Ramtani Walid
Zeineddine Nour

Architecture du Projet

Dans ce projet, le code a été organisé en plusieurs packages afin de rendre la gestion des différentes fonctionnalités plus claire et efficace. Avec le grand nombre de classes et la variété des éléments du jeu, cette structure permet de garder une architecture propre, modulaire et facile à faire évoluer au fur et à mesure du développement. Voici un aperçu des principaux packages et de ce qu'ils gèrent :

Package : effects

Le package **effects** gère les différents effets que les objets et ennemis peuvent avoir sur le héros et les ennemis pendant les combats. Chaque effet représente une action spécifique, comme infliger des dégâts, soigner ou appliquer une malédiction. Voici une description de chaque classe dans ce package :

AddDamageAdjWeapon

Cette classe applique un effet d'augmentation de dégâts à une arme adjacente dans le sac à dos du héros. Si un objet adjacent est une arme, elle verra ses dégâts augmenter.

- Méthode clé : execute() – Ajoute des dégâts à l'arme adjacente dans le sac à dos du héros.

AddDamageWeapon

Cette classe applique un bonus de dégâts à une arme pendant une durée limitée. Une fois l'effet appliqué, la durée de l'effet diminue jusqu'à ce qu'il expire.

AddEnergyCost

Cette classe ajoute ou retire des points d'énergie au héros. Elle est utilisée pour ajuster l'énergie du héros en fonction de différents événements.

AddHpHero

Cette classe augmente ou réduit les points de vie du héros. Elle permet de restaurer ou de perdre de la santé pendant les combats.

AddMana

Cette classe ajoute des points de mana au héros. Elle est utilisée pour gérer les ressources magiques nécessaires pour certaines capacités du héros.

AddProtectionAdj

Cette classe ajoute une protection (défense) à une armure adjacente dans le sac à dos du héros.

AttackAllEnemies

Cette classe inflige des dégâts à tous les ennemis présents dans le jeu. Elle est utilisée pour appliquer des effets de zone pendant les combats.

Curse

Cette classe applique une malédiction au héros. La malédiction peut avoir des effets négatifs comme réduire la santé du héros.

HeallAllEnemies

Cette classe guérit tous les ennemis, augmentant leurs points de vie. Elle peut être utilisée pour des ennemis qui s'entraident pendant le combat.

NumberUses

Cette classe gère un compteur du nombre d'utilisations d'un objet. À chaque utilisation, ce compteur est décrémenté jusqu'à ce qu'il atteigne zéro.

PoisonEffect

Inflige des dégâts à tous les ennemis du jeu, représentant un effet de poison sur plusieurs tours.

PoisonHero

Inflige des dégâts au héros, représentant un effet de poison appliqué à celui-ci.

ReduceDamageEffectEnemy

Cette classe réduit les dégâts infligés par un ennemi en diminuant son attaque. Elle est utilisée pour affaiblir les ennemis.

RemovePoisonFromSelf

Cette classe supprime l'effet de poison appliqué au héros, le protégeant ainsi d'un effet négatif continu.

Package : enemies

Le package enemies gère les différentes classes d'ennemis rencontrées par le héros dans le donjon. Chaque ennemi a des statistiques telles que la santé, l'attaque et la défense, ainsi que des effets ou actions spécifiques qu'il peut effectuer pendant les combats. Voici une description des différentes classes dans ce package :

Enemy (interface)

L'interface **Enemy** définit les comportements et les attributs communs à tous les ennemis. Elle inclut des méthodes pour gérer les points de vie, l'attaque, la défense et les actions spécifiques de l'ennemi.

- Méthodes clés : execute(), getAttacked(), defend(), apply(), isDead().

FrogWizard

Le FrogWizard est un ennemi qui peut empoisonner le héros et guérir les alliés. Il dispose de plusieurs effets et actions, ce qui le rend polyvalent sur le champ de bataille.

LivingShadow

Le LivingShadow est un ennemi sombre qui peut appliquer la malédiction au héros. Ses attaques sont plus orientées vers des effets négatifs comme le curse.

QueenBee

La QueenBee est un ennemi qui guérit ses alliés tout en empoisonnant le héros. Elle dispose de pouvoirs de guérison et de poison qui affectent les ennemis et le héros.

Ratwolf

Le **Ratwolf** est un ennemi classique qui attaque directement le héros. Il est robuste et peut infliger de lourds dégâts.

SmallRatwolf

Le **SmallRatwolf** est une version plus petite et moins puissante du **Ratwolf**, avec des statistiques réduites.

