

14 Décembre 2022

Rapport

Projet

Gyromite

Nicolas Planche

Théo Bastienne-Banco

P1805532

P2209159

Table des matières

Table des matières 1

But du jeu 2

la liste des fonctionnalités et extensions 2

Placement des éléments 2

Intelligence artificielle des ennemis 2

Gestion des collisions 3

Capacité de monter/descendre aux cordes 3

Capacité de faire se déplacer les Piliers + collisions Piliers 4

Bonus : Déplacement du personnage avec les colonnes 4

Système de points 5

Règles du jeu 5

Jeu collaboratif 6

Extension : Capacité de ramasser et déposer des Radis 7

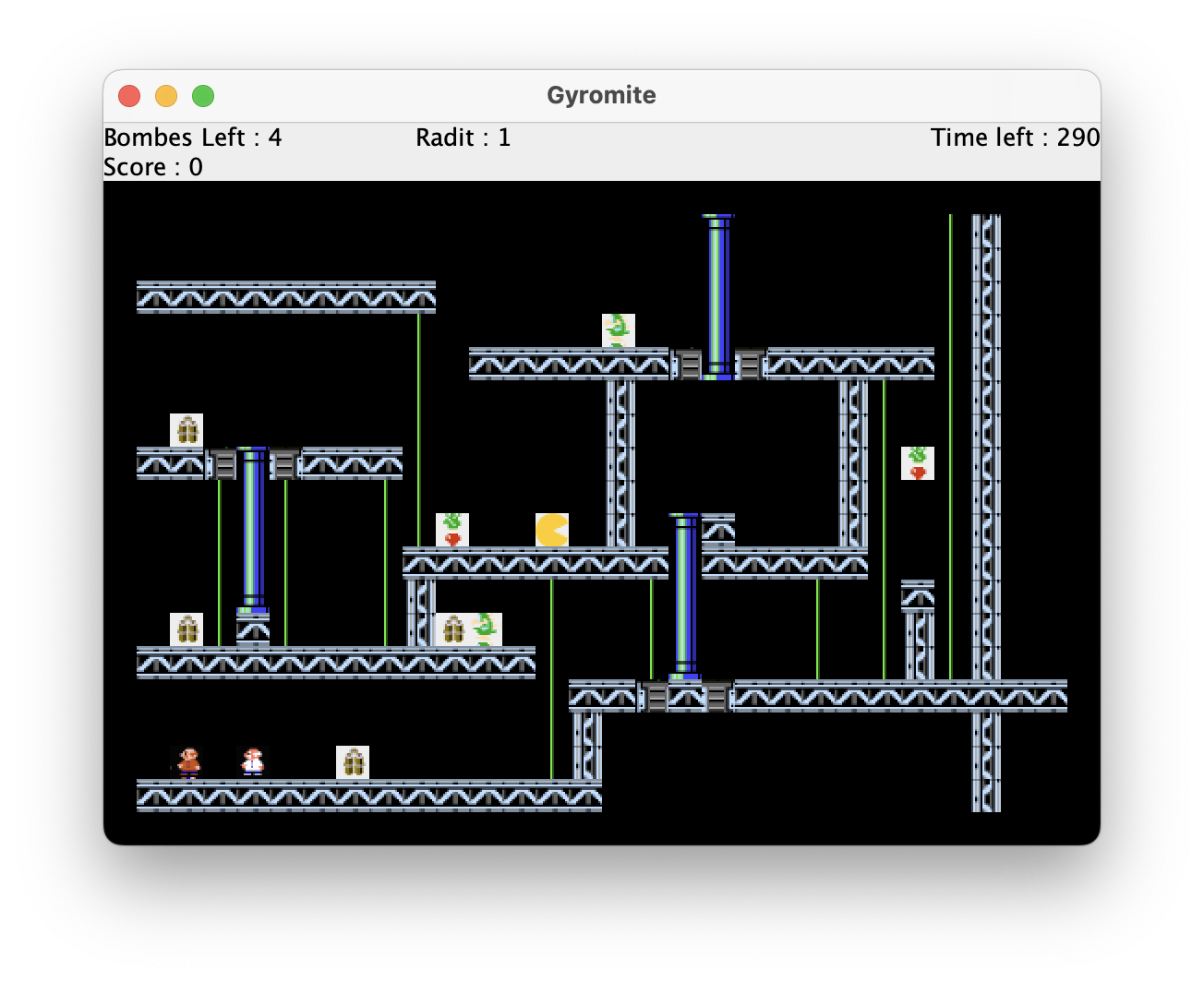
documentation UML 7

Gyromite

## But du jeu

On contrôle le professeur Hector qui se déplace pour ramasser les bombes dans son laboratoire, vu en 2D de côté. Le scientifique peut se déplacer à gauche, à droite, grimper ou descendre à l’aide de cordes. Le niveau contient des piliers bleus qui peuvent monter et descendre de manière régulière. Enfin, le professeur doit éviter les ennemis, appelés les Smicks. Dans cette version, il y a deux Hector en coopération. Ils peuvent chacun ramasser tout type d’objet (bombes, radis et bonus) et déposer des radis. L’objectif est donc de récupérer toutes les bombes, en évitant les ennemis, tout en déposant des radis pour créer des chemins si nécessaire pendant le temps impartie.

