jouer, on entre une commande au clavier :

```
w: déplace Zoe une case vers le haut
a: déplace Zoe une case vers la gauche
s: déplace Zoe une case vers le bas
d: déplace Zoe une case vers la droite
c: creuser dans les murs avoisinants
   jusqu'à 8 murs voisins -- notez qu'on peut seulement creuser dans les murs,
   on ne peut pas creuser dans les trésors, monstres et autres
x: attaque tous les monstres dans le voisinage de Zoe
   voisinage de 8 voisins, incluant les diagonales + la case sur laquelle
   Zoe se trouve (si jamais un monstre est sur la même case que Zoe)
o: ouvre tous les trésors dans le voisinage de Zoe
   jusqu'à 8 voisins
q: ferme le jeu d'un coup sec avec System.exit(0);
```

Toute autre commande ou toute commande invalide (ex.: tenter de creuser lorsqu'il n'y a pas de murs autour, d'avancer dans un mur, entrer la commande `f` qui n'est pas définie, etc.) a pour effet de faire passer un tour à *Zoe*.

Notez qu'on ne peut pas sortir des bordures du jeu, c'est comme s'il y avait des murs tout autour de la carte visible.

Notez également qu'on doit pouvoir entrer plus d'une seule commande d'un coup, donc la commande :

`awww`

Fait avancer *Zoe* à gauche d'une case puis en haut de trois cases.

Chaque commande compte pour un tour différent, donc `awww` demande 4 tours de jeu à *Zoe* pour s'exécuter au complet. Notez que `c` (creuser dans le voisinage), `x` (attaquer le voisinage) et `o` (ouvrir les trésors du voisinage) comptent toutes pour un seul tour.

Code fourni

Les fichiers suivants vous sont fournis :

```
LegendOfZoe.java ------- Fichier principal pour le programme
LevelGenerator.java ---- Sert à générer un niveau du jeu
Messages.java ---------- Contient des messages à afficher à différents moments du jeu
Paire.java ------------- Utilitaire pour stocker une paire de deux objets
Automate2D.java -------- Utilitaire (nécessaire au générateur de niveaux)
```

`LegendOfZoe.java`

Cette classe doit être le point d'entrée de votre programme.

`LevelGenerator.java`

Cette classe permet de générer un niveau du jeu. Le premier niveau est le niveau 1, il y a six niveaux au total (un morceau d'*Hexaforce* par niveau).

La méthode `generateLevel()` permet d'obtenir des informations sur le niveau généré, soit un tableau 2D de `boolean`s indiquant où sont les murs (`carte[y][x] == true -> mur, false -> vide`) et une description textuelle des objets sur le niveau au format `String[]` :

```
[tresor:coeur:6:7, monstre:hexaforce:34:2, tresor:potionvie:19:7, sortie:28:9, zoe:25:4]
```

Les `tresor`s et les `monstre`s sont suivis du type d'item qu'ils contiennent (`coeur/potionvie/hexaforce`) puis de leur position sur la carte au format `x:y`. La `sortie` et `Zoe` sont uniquement suivies de leur position `x:y`.

`Messages.java`

Cette classe contient trois méthodes statiques qui affichent de jolis dessins textuels sur la console pour rendre le jeu plus dynamique et agréable :

```
afficherIntro() ------ À appeler avant de commencer le jeu
afficherVictoire() --- À appeler lorsque Zoe gagne
afficherDefaite() ---- À appeler lorsque Zoe meurt
```

Au tout début du programme, on devrait appeller `afficherIntro()`.

Lorsque *Zoe* gagne ou perd, `afficherVictoire()` ou `afficherDefaite()` est appelée (selon le cas) puis le programme se termine.

`Automate2D.java`

Cette classe permet de générer un pattern de caverne. Jettez-y un coup d'oeil si vous êtes curieux, mais gardez en tête que vous n'aurez pas à l'utiliser ni à la modifier, la classe `LevelGenerator` se charge déjà de l'utiliser.

Bonus (5%)

L'"intelligence artificielle" des ennemis telle que décrite dans l'énoncé est très simpliste... Ajoutez un algorithme de *pathfinding* plus sophistiqué pour choisir les mouvements des ennemis, de façon à ce qu'ils ne se bloquent pas dans les murs.

De base, lorsqu'on lance le programme, l'algorithme décrit plus haut doit être exécuté lors du tour des ennemis. On devrait pouvoir activer l'algorithme sophistiqué manuellement avec l'argument en ligne de commande :

```
java LegendOfZoe --bonus
```

Si vous choisissez de faire le bonus, ajoutez un fichier `BONUS.txt` à votre rapport dans lequel vous devez décrire ce que vous avez fait.

Barème

- **35** ~ Découpage en classes et utilisation judicieuse de l'héritage
- **40%** ~ Exécution (tests du programme)
- **25%** ~ Qualité du code
  - Code bien commenté (*JavaDoc*), bien indenté, bon découpage en fonctions, encapsulation, noms de variables bien choisis, performance du code raisonnable, camelCase, pas trop compact, pas trop espacé... bref, du code clair et lisible en suivant les principes que vous connaissez
  - Limitez vos lignes de code à 120 caractères de large
- **5%** ~ Bonus

Indications supplémentaires

- La date de remise est le *18 mars 2019 à 23h55*. Il y a une pénalité de **25%** pour chaque jour de retard.
- Vous **devez** faire le travail par groupes de 2 personnes. Indiquez vos noms clairement dans les commentaires au début de votre code, dans le fichier principal (`LegendOfZoe.java`).
- Une seule personne par équipe doit remettre le travail sur StudiUM, l'autre doit remettre un fichier nommé `equipe.txt` contenant uniquement le code d'identification de l'autre personne (`p1234...`, soit ce que vous utilisez pour vous connecter sur StudiUM)
- Un travail fait seul engendrera une pénalité. Les équipes de plus de deux ne sont pas acceptées.
- Basez-vous sur le code fourni et remettez tous les fichiers dans une archive `.zip`
- De plus :
  - La performance de votre code doit être raisonnable
  - Chaque fonction devrait être documentée en suivant le standard `JavaDoc`
  - Il devrait y avoir des lignes blanches pour que le code ne soit pas trop dense (utilisez votre bon sens pour arriver à un code facile à lire)
  - Les identificateurs doivent être bien choisis pour être compréhensibles (évitez les noms à une lettre, à l'exception de i, j, ... pour les variables d'itérations des boucles for)
  - Vous devez respecter le standard de code pour ce projet, soit les noms de variables et de méthodes en camelCase