Package : Items

Le package **Items** gère tous les objets du jeu que le héros peut acquérir, utiliser et interagir avec. Ces objets ont des effets variés, allant des armes et armures aux potions, en passant par l'or, les clés et les objets magiques. Ils sont tous définis par l'interface **Item**, qui permet de standardiser et de gérer les objets de manière modulaire.

Item (interface)

L'interface **Item** définit les caractéristiques et comportements des objets du jeu. Chaque objet implémente cette interface et fournit les méthodes permettant de gérer ses effets, son utilisation, ses dimensions dans la grille du sac à dos, ainsi que son coût en énergie, son nom, sa rareté, et ses effets.

Classes Implémentant l'interface Item

Armor

La classe **Armor** représente un objet de protection pour le héros. Elle augmente la défense du héros et consomme de l'énergie lorsqu'elle est utilisée, mais ne disparaît pas après usage. Certaines armures peuvent aussi avoir des effets supplémentaires, comme renforcer d'autres objets adjacents.

Curse

La classe **Curse** représente une malédiction qui peut être appliquée au héros. Elle inflige des dégâts et peut être retirée de l'inventaire après usage.

Food

La classe **Food** représente un objet alimentaire que le héros peut consommer pour appliquer des effets comme la guérison ou l'augmentation de l'énergie. Certains aliments peuvent disparaître après usage.

Gold

La classe **Gold** représente des pièces d'or que le héros peut récupérer dans le jeu. Elles sont utilisées pour acheter des objets ou interagir avec des mécanismes.

Key

La classe **Key** représente une clé utilisée pour déverrouiller des portes ou des mécanismes dans le jeu. Elle n'a pas d'effet direct sur les personnages mais est nécessaire pour certaines interactions.

MagicItem

La classe **MagicItem** représente un objet magique utilisé par le héros. Il consomme du mana et peut infliger des dégâts à un ennemi ou appliquer des effets spéciaux, comme une attaque de zone.

ManaStone

La classe **ManaStone** représente un objet magique qui augmente les points de mana du héros. Il peut être utilisé pour restaurer la mana et peut être détruit après utilisation.

MeleeWeapon

La classe **MeleeWeapon** représente une arme de corps à corps, comme une épée ou une hache. Elle inflige des dégâts à un ennemi et consomme de l'énergie à chaque utilisation.

Potion

La classe **Potion** représente un objet de soin ou magique. Elle peut restaurer la santé, l'énergie ou appliquer des effets à l'ennemi ou au héros.

RangedWeapon

La classe **RangedWeapon** représente une arme à distance, comme un arc ou une arbalète. Elle inflige des dégâts à l'ennemi à distance et consomme de l'énergie.

Shield

La classe **Shield** représente un bouclier qui protège le héros contre les attaques ennemies. Il peut consommer de l'énergie pour fournir une protection supplémentaire.

- Méthodes clés :
    - onUse() : Applique la protection et réduit l'énergie du héros.
    - createNewInstance() : Crée une nouvelle instance du bouclier.
    - baseShape() : Définit la valeur de protection de base du bouclier.

Weapon

L'interface **Weapon** est implémentée par toutes les armes du jeu, qu'elles soient de corps à corps ou à distance. Elle définit les méthodes pour gérer les dégâts et les effets d'attaque.

- Méthodes clés :
    - damage() : Retourne la valeur des dégâts infligés par l'arme.
    - setAttack() : Permet de modifier les dégâts d'une arme.

Package : model

Le package **model** contient le cœur logique du jeu. Il regroupe toutes les classes qui représentent l'état du jeu : le héros, son inventaire, le donjon, les étages, ainsi que les entités avec lesquelles le joueur interagit (marchand, soigneur). Ce package ne gère pas l'affichage, mais uniquement les règles, les données et leur évolution pendant la partie.

BackPack

La classe **BackPack** gère l'inventaire du héros sous forme d'une grille. Elle permet de placer, retirer et organiser les objets tout en respectant les contraintes de taille et de position.

- L'inventaire est représenté par :

  - une grille `Item[][]` de taille 5×7,
  - une grille booléenne `openGrid` indiquant quelles cases sont accessibles,
  - une map associant chaque item aux cases qu'il occupe.

- Certaines cases sont verrouillées par défaut, ce qui limite l'espace disponible et oblige le joueur à optimiser son sac.
- La méthode **getGold**() permet de récupérer rapidement l'or présent dans l'inventaire.
- La méthode **check_boundaries**() vérifie si un objet peut être placé à un endroit donné sans dépasser la grille ni chevaucher un autre objet.

Cette classe est essentielle pour toute la gestion des items du jeu.

Classe : Dungeon

La classe **Dungeon** représente le donjon dans son ensemble. Elle contient plusieurs étages (floors) et gère la progression du héros.