la liste des fonctionnalités et extensions

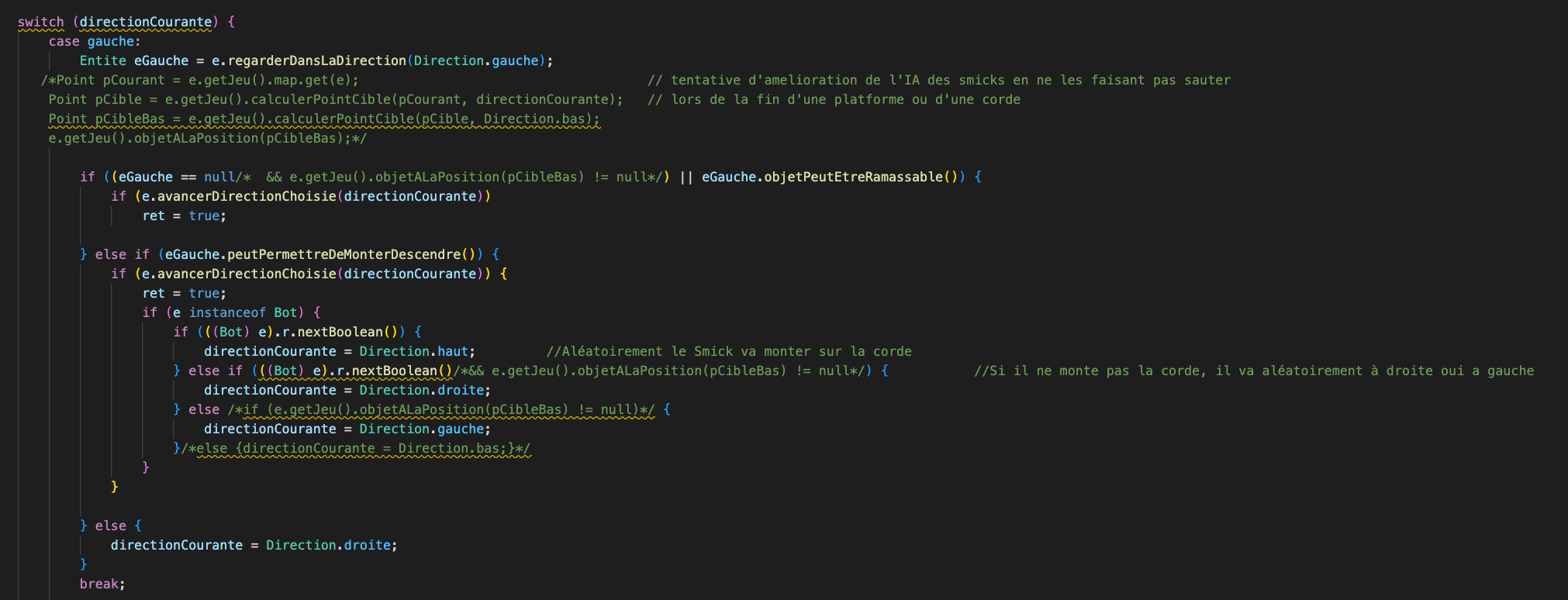


## Placement des éléments

**P**our placer les éléments, il a fallu ajouter chaque entité dans *Jeu.java* avec une gravité pour le(s) héro(s) et le smick. Nous trouvions qu’il était plus intéressant en terme de jouabilité de lui garder la gravité. Il a également fallu instancier les directions afin que chaque joueurs aient des touches de directions pour pouvoir le contrôler. Pour les bombes, radis et bonus nous avons créés une classe Ramassable avec trois type d’entité ramassable (afin de savoir si on peut le déposer). Pour les murs/sols/plateformes, ainsi que pour les entités ramassables, Nous avons ajouter un attribut type, permettant de différencier les entités, notament pour l’affichage des différents sprites.

De façon général, les éléments on été placés en « hard coder », en indiquant les coordonnées x y.

## Intelligence artificielle des ennemis



**L**intelligence artificielle est, tout comme les 4 directions du joueur, un réalisateur de déplacement. Il affecte un comportement randomisé