Yanis SELLALI
Tony BOUVART
Emilie COATELANT

Hidden Hills

Introduction

Dans le cadre de notre projet, nous avons optés pour une architecture avec des superclasses et des classes abstraites. Dans le cas du personnage principal, nous avons décidés d'inclure dans une superclasse nommé "Personnage" tout les types d'objets vivants et dynamiques de notre projet. Ces objets partagent une multitude d'attributs et de méthodes, ce qui nous a mis d'accord sur comment nous allions organiser nos classes.
L'inventaire est quant à lui créé d'une manière similaire, une superclasse Item regroupant les différents types d'objets séparés en sous-catégories selon leur utilisations.

I. Architecture

Nous avons utilisés l'architecture MVC dans notre projet avec une séparation entre le contrôleur, la vue et le modèle. Cela permet de créer un affichage dépendant du modèle et permet une cohésion complète entre le modèle et la vue et une interaction entre eux via le contrôleur.

II. Détails : Diagramme de classe

1) Inventaire

L'inventaire est donc une ObservableList d'item attribuée au personnage (cf diagramme de classe Inventaire), dont le tri et l'organisation permet au joueur d'effectuer le choix de l'objet à utiliser.

1.1) Item

La classe Item permet de comptabiliser le nombre d'utilisation restantes à un objet ainsi que le nombre maximum d'un type d'objet à stocker dans une case. Elle est répartie en trois catégories : Tool, BlockItem et Furniture. Ces 3 classes possèdent des méthodes abstraites et sont également différenciées par le nombre d'exemplaire stockables par case de l'inventaire.

1.2) Craft

Le craft est une classe contenant les ressources nécessaires pour créer un objet via l'interface graphique de craft et également d'exécuter le retrait de ressources et la mise à jour de l'inventaire pour ajouter le nouvel objet créé.

1.3) Tools et Fourniture

Les Tools et Fourniture sont également séparés en classe unique pour chaque outil, étant donné que les utilisations faites peuvent être différentes selon l'objet. Ils sont ainsi tous accessibles par l'utilisateur qui peut interagir via les clics pour exécuter les différentes méthodes et fonctions des items.
Les tools remplissent une fonction d'outil à utiliser pour interagir avec la carte, et les fournitures sont des objets à poser avec lesquels on peut interagir en étant proche.

2) Personnage

La classe Personnage représente donc tous les objets actifs comme le Joueur (personnage principal) , ainsi que les Ennemis. Car chacun d'eux se déplace, attaque, perde ou gagne de la vie. Cependant, ils n'ont pas les mêmes comportement par exemple le Joueur et les Ennemis (ainsi que les Ennemis entre eux) ne font pas les mêmes dégâts, c'est pour cela que la méthode attaque est abstraite et que la classe l'est aussi.

2.1) Ennemis

La classe Ennemis est une sous-classe de Personnage mais une superclasse de tous les objets représentant les types d'Ennemis comme les Chauves Souris. Les Ennemis quant à eux sont dotés d'une IA, c'est à dire que leurs déplacements ou bien même leurs attaques dépendent du Joueur, ce qui permet de les différencier du Joueur.

2.1.1) ChauveSouris

La classe ChauveSouris est une sous-classe d'Ennemis, ce qui la diffère des autres Ennemis c'est notamment sa portée de détection, sa portée d'attaque, et ses dégâts.
La chauve-souris est également un ennemi volant, ce qui fait qu'elle n'est pas affectée par la gravité.

2.1.2) Zombie

Le zombie est un ennemi qui peut sauter et se déplacer au sol.

2.2) Joueur

La classe Joueur est également et seulement une sous-classe de Personnage étant donné que le Joueur est directement guidé grâce à l'interface homme-machine.