- Le donjon est composé d'une liste d'étages.
- Il stocke la position actuelle du héros et l'étage en cours.
- La méthode **goNextFloor**() permet de passer à l'étage suivant.
- **isDungeonCompleted**() indique si le joueur a atteint le dernier étage.
- Lors de la création du donjon, les étages sont générés automatiquement.

Cette classe permet de gérer la progression globale de la partie.

Classe : Floor

La classe **Floor** représente un étage du donjon, composé d'une grille de salles.

- Chaque étage est une grille 5×11 de `RoomType`.
- Il contient :
  - une entrée, une sortie, des salles spéciales (ennemis, trésors, marchand, soigneur).
- Les salles sont reliées entre elles par des couloirs générés automatiquement.
- La génération est procédurale : chaque étage est différent.
- La méthode **allowedFromTo**() utilise un algorithme de parcours pour vérifier si un chemin est accessible entre deux points.

Cette classe gère toute la structure et la navigation d'un étage.

Classe : Hero

La classe **Hero** représente le personnage du joueur et centralise toutes ses statistiques.

- Elle stocke :
  - les points de vie, l'énergie, le mana,
  - le niveau et l'expérience,
  - la défense et la protection,
  - le sac à dos et les effets actifs.
- La méthode **takeDamage()** applique les dégâts en tenant compte de la protection.
- Le système de malédictions est géré via :
  - **applyCurse()**
  - **popCurse()**
- Le héros possède un **BackPack**, ce qui relie directement le gameplay des objets à ses statistiques.

Cette classe est le cœur du gameplay côté joueur.

Classe : Merchant

La classe **Merchant** représente un marchand avec lequel le héros peut échanger des objets.

- Le marchand possède un inventaire généré automatiquement.
- **sellItem()** permet au héros d'acheter un objet si il possède assez d'or.
- **BuyItem()** permet au héros de vendre un objet au marchand.
- Les échanges modifient directement la quantité d'or dans le sac du héros.

Cette classe introduit une dimension économique au jeu.

Classe : Healer

La classe **Healer** représente un soigneur.

Package : ressources

Le package **ressources** contient l'ensemble des ressources graphiques utilisées dans le jeu. Il regroupe toutes les images nécessaires à l'affichage, ce qui permet de séparer clairement le code logique des éléments visuels. Cette organisation facilite la gestion, la modification et l'ajout de nouveaux assets sans impacter le reste du projet.

resources/enemies

Ce dossier contient les images associées aux différents ennemis du jeu.
Chaque sprite correspond à un type d'ennemi et est utilisé lors des combats ou de l'affichage des salles contenant des ennemis.

resources/fond

Le dossier **fond** regroupe les images d'arrière-plan du jeu.
Ces images sont utilisées pour l'affichage général du donjon, des salles ou de l'interface, afin de donner une ambiance visuelle cohérente au jeu.

resources/hero

Ce dossier contient les images liées au héros.
On y retrouve les sprites représentant le personnage du joueur, utilisés lors de l'exploration et des combats.

resources/icones

Le dossier **icones** rassemble les icônes utilisées dans l'interface graphique.

Package : rooms

Le package **rooms** regroupe l'ensemble des classes représentant les différentes salles du donjon. Chaque salle correspond à un type précis (combat, soin, trésor, commerce, couloir ou mur) et définit les interactions possibles lorsque le héros s'y rend.

RoomType

L'interface **RoomType** définit le comportement commun à toutes les salles du donjon.

- Elle impose la méthode `getTypeName()`, utilisée pour identifier le type de salle.
- Toutes les salles du jeu implémentent cette interface, ce qui permet de les stocker et de les manipuler de manière uniforme dans la génération des étages et la navigation.

CorridorRoom

La classe **CorridorRoom** représente un couloir reliant différentes salles.

- Un couloir peut être une **entrée**, une **sortie** ou un simple passage.
- Il peut également être **verrouillé** (closed), bloquant temporairement le déplacement du héros.
- Le constructeur empêche qu'un couloir soit à la fois une entrée et une sortie.

Les couloirs structurent les déplacements et assurent la cohérence du cheminement dans le donjon.

EnemyRoom

La classe **EnemyRoom** représente une salle de combat.

- Elle contient une liste d'ennemis ainsi qu'une liste de récompenses.
- L'attribut isCleared permet de savoir si la salle a déjà été nettoyée.
- La méthode enemiesTurn(Hero hero) gère le tour des ennemis :
  - Les ennemis exécutent leurs actions (attaque, effets, etc.).
  - Les statistiques temporaires du héros (énergie, mana, protection) sont ensuite réinitialisées.
- La méthode chooseActionEnemies permet de sélectionner aléatoirement les actions d'un ennemi.

