

Projet Logiciel Transversal

« Pixel Dungeon »

Yijie LI – Florian VANDEBROUCK



Illustration 1 : Exemple du jeu Pixel Dungeon

Table des matières

1 Objectif.....	3
1.1 Présentation générale.....	3
1.2 Règles du jeu.....	3
1.3 Conception Logiciel.....	3
2 Description et conception des états.....	4
2.1 Description des états.....	4
2.2 Conception logiciel.....	4
2.3 Conception logiciel : extension pour le rendu.....	4
2.4 Conception logiciel : extension pour le moteur de jeu.....	4
2.5 Ressources.....	4
3 Rendu : Stratégie et Conception.....	6
3.1 Stratégie de rendu d'un état.....	6
3.2 Conception logiciel.....	6
3.3 Conception logiciel : extension pour les animations.....	6
3.4 Ressources.....	6
3.5 Exemple de rendu.....	6
4 Règles de changement d'états et moteur de jeu.....	8
4.1 Horloge globale.....	8
4.2 Changements extérieurs.....	8
4.3 Changements autonomes.....	8
4.4 Conception logiciel.....	8
4.5 Conception logiciel : extension pour l'IA.....	8
4.6 Conception logiciel : extension pour la parallélisation.....	8
5 Intelligence Artificielle.....	10
5.1 Stratégies.....	10
5.1.1 Intelligence minimale.....	10
5.1.2 Intelligence basée sur des heuristiques.....	10
5.1.3 Intelligence basée sur les arbres de recherche.....	10
5.2 Conception logiciel.....	10
5.3 Conception logiciel : extension pour l'IA composée.....	10
5.4 Conception logiciel : extension pour IA avancée.....	10
5.5 Conception logiciel : extension pour la parallélisation.....	10
6 Modularisation.....	11
6.1 Organisation des modules.....	11
6.1.1 Répartition sur différents threads.....	11
6.1.2 Répartition sur différentes machines.....	11
6.2 Conception logiciel.....	11
6.3 Conception logiciel : extension réseau.....	11
6.4 Conception logiciel : client Android.....	11

1 Objectif

1.1 Présentation générale

L'objectif de ce projet est la réalisation du jeu reprenant le principe de Pixel Dungeon. Pixel Dungeon est un jeu mobile Android de type « roguelike ». Le joueur doit avancer le plus loin possible dans un environnement généré aléatoirement. Sur l'illustration 1, on peut voir différents états du jeu : le menu à gauche, le choix du personnage au milieu et le début d'une partie à droite.

1.2 Règles du jeu

Univers du jeu. Le joueur incarne un personnage et explore tour par tour les étages d'un donjon générés aléatoirement. Le joueur passe d'étage en étage par des escaliers. On peut les descendre mais aussi les remonter. Les étages sont constitués de salles comportant des monstres, des coffres, des objets, des pièges... Il existe des salles particulières qui peuvent être cachées ou non. Il y a des boutiques mises à certains endroits du jeu. Le but principal du jeu est d'avancer jusqu'au dernier étage contenant un trésor : une amulette.

Joueur. Le personnage jouable est défini par une classe avec des statistiques et des objets de départ qui lui sont propres. Il pourra aussi gagner de l'expérience pour monter de niveau et ainsi améliorer ses caractéristiques. On pourra récupérer et utiliser différents objets pour nous aider à avancer. Chaque objet a un rôle particulier et possède certaines caractéristiques. Le joueur n'a aucune connaissance au départ sur l'effet ou sur certaines statistiques des objets. C'est au moyen de son utilisation ou d'objets dédiés à la reconnaissance qu'il pourra savoir. Il existe aussi un système de statuts qui peuvent être donnés par des objets, par des pièges, par des monstres... Il y a aussi la faim : il existe de la nourriture et chaque pas du personnage entame sa faim. A partir d'un certain moment, le personnage peut mourir de faim. Le joueur possède aussi une « vision ». Il ne peut pas voir au-delà d'un certain nombre de cases. Les cases non visitées sont noires. Les cases visitées hors vision, le joueur peut savoir le type de case mais pas ce qu'elles contiennent.