aux smicks. Leur déplacement de droite à gauche sont différents celon les entités qu’ils rencontrent. Les bot peuvent monter les cordes, et choisir d’aller à droite ou à gauche si une collision les empêchent de monter d’avantage sur la corde, ce choix est déterminer de façon aléatoire par une fonction ramdom. Pour rendre le jeu un peu plus difficiles, nous avons fait le choix de permettre aux smicks de descendre des plateformes. Chaque bot à ca propre IA, leurs déplacement est donc indépendant un des autres. Ayant conscience que notre approche du 1 smick - 1 realisateur de déplacement est non optimisé, il aurai été préférable d’avoir une liste de smick répondant à une IA. En faisant attention à ce que chaque BOT est sa propre direction courante etc …

## 

## Gestion des collisions

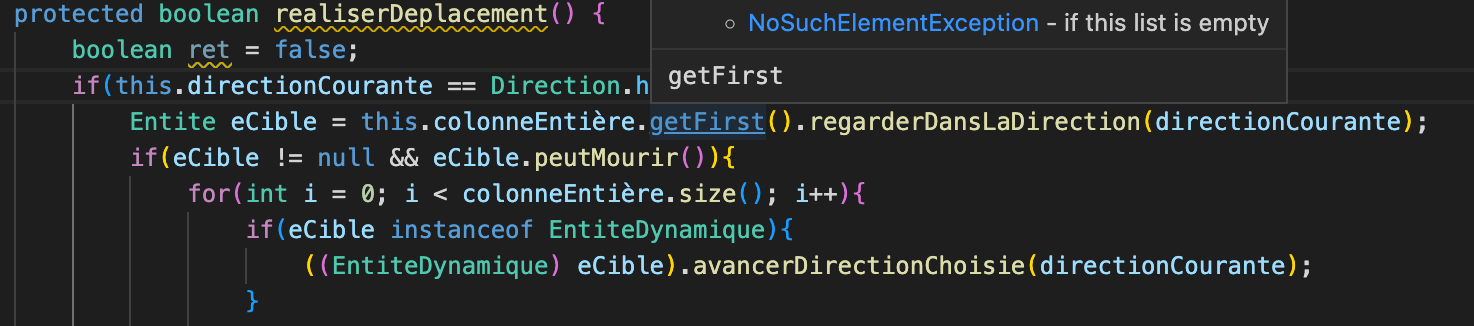
**t**outes les colision on été réalisé (Prof. / Murs, Prof. / Smicks, Smicks / Murs, Smicks / Smicks) Cela a été gerer dans les réalisateur de déplacements. Nous avons enlevé les colision entre les deu personnages jouable pour éviter les interblocages.

