Tag : iteration-0

Itération 0

Dans cette itération, nous avons choisi de refactorer tout le code de la classe Labyrinthe afin de partir sur de bonnes bases où tout le monde comprend ce qui se passe.

Ainsi, notre objectif durant cette itération est d'ajouter ces fonctionnalitées :

- 1.1. Création d'un labyrinthe par défaut
- 1.2. Collision avec les Murs
- 1.3. Gestion des niveaux

Pour ce faire, nous avons pensé à faire en préalable :

- Transformer notre labyrinthe en un plateau de jeu représenté par un tableau 2D.
- Faire fonctionner les déplacements du personnages case par case.
- Mettre en place des variables globales pour faciliter l'adapatabilité du code.

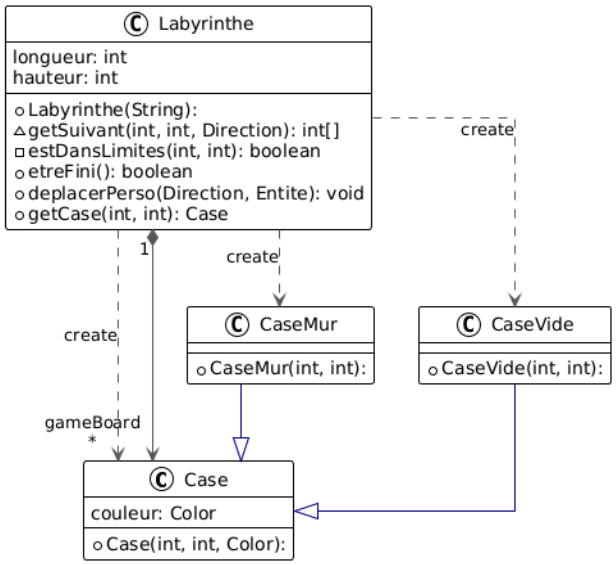
Nous prenons cette itération comme un moyen d'établir toutes les bases au bon fonctionnement du projet et pour faciliter l'ajout de futur fonctionnalité.

Étape 1. Transformer notre labyrinthe en un plateau de jeu représenté par un tableau 2D

Nous avons choisi de nous baser sur ce que non avons fait en cours de IHM Javafx sur le projet "Snake". Nous voulons donc représenter le tableau de jeu (`gameBoard`) par un tableau 2D d'une classe abstraite `Case`.

Nous avons pour l'instant que deux types de cases : une case vide et une case mur, qui sont représentées par deux autres classes héritant de `Case`.

Voici une représentation UML de ce que nous aimerions réaliser :