Monstres. Il faudra affronter différents monstres et il existe des boss à intervalle d'étages régulier. Dans un étage, le joueur n'est pas obligé de tuer tous les monstres de l'étage pour passer à l'étage suivant, alors que pour le boss il faut le tuer afin de récupérer une clé pour passer à l'étage suivant. Tuer un monstre donne de l'expérience et peut donner des objets et de l'or, monnaie du jeu.

Fin du jeu. La mort est punitive, c'est-à-dire qu'il n'existe pas de système de sauvegarde du jeu. La mort entraîne l'arrêt du jeu et il faudra recommencer la partie au tout début avec un nouveau personnage.

1.3 Conception Logiciel

Présenter ici les packages de votre solution, ainsi que leurs dépendances.

2 Description et conception des états

L'objectif de cette section est une description très fine des états dans le projet. Plusieurs niveaux de descriptions sont attendus. Le premier doit être général, afin que le lecteur puisse comprendre les éléments et principes en jeux. Le niveau suivant est celui de la conception logiciel. Pour ce faire, on présente à la fois un diagramme des classes, ainsi qu'un commentaire détaillé de ce diagramme. Indiquer l'utilisation de patron de conception sera très apprécié. Notez bien que les règles de changement d'état ne sont pas attendues dans cette section, même s'il n'est pas interdit d'illustrer de temps à autre des états par leur possibles changements.

2.1 Description des états

Un état du jeu est formée par un étage (ensemble d'éléments décor), des objets et de personnages. Tous les éléments possèdent les propriétés suivantes :

- Coordonnées en X
- Coordonnées en Y

2.1.1 État élément type Décor

Chaque étage est composé d'une grille d'éléments nommé « Cases ». La taille de cette grille est défini au début du jeu . Il existe deux types de Cases :

- **Case « Wall »** : ces cases ne peuvent pas être traversé par les autres éléments du jeu. C'est une case qui définit les limites d'une salle.
- **Case « Space »** : les cases « Espace » sont franchissables par les autres éléments. Il en existe deux types en fonction de son interaction avec le joueur.
 - **Case « Ground »** : Il en existe qu'un seul type de case : **Case « Sol »**.
 - **Case « Interactive »** : Il en existe plusieurs types :
 - **Case « Stair »** : Quand le héros arrive sur cette case il change d'étage. Il en existe deux par étage un montant et un descendant. L'escalier montant du premier niveau est le point de départ du joueur.
 - **Case « Door »** : Elle sépare deux salles entre elles. Elles peuvent être ouvertes et fermées
 - **Case « Trap »** : Quand le héros marche dessus, cela déclenche un effet sur lui.

2.1.2 État élément type Perso

Ce sont les éléments qui peuvent être contrôlés par le joueur ou par l'IA. Il existe deux types de personnages :

- **Perso « Héros »** : Cet élément est dirigé par le joueur, qui commande la propriété de direction et mouvement .La Chasserresse n'est disponible qu'après avoir battu les 3 premiers boss du jeu. Chaque héros possède ses propres avantages et défauts. Il peut avoir deux statuts :
 - **Statut normal** : Le héros peut se déplacer, détecter les objets et tuer les monstres.
 - **Statut mort** : Le héros a été tué par les monstres. C'est la fin du jeu.
- **Perso « Monstre »** : Ils sont également commandé par la propriété de direction,qu'elle proviennent d'un humain ou d'une IA :
 - **Statut non trouvé** : cas où le monstre n'est pas trouvé par joueur, le monstre ne bouge

pas.

- **Statut trouvé** : cas où le monstre est détecté par joueur, il s'élance vers le héros et l'attaque.
- **Statut mort** : cas où le monstre a été tué par le joueur.

2.1.3 État élément type Objets

Ce sont les éléments qui se trouvent dans le donjon, ou sur les monstres et qui peuvent récupérer pour ensuite être utilisé par le héros. On en retrouve deux catégories :

- **Objet « Consommable »** : Ce sont les objets qui peuvent être consommé par le héros. Dedans on retrouve la nourriture, les potions et les parchemins. Ces deux derniers ont des effets assez particuliers.
- **Objet « No Consommable »** : Ce sont les autres types d'objets, on en repère deux types :
 - **Objet « Equipment »** : Ce sont les objets que peut équiper le héros. On retrouve les armes, les armures, les bagues et les baguettes.
 - **Objet « KeyGold »** : Dans cette classe on retrouve les clés et l'or.