## Capacité de monter/descendre aux cordes

**L**e professeur peut monter, descendre aux cordes, les smicks peuvent aussi monter les cordes. Cela à été réaliser avec l’aide de la gravité notament. Et les réalisateur de déplacements.

## Capacité de faire se déplacer les Piliers + collisions Piliers

**L**es pilier ce déplace de façon indépendante, donc asynchrone. Cela à été realisé grace à une liste chainé, afin d’avoir le premier et dernier élément (entité colonne). Cela est codé dans un réalisateur de déplacement colonneDéplacement. Comme défini dans les consigne, les entités colonne sont déplacé une par une dans un ordre précie. Dans notre version du jeu, les smicks ne peuvent pas ce faire ecraser par les colonnes pour rendre le jeu un peu plus dur. Le héro ce fait tuer si un ce fait ecraser par le dessus ou le dessous d’un piller.



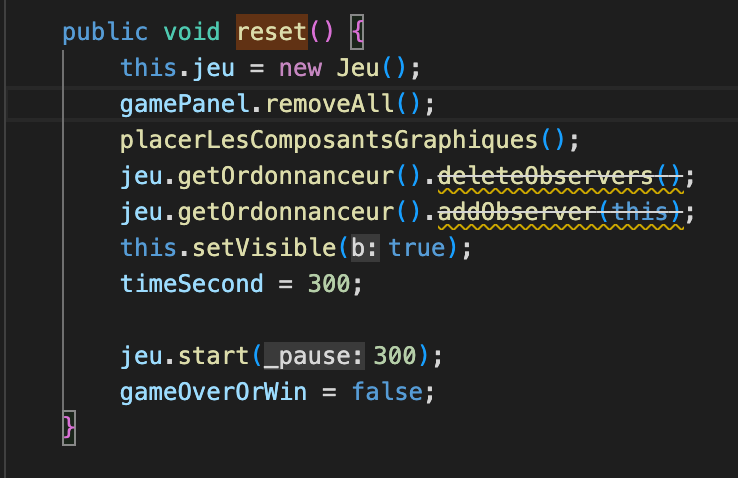
## Bonus : Déplacement du personnage avec les colonnes

**L**e hero et les smick sont déplacés vers le haut par les colonnes s’ils sont positionnés au dessus d’une colonne en mouvement. La colonne tue le héro s’il ce retrouve ecraser par le dessus.