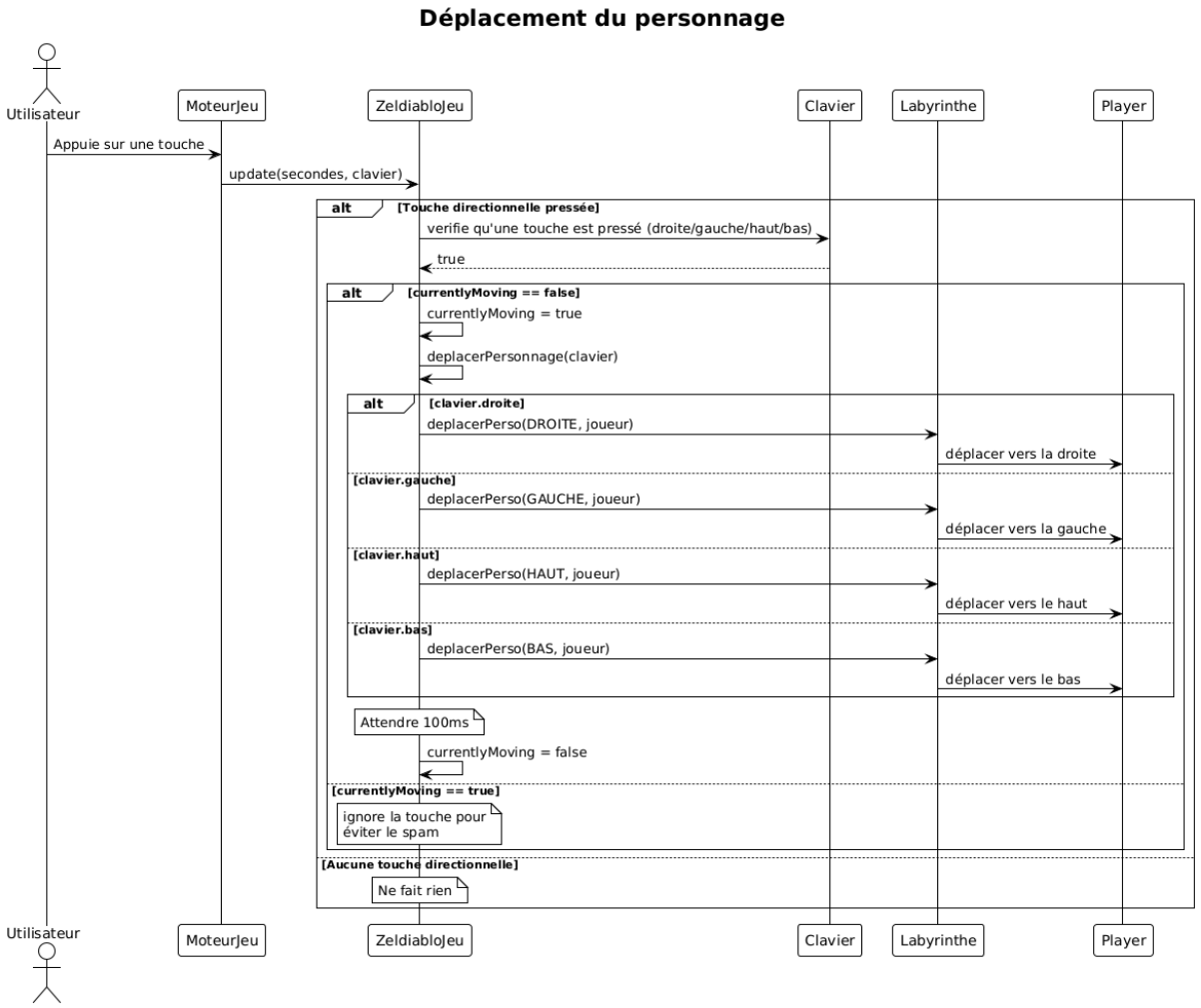
Etape 2. Faire fonctionner le déplacement case par case

Cette partie semblait terriblement simple au premier abord, quoi de plus simple que de faire se déplacer un personnage d'une case à l'autre? et bien rien sauf quand nous voulons nous déplacer d'uniquement une case.

Effectivement, le déplacement est simple sauf qu'il suffit d'une simple pression sur la touche de mouvement pour que notre petit personnage se prenne pour un jeune conducteur et ne fonce le plus rapidement possible dans les murs du niveau, cela est dû à la manière dont la méthode "update" est utilisée. Elle se contente de se lancer en boucle très rapidement et donc elle se déplace très rapidement.

Le moyen de pallier ce problème que nous avons trouvé est en fait très simple, nous avons utilisé la classe `ScheduledExecutorService` qui crée une classe anonyme exécutant le code qui lui est entré. A chaque entrée dans la méthode "update" un attribut booléen passe un test, si il le réussit il est tout de suite passé en faux. le code instancie un Scheduled qui le remet à faux dans 0.5 secondes puis modifie la position du personnage.

C'est ainsi que nous arrivons à nous déplacer de manière fluide et contrôlée. Nous avons concocté un diagramme de séquence correspondant .



Etape 3: Utilisation de variable globales pour un code plus propre

Afin de nous aider à créer un code qui soit à la fois lisible et efficace, nous utiliserons une classe VariablesGlobales contenant toutes les valeurs utilisables partout.

Réalisations

Nous avons réalisé la classe abstraite `Case`, ainsi que les deux classes `CaseVide` et `CaseMur` héritant de Case. Pour leur couleur, nous les avons mis dans le fichier `VariablesGlobales.java`.

Nous avons de même codé la méthode de déplacement de personnage.

Nous nous attardons pas plus sur cette itération de “set-up”, et commençons maintenant à faire une vraie première itération (celle là étant faite en dehors des heures de SAE).