2.2 Conception logiciel

Le diagramme pour les états est présenté en Illustration 3, nous pouvons mettre en évidence les groupes suivants :

Classes Element. Toute la hiérarchie des classes filles d'Element permettent de représenter les différentes catégories et types d'élément à l'aide du Polymorphisme. Pour indiquer la classe de l'élément, on utilisera des méthodes pour savoir dans quelle branche fille il se trouve.

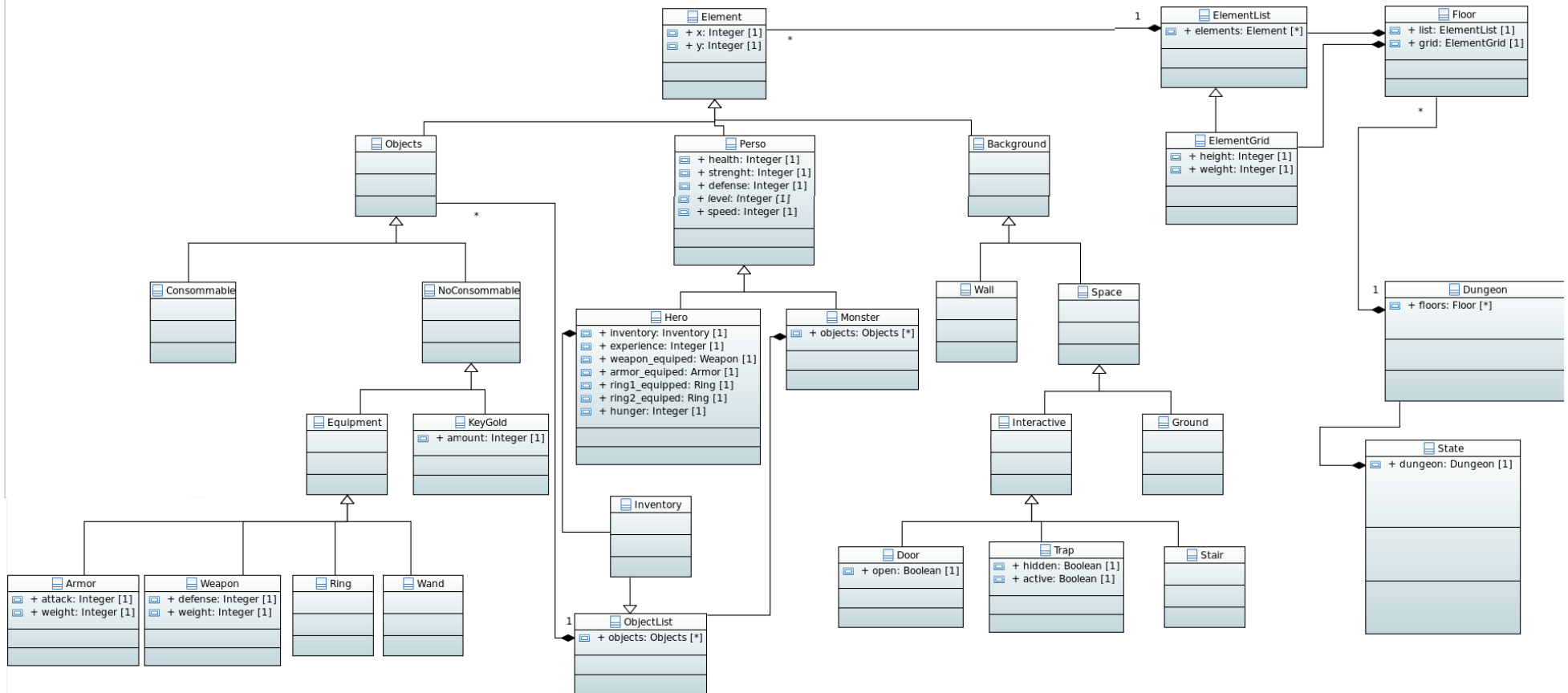
Conteneurs d'élément. Viennent ensuite les classes State, Dungeon, Floor, ElementList et ElementGrid qui permettent de contenir des ensembles d'éléments. ElementList contient une liste d'éléments, et ElementGrid étend ce conteneur pour lui ajouter des fonctions permettant de gérer une grille. Ces deux derniers éléments se retrouve dans la classe Floor, qui contient une grille pour placer les éléments et la liste de ces éléments (Perso, Background, Objects). Dungeon qui est l'ensemble des niveau, c'est-à-dire qu'il contient une liste de Floor. Enfin, la classe State est le conteneur principal, avec un Dungeon pour les niveaux.