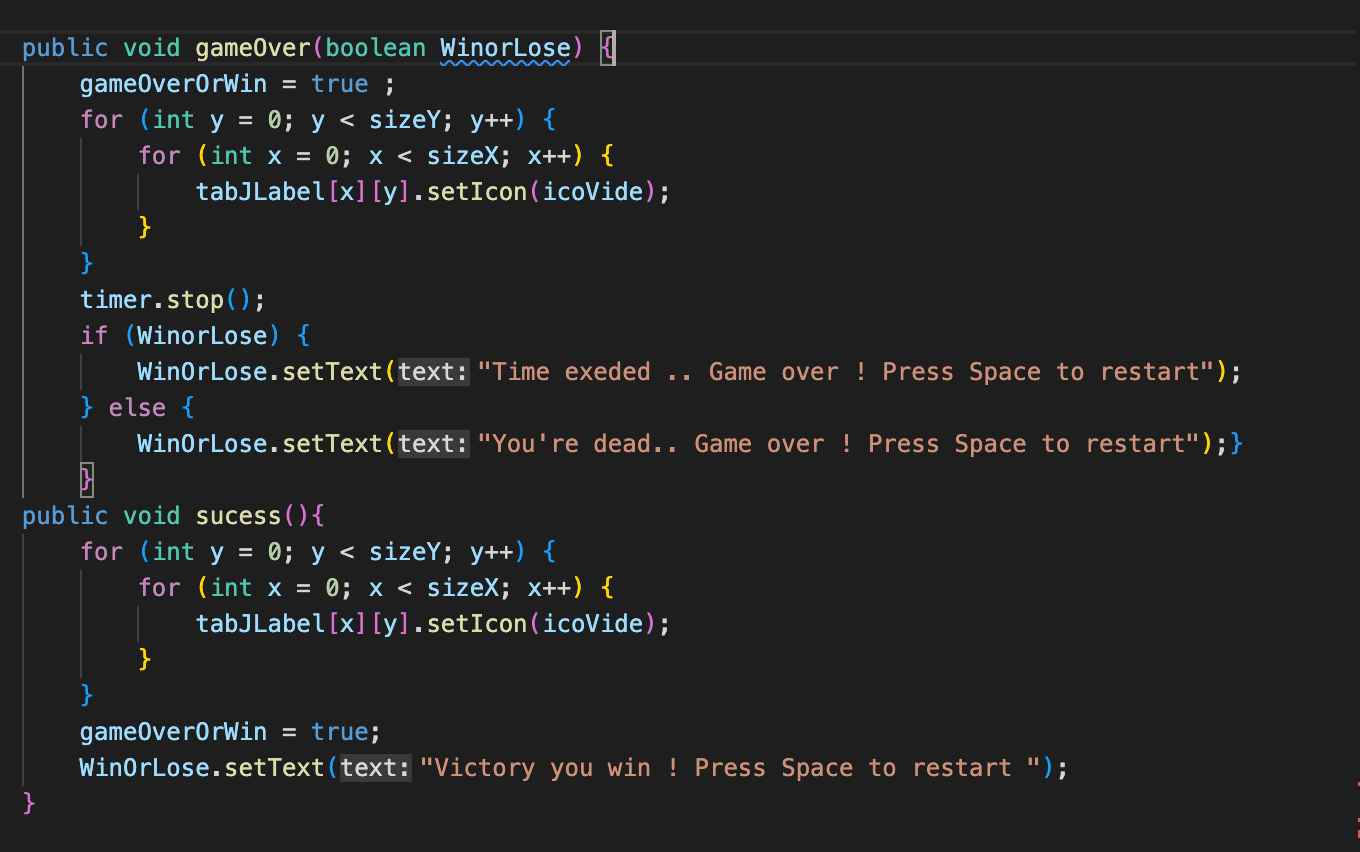
## Système de points

**L**e système de point réside dans le ramassage des bombes. Le score total s’affiche sur le dessus. Avec un petit plus de temps nous aurions mis en place un ajout de point lorsque le niveau est fini dans un meilleurs délai. **Le** système de point doit être à la fois affiché et incrémenté. Pour l’incrémenter, on ajoute une variable globale score dans *Jeu.java*, qui est incrémenté lorsque le personnage jouable entre en collision avec une entité *ramassable. O*n ajoute les points suivant le type d’entité, car elles ne valent pas les mêmes points. Pour l’affichage, on implémente un *JLabel score* qui récupère le score venant de *Jeu.java pour l’ajouter à un* JComponent (toolbar) afin d’afficher le score en le convertissant en un type *String.* Pour faire une incrémentation à l’affichage, on met à jour dans *mettreAJourAffichage* en récupérant le score actuel dan*s Jeu.java.*

