Documents Pour les SAEs S2 2024-2025

1 Auteurs

CAI Luc

ZAPOI Denis

DUPUIS Edin

LAY Jean-Christophe

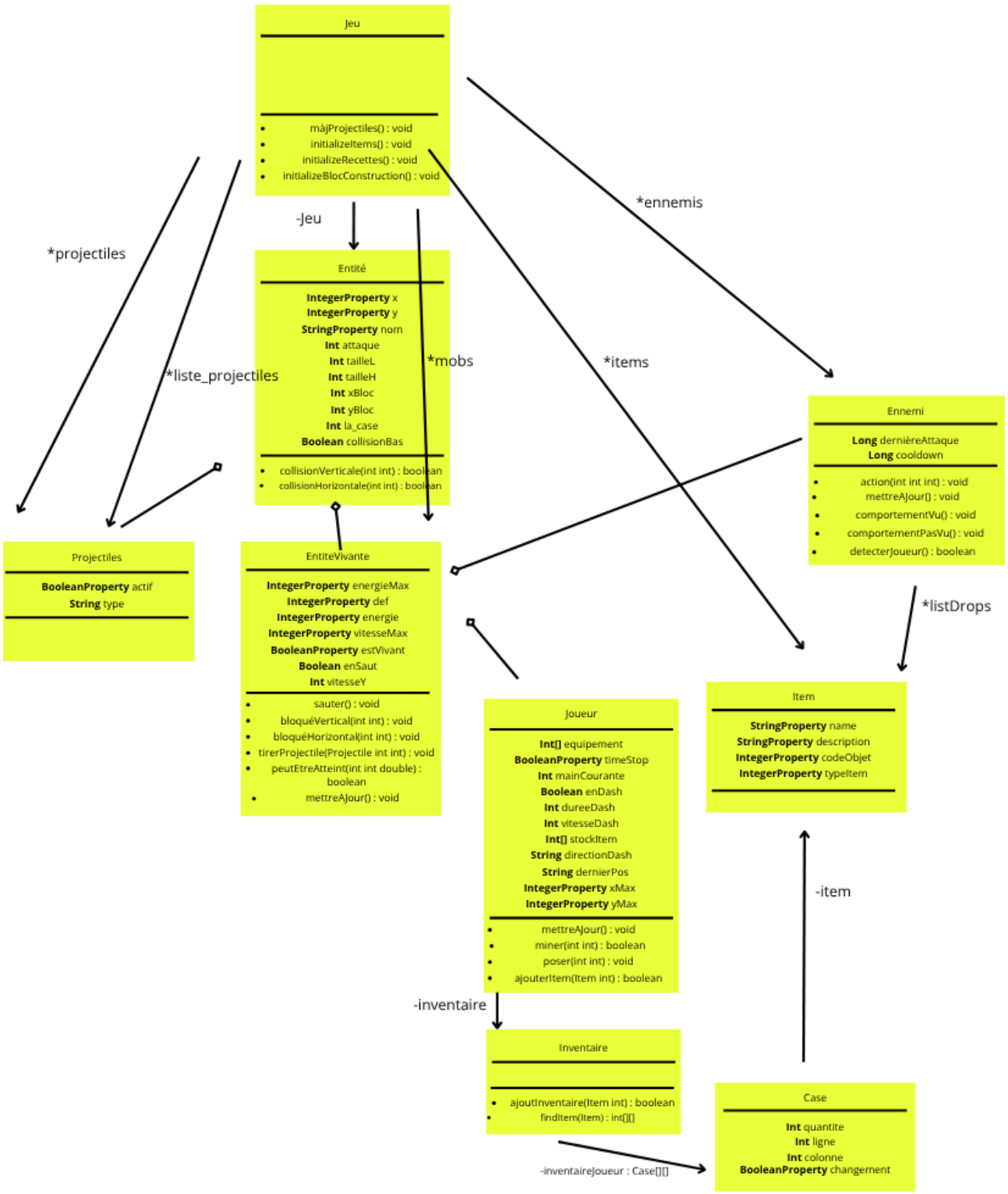
2 Diagrammes de classe (15 points pour SAEs S2.01 S2.02)

1 : Vous donnerez une vision d'ensemble de la partie modèle de votre programme à l'aide d'un ou de plusieurs diagrammes de classe commentés.

2 ; Vous choisirez des parties du modèle que vous considérez particulièrement intéressantes du point de vue de la programmation à objets (héritage, composition, polymorphisme…).

Vous expliquerez vos choix et les illustrerez par des diagrammes de classe.