Cette classe centralise toute la logique liée aux combats dans le donjon.

HealerRoom

La classe **HealerRoom** représente une salle de soin.

- Un booléen permet de savoir si le soin a déjà été utilisé.

Elle apporte un aspect de gestion et de survie au gameplay.

MerchantRoom

La classe **MerchantRoom** représente une salle contenant un marchand.

- Le héros peut acheter ou vendre des objets en échange d'or.

TreasureRoom

La classe **TreasureRoom** représente une salle contenant des trésors.

- Elle stocke une liste d'objets générés aléatoirement.
- Certaines salles peuvent nécessiter une clé (needKey) pour être ouvertes.

WallRoom

La classe **WallRoom** représente une salle bloquée.

- Le héros ne peut ni y entrer ni interagir avec elle.
- Elle sert à structurer la carte et à remplir les zones non accessibles du donjon.

Package : ui

Le package **ui** regroupe toutes les classes liées à l'interface graphique et aux interactions utilisateur.
Il fait le lien entre les actions du joueur et la logique du jeu.
Ce package est responsable de l'affichage des écrans, de la gestion des objets graphiques et de l'interaction avec le héros, les ennemis et l'inventaire.

Button

La classe **Button** représente un bouton rectangulaire simple.

- Elle stocke sa position et ses dimensions.
- La méthode contains(x, y) permet de vérifier si un clic de souris se situe à l'intérieur du bouton.

DragSupport

L'interface **DragSupport** définit le comportement des éléments pouvant être déplacés à la souris.

- Elle impose la méthode onDrag(int x, int y).

EcranCombat

La classe **EcranCombat** est la plus importante du package **ui**.
Elle gère l'intégralité de l'interface de combat ainsi que les interactions associées.

Cette classe centralise donc l'essentiel du gameplay visuel et interactif.

EcranHealer

La classe **EcranHealer** représente l'écran affiché lorsque le héros entre dans une **salle de soin**. Elle permet au joueur de consulter ses statistiques, et bénéficier d'un soin unique fourni par le guérisseur.

EcranMenuPrincipal

La classe EcranMenuPrincipal correspond au **menu principal** du jeu.

- Affiche l'écran d'accueil.
- Propose les actions principales au joueur.

EcranRegles

La classe **EcranRegles** affiche les règles du jeu.

EcranPartie

La classe **EcranPartie** représente l'écran principal hors combat, utilisé lors de l'exploration.

- Affiche le héros et son sac à dos.
- Gère l'inventaire en dehors des combats.
- Permet au joueur d'organiser son sac.

Cet écran est central pour la mécanique principale du jeu : l'optimisation du sac à dos.

EcranMerchant

La classe **EcranMerchant** gère l'interface de la salle du marchand.

- Permet l'achat et la vente d'objets.
- Gère l'économie du jeu.

EcranHealer

La classe **EcranHealer** correspond à la salle de soin.

- Offre une récupération partielle du héros.

EcranTreasure

La classe **EcranTreasure** gère les salles de trésor.

- Permet au héros de récupérer des récompenses.

EnemyGraphique

La classe **EnemyGraphique** représente la version graphique d'un ennemi.

- Gère l'affichage d'un ennemi à l'écran.

ItemGraphique

La classe **ItemGraphique** est la représentation graphique d'un objet du jeu.

- Stocke l'image, la position écran et la rotation de l'objet.
- Gère la rotation par pas de 90°, utilisée lors de l'optimisation du sac.

MapOverlay

La classe **MapOverlay** représente la carte du donjon, affichée comme un écran superposé.

- Affiche la structure du donjon.
- Permet le déplacement du héros entre les salles.
- Gère les transitions vers les écrans correspondants.
- Position actuelle du héros affichée sur la carte.

Gestion de la progression

- Détection de la sortie du donjon.
- Passage à l'étage suivant.
- Affichage d'un message de victoire lorsque le donjon est terminé.

Interface Screen

L'interface **Screen** représente un écran du jeu.

Chaque écran (menu, combat, marchand, trésor, carte…) implémente cette interface.

- Définit une structure commune pour tous les écrans
- Garantit que chaque écran peut :
    - s'afficher, réagir aux clics, gérer les transitions

Classe ZenGameView

La classe **ZenGameView** est le cœur de l'interface graphique.
C'est elle qui lance le jeu, gère les événements clavier/souris et affiche les écrans.

- Lance la boucle principale du jeu (via la librairie Zen)
- Redirige les événements vers le bon écran

Pour lancer les commandes du build .xml il faut utiliser le logiciel ant comme target.