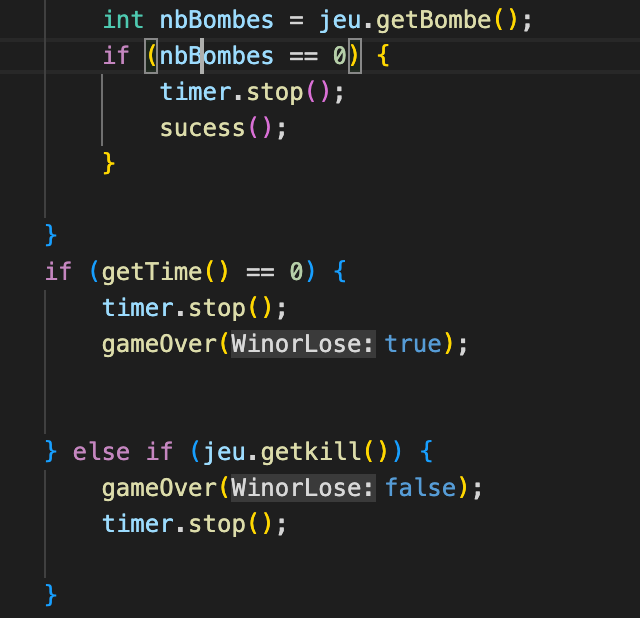
## Règles du jeu



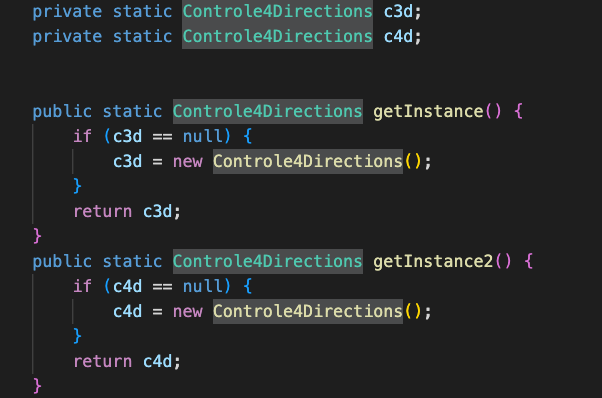
**P**our les règles du jeu, nous avons implémenté une fonction *Gameover* et *Success* dans Jeu.java pour savoir si la partie était gagnée ou perdue. Également pour avoir la possibilité d’arrêter le jeu et de le remettre à zéro. Dans l’éventualité, l’utilisateur peut appuyer sur la touche ESPACE (à la fin d’une partie) afin de relancer une partie ou ESCAPE pour fermer la fenêtre. Les fonctions *Gameover* et *Success* ont été également implémenté dans le fichier *VueControleurGyromite.java*. Afin de redémarrer le jeu, nous avons du arrêter tout les éléments lors du game over, l’ordonnancer, le temps, rendre l’écran noir.



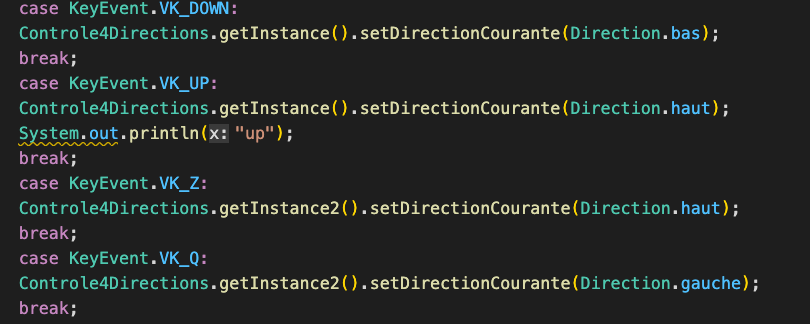
Ensuite, une fois la touche espace pressée nous avons créés une fonction reset permettant de redémarrer le jeu. Le ou les Hector peuvent perdre une partie de deux façons : soit par le temps arrivant à 0, soit en étant écrasé par un colonne ou touché par un *Smick*. Pour gagner, il faut pouvoir ramasser toutes les bombes.