2.3 Ressources

Pour les ressources, on utilisera un fichier texte pour générer les niveaux qui sera ensuite transformé en image à l'aide de matrice de textures.

Illustration 2 : Exemple de textures qui seront utilisées dans le jeu

Illustration 3: Diagramme des classes d'état



3 Rendu : Stratégie et Conception

Présentez ici la stratégie générale que vous comptez suivre pour rendre un état. Cela doit tenir compte des problématiques de synchronisation entre les changements d'états et la vitesse d'affichage à l'écran. Puis, lorsque vous serez rendu à la partie client/serveur, expliquez comment vous aller gérer les problèmes liés à la latence. Après cette description, présentez la conception logicielle. Pour celle-ci, il est fortement recommandé de former une première partie indépendante de toute librairie graphique, puis de présenter d'autres parties qui l'implémente pour une librairie particulière. Enfin, toutes les classes de la première partie doivent avoir pour unique dépendance les classes d'état de la section précédente.

3.1 Stratégie de rendu d'un état

3.2 Conception logiciel

3.3 Conception logiciel : extension pour les animations

3.4 Ressources

3.5 Exemple de rendu

Illustration 1: Diagramme de classes pour le rendu

4 Règles de changement d'états et moteur de jeu

Dans cette section, il faut présenter les événements qui peuvent faire passer d'un état à un autre. Il faut également décrire les aspects liés au temps, comme la chronologie des événements et les aspects de synchronisation. Une fois ceci présenté, on propose une conception logiciel pour pouvoir mettre en œuvre ces règles, autrement dit le moteur de jeu.

4.1 Horloge globale

4.2 Changements extérieurs

4.3 Changements autonomes

4.4 Conception logiciel

4.5 Conception logiciel : extension pour l'IA

4.6 Conception logiciel : extension pour la parallélisation

Illustration 2: Diagrammes des classes pour le moteur de jeu

5 Intelligence Artificielle

Cette section est dédiée aux stratégies et outils développés pour créer un joueur artificiel. Ce robot doit utiliser les mêmes commandes qu'un joueur humain, ie utiliser les mêmes actions/ordres que ceux produit par le clavier ou la souris. Le robot ne doit pas avoir accès à plus information qu'un joueur humain. Comme pour les autres sections, commencez par présenter la stratégie, puis la conception logicielle.

5.1 Stratégies

5.1.1 Intelligence minimale

5.1.2 Intelligence basée sur des heuristiques

5.1.3 Intelligence basée sur les arbres de recherche

5.2 Conception logiciel

5.3 Conception logiciel : extension pour l'IA composée

5.4 Conception logiciel : extension pour IA avancée

5.5 Conception logiciel : extension pour la parallélisation

6 Modularisation

Cette section se concentre sur la répartition des différents modules du jeu dans différents processus. Deux niveaux doivent être considérés. Le premier est la répartition des modules sur différents threads. Notons bien que ce qui est attendu est une parallélisation maximale des traitements: il faut bien démontrer que l'intersection des processus communs ou bloquant est minimale. Le deuxième niveau est la répartition des modules sur différentes machines, via une interface réseau. Dans tous les cas, motivez vos choix, et indiquez également les latences qui en résulte.

6.1 Organisation des modules

6.1.1 Répartition sur différents threads

6.1.2 Répartition sur différentes machines

6.2 Conception logiciel

6.3 Conception logiciel : extension réseau

6.4 Conception logiciel : client Android

Illustration 3: Diagramme de classes pour la modularisation