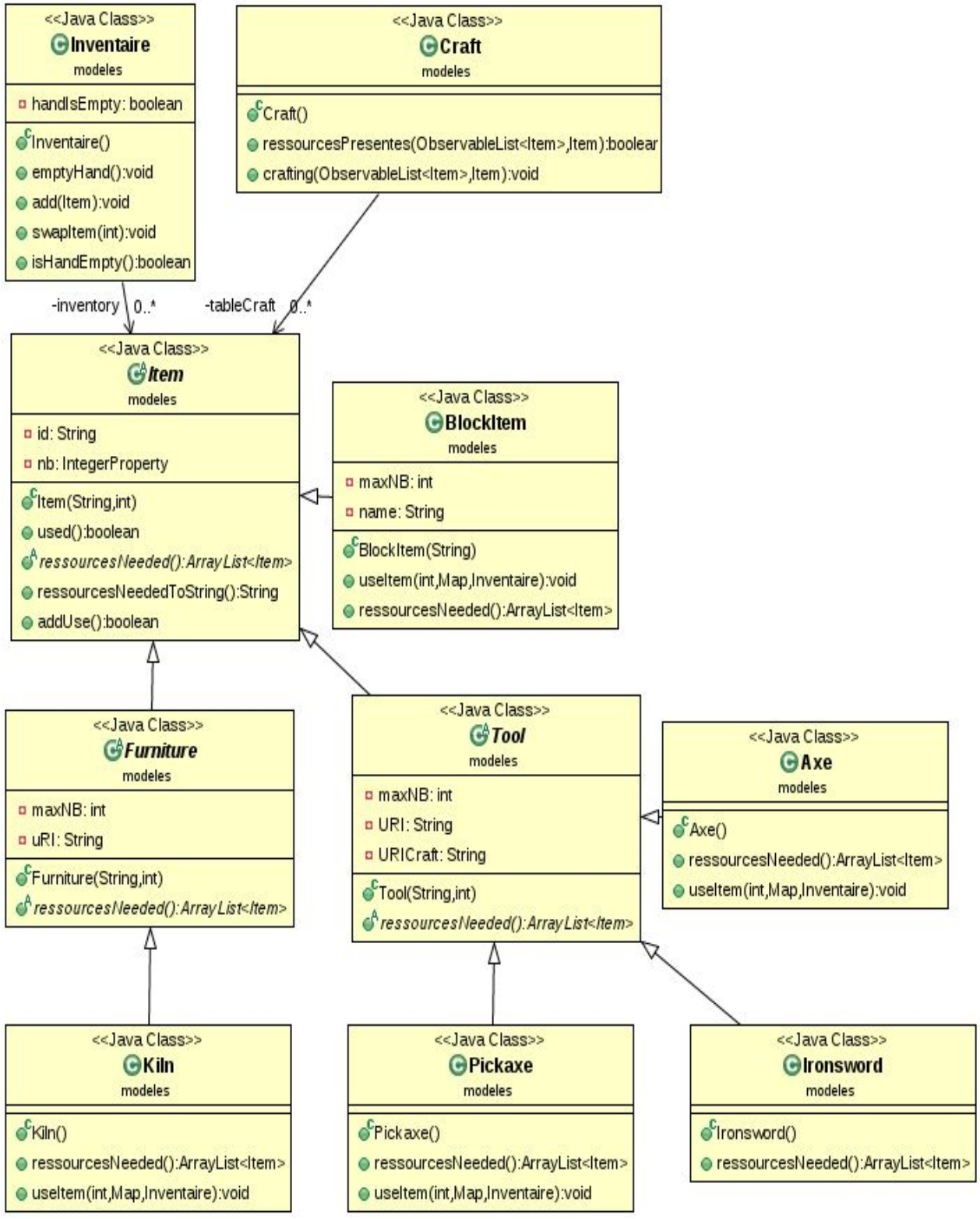

Diagramme de classe : Inventaire

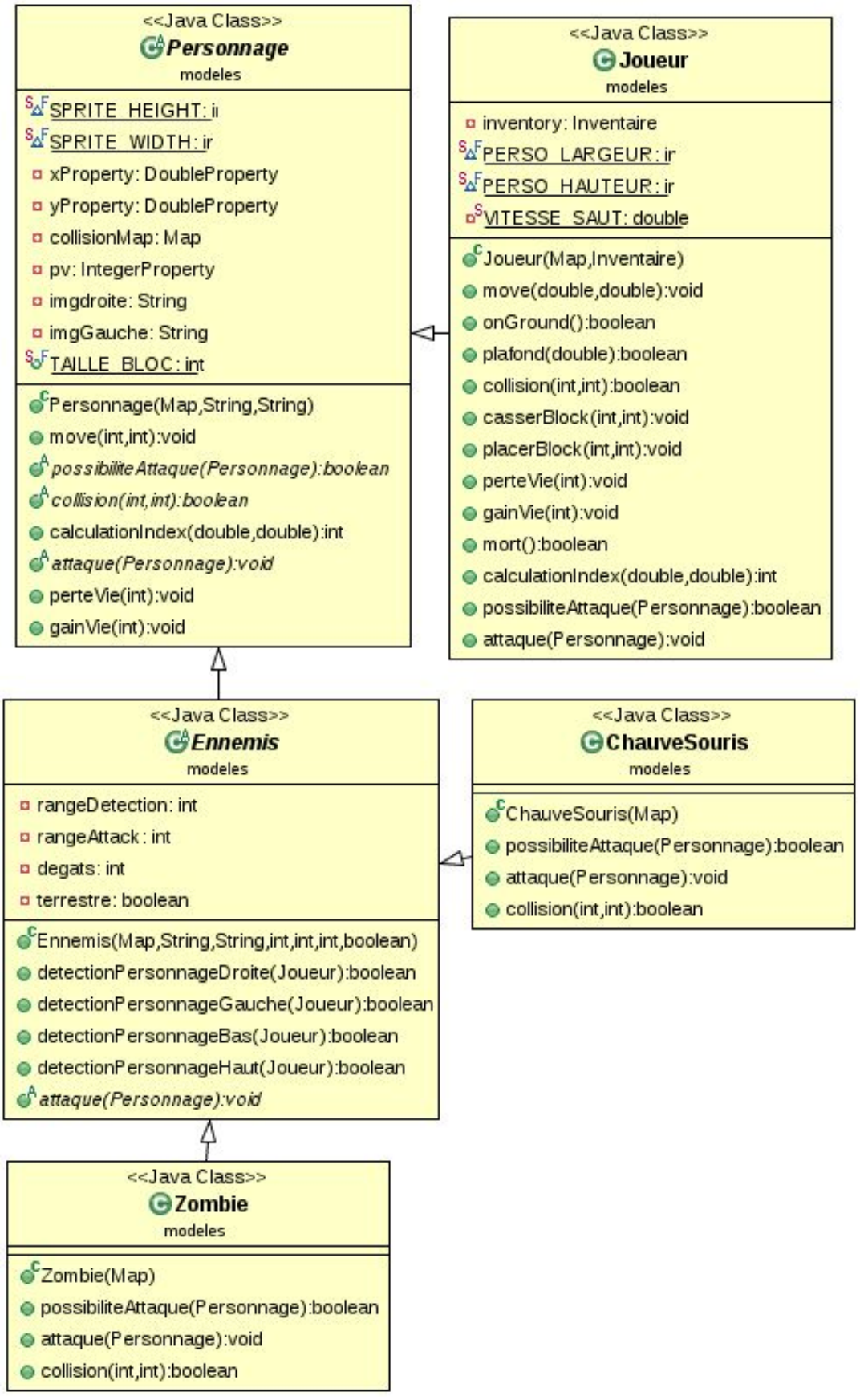

Diagramme de classe : Personnage

III. Diagrammes de séquence :

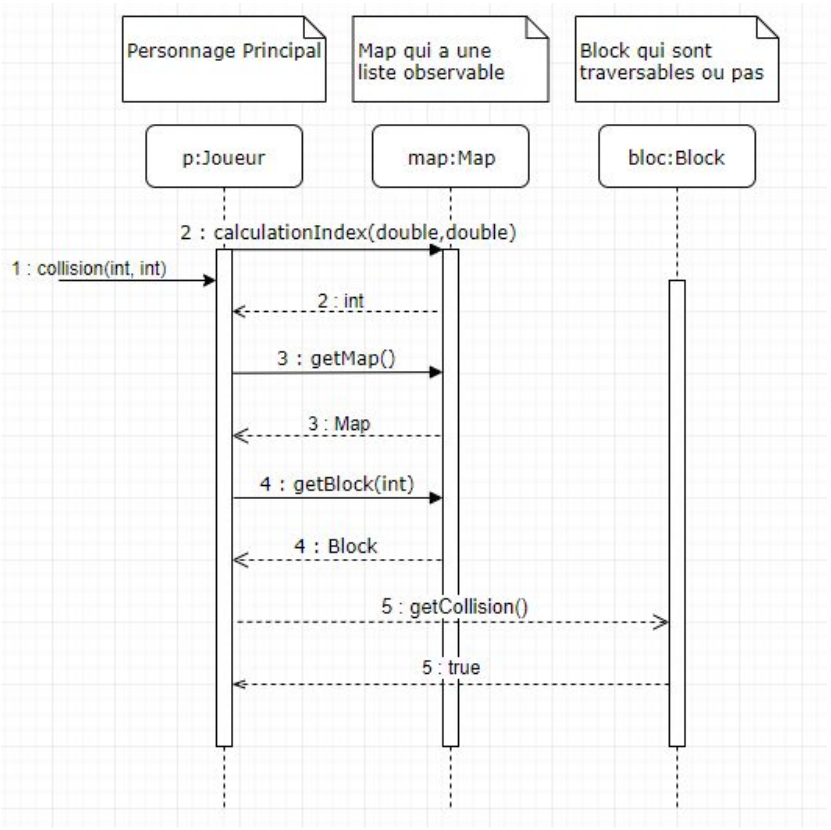

Diagramme de séquence : Joueur (Collision) présente

1 : On appelle la méthode collision(int,int) dans la classe Joueur qui retourne true lorsqu'il y'a un bloc non-traversable autour du Joueur (dépend de getCollision()).

2 : La méthode collision(int,int) appelle la méthode calculationIndex (double,double) afin d'obtenir la position du joueur en bloc qui est par la suite retournée.