## Jeu collaboratif

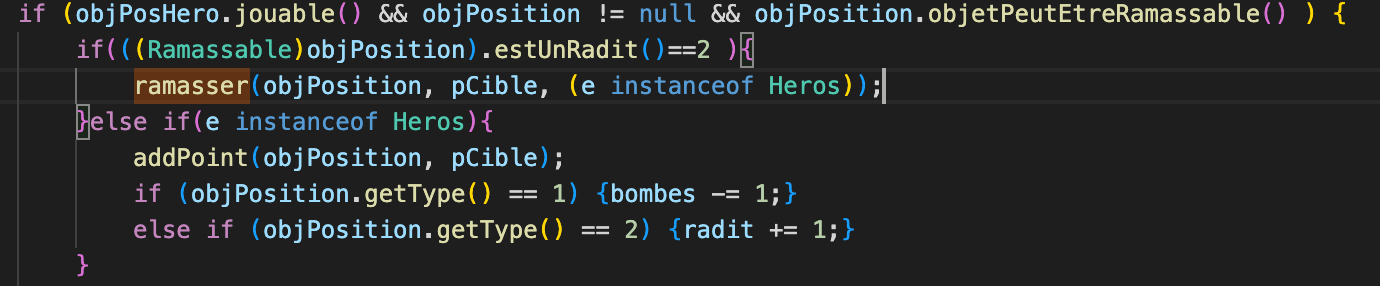


**C**reation d’un deuxième hector avec une deuxième image dans *VueControleurGyromite.java* afin de pouvoir jouer en coopération. Les deux joueurs sont donc en équipe. Pour réaliser cela, il a fallu créer un deuxième héros avec son *GetHero* dans *Jeu.java* et une deuxième instance différente dans *Controle4Directions.java* pour avoir la possibilité d’avoir des touches différentes pour le deuxième Hector. Il a fallut ensuite ajouter une condition dans la partie où le joueur perd pour avoir eu une collision avec une colonne ou avec un *Smick*. L’implémentation d’une fonction booléenne jouable dans *Heros.java* afin de vérifier si l’objet qui se déplace est jouable ou pas (en utilisant une direction sur lui-même) et si l’entité étant dans la direction voulu est jouable aussi ou non. Dans le cas où ils le sont tout les deux (deux Hector en coopération), ils ne se tuent pas et ils se passent donc au travers. Si ce n’est pas le cas (ex : une colonne et le deuxième hector), le joueur a perdu.



## Extension : Capacité de ramasser et déposer des Radis

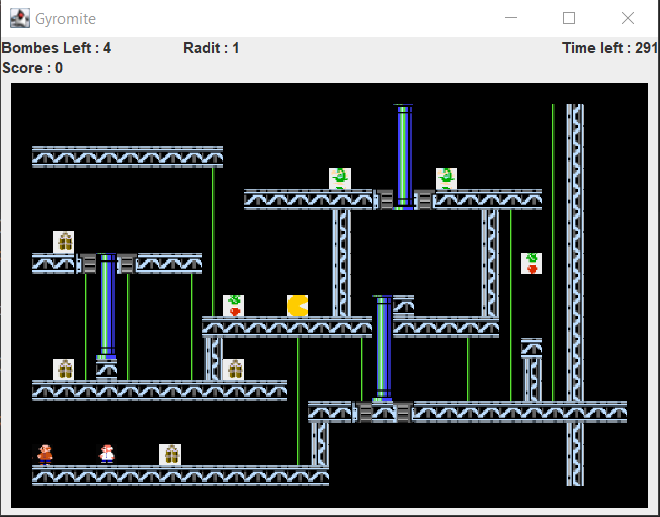
**D**éroulement des étapes. Création d’une deuxième fonction dropRadis pour permettre au deuxième Hector de déposer un radis avec une touche différente du premier Hector. Implémentation d’une fonction retournant le type d’entité *ramassable* dans *Ramassable.java* afin de déposer uniquement un radis au lieu des bombes et d’avoir un compteur de radis actif. Afin d’handicaper le *Smick* lorsqu’il croise un radis, on ajoute un délai avant que le *Smick* puisse bouger à nouveau. Il est capable de manger les radis mais nous avons enlever cette fonctionnalité.