Chaque diagramme utilisé doit être focalisé sur un objectif de communication. Il ne doit par exemple pas forcément montrer toutes les méthodes et dépendances, mais juste ce qui est nécessaire pour montrer ce que le diagramme veut montrer. Chaque diagramme doit être commenté



1) La classe Jeu représente l'environnement du modèle : Il possède des méthodes comme majProjectiles (qui permet de mettre à jour la totalité des projectiles encore en activité, soit qui n'ont pas encore disparu). Il possède également des méthodes permettant d'initialiser les éléments nécessaires du jeu comme les blocs de constructions, les autres items, les recettes…

➔ C’est la méthode qui permet à beaucoup d’autres classes (comme les contrôleurs) d’accéder au joueur (et à ses attributs), aux items, monstres… Il permet donc de centraliser le modèle.

La classe Entité est la classe mère des personnages de ce jeu Terraria : il consiste ici à gérer les méthodes de bases qui sont les collisions verticales et horizontales dans des situations simples (sans déplacements ou interactions avec d’autres personnages)

➔ Il se contente ici d’indiquer s’il y a une collision ou non sans modifier quoi que ce soit

La classe Entité vivante est une sous classe mère des personnages vivants : il va gérer une multitude de fonctions comme les interactions (collisions) beaucoup plus complexes que la classe entité

➔ Il va donc ici faire plus que vérifier s’il a une collision, il va aussi modifier des valeurs, appeler des fonctions pour permettre un comportement adapté du joueur ou du monstre, projectile (comme supprimer un projectile, arrêter le saut et vitesse en cas de collision du joueur…)

La classe Joueur est la classe principale pour tout ce qui est lié au joueur : il va avoir toutes les actions que le joueur peut faire (miner, poser, attaquer, construire un objet ou encore change l’objet qu’il a en main)

Les classe Inventaire et Case sont ceux pour tout ce qui est lié à l’inventaire du joueur : L’inventaire est un tableau à 2 dimensions de même taille que celui du joueur, de type Case (dont c’est lui qui a les données importantes pour chaque case de l’inventaire comme l’objet stocké, sa quantité…). Ces 2 classes gèrent également tout ce qui est ajout/diminution/suppression d’un objet dans l’inventaire, recherche d’un objet dans l’inventaire, et le système de craft.

La classe Ennemi quant à lui, gère les ennemis : c’est-à-dire leur action (attaquer) ainsi que leurs mouvements (grâce à A*) selon s’ils ont détecté le joueur dans leur champ de vision. (fait par comportementvu/pasvu et détecterjoueur, mais orchestré par mettreAJour)

Les classes Item et Projectile, quant à eux se contentent de gérer les fonctions simples des items et projectiles (setters et getters)

2) Séparation de entité et entité vivantes : on a utilisé des héritages ici car on a voulu séparer les fonctions et attributs communs à toute entité (monstres,joueur,projectiles) et ceux uniques aux entités vivants (monstres,joueur), comme la gestion de la vitesse, saut ou interactions complexes grâce à mettreAJour.

Séparation de l'inventaire en Inventaire et Case : on a décidé de faire une décomposition de cette fonctionnalité afin de bien représenter le concept d'inventaire : un inventaire est composé de cases = eux-mêmes ayant un objet et sa quantité. Ce qui permet de bien structurer cette fonctionnalité car la case gère uniquement les fonctions liées à lui-même (vérifier si l'item qu'il possède atteint sa limite maximale, ou enlever, incrémenter, changer un objet par exemple). L'inventaire quant à lui va seulement ajouter un objet, chercher la présence ou non d'objet particulier dans son inventaire, mais sans intervenir par lui-même dans les modifications de cases (il se contente « d'orchestrer » ces fonctionnalités de l'inventaire)

3 Tests Junit : (15 points pour SAEs S2.01 S2.02)

Une ou deux classes bien choisies devront être accompagnées de tests Junit cohérents et raisonnablement complets.

Voir dans le github pour les Tests JUnit

4 Structures de données (15 points pour SAEs S2.01 S2.02)

Mettre ici la liste des structures de données autres que ArrayList et tableaux utilisées et les classes où elles sont définies.

- Hashmap : Fond.java, SpriteItem.java, Jeu.java, ObsEnnemi.java,VueSon.java,BlocConstruction.java, ObsProjectile.java
- ObservableList : Jeu.java

5 algorithmique (30 points pour SAEs S2.01 S2.02)

Mettre ici la liste des algorithmes intéressants présents dans votre programme et les classes où ils sont définis.

- public boolean collisionVerticale() : dans Entite.java
- public boolean collisionHorizontale() : dans Entite.java
- public boolean peutEtreAtteint(int blocX, int blocY,double val) : dans EntiteVivante.java

6 Document utilisateur (40 points pour SAE S2.05) :

- Objectif et univers du jeu
- .Présentation du jeu avec captures d'écran.
- Mécanique des actions utilisateurs.
- Caractéristiques des armes et accessoires de Link, des types d'éléments de la map, des ennemis (tableau de relation ...).
- Toutes fonctionnalité définie et utilisable doit être documentée. A l'inverse il est hors de question (et donc pénalisant) de documenter une fonctionnalité non encore utilisable

Mettre ici votre document utilisateur (ou le joindre au dépôt dans un fichier pdf à part).

Voir dans le dépôt Github