3 : Ensuite, on récupère la Map du Joueur grâce à la méthode getMap() qui retourne la map du Joueur.

4 : Par la Map, on appelle la méthode getBlock(int) qui retourne un bloc à la position en paramètre qui est celle récupérer avec la méthode calculationIndex(double,double).

5 : Puis grâce, une fois le bloc en question retourné, on appelle la méthode getCollision() qui renvoie true si le bloc n'est pas traversable.

IV. Structures de données :

Nous avons utilisés les structures ObservableList afin de pouvoir interagir avec la vue lorsque nous modifions les tableaux afin de changer l'affichage.
Nous avons également utilisés BufferedReader et BufferedWriter pour lire et sauvegarder la map d'un fichier csv.
Nous avons aussi utilisés un MediaPlayer et Media afin de lire de la musique et des effets sonores dans le jeu.

V. Exceptions :

Nous avons utilisés le file not found exception afin de vérifier si le fichier entré était trouvable.

VI. Algorithmes :

Nous avons un algorithmes permettant à un ennemi de se déplacer vers nous et de vérifier via sa position et la nôtre si il est en capacité de nous attaquer.
Nous avons également un système de collision avec 3 méthodes, une vérifiant la collision au sol, une au plafond et une sur le personnage entier permettant de s'adapter au fait que le personnage soit en l'air ou au sol lors de saut ou de chute.

Document Utilisateur :

Univers :

Hidden Hills est un jeu de type sandbox en monde ouvert dont le but est l'exploration et l'appropriation d'un monde comme on le désire. Le personnage principal est un des habitants d'un village reculé et reçoit des objectifs des autres habitants du village qu'il doit remplir pour progresser, débloquer de nouvelles mécaniques et faire avancer l'histoire. La map est divisée en deux parties, celle du village, en sécurité et la partie extérieure, constituée d'une vallée inexplorée et dangereuse que le joueur devra apprivoiser.

Mécaniques :

Affichage de la carte :
La carte est représentée via la lecture d'un fichier csv qui permet de mettre en tableau la liste des blocs. Ce tableau est ensuite représenté sur la vue dans un TilePane. La carte choisie est modifiable en changeant la constante avec le nom de la carte.

Sauvegarde de la carte :
Lors de la fermeture de la fenêtre, une méthode est lancée permettant de réécrire le contenu du tableau dans le fichier ouvert à la base par la fonction d'affichage (méthode inverse de la lecture en quelques sortes).

Déplacement :
Pour se déplacer à gauche et à droite, le joueur peut utiliser les touches q pour aller vers la gauche, et d pour aller vers la droite, que ce soit au sol ou en l'air, à condition qu'il n'aille pas vers un bloc solide.

Scrolling :
La caméra suit le personnage lui permettant de se déplacer sur la carte globale en affichant une portée limitée.

Saut :
En appuyant sur la barre espace, le joueur peut effectuer un saut de 4 blocs sur une seconde pour atteindre des espaces en hauteur ou éviter des objets. Sa vitesse horizontale n'est pas modifiée, il peut donc choisir le côté ou il se déplace en l'air.

Barre d'inventaire :
Le joueur possède une barre d'inventaire se remplissant avec les objets qu'il collecte et peut intervertir les objets dans cette barre en cliquant sur l'objet à échanger avec l'objet en 1ère case qui est le seul à pouvoir être utilisé.

Casser blocs :
En utilisant un outil tel que la pioche, le joueur va pouvoir interagir avec certains blocs en effectuant un clic gauche afin de détruire le bloc qui sera remplacé par un bloc vide et ramassera le bloc détruit dans son inventaire. Dans une portée de 5 blocs.

Placer blocs :
En utilisant un bloc tel que le bloc de terre, le joueur va pouvoir interagir avec les blocs vide en effectuant un clic droit pour remplacer un bloc d'air par le bloc posé. Il ne peut pas remplacer un bloc déjà posé, seule les cases vides sont utilisables. Dans une portée de 5 blocs.

Gravité :
Tant que les personnages soumis à la gravité ne touchent pas le sol ils sont soumis à la gravité et descendent vers le bas sauf si une force opposée s'applique (ex : saut).

Crafting :
Le personnage peut utiliser les ressources dans son inventaire en cliquant sur une fenêtre de crafting située en haut à gauche. Les différents objets craftables sont :

- Pioche : 1 bois
- Epée : 3 pierres
- Hache : 1 bois, 2 pierres

Combattre :
En étant au corps à corps avec un ennemi, le personnage se mettra automatiquement à combattre.

Subir des dégâts :
En étant au corps à corps avec un ennemi, le personnage se mettra automatiquement à subir des dégâts des ennemis proches.

Déplacement des ennemis :
Si ils sont à portée, les ennemis se dirigeront automatiquement vers le joueur.

Mort du joueur :
En cas de mort, le joueur peut choisir de recommencer son aventure en générant une nouvelle fenêtre avec la carte avec laquelle il a lancé le jeu ou quitter pour sauvegarder.

Sons et musique :
Une musique de fond est jouée en permanence et certaines actions déclenchent des sons(dégâts, poser et casser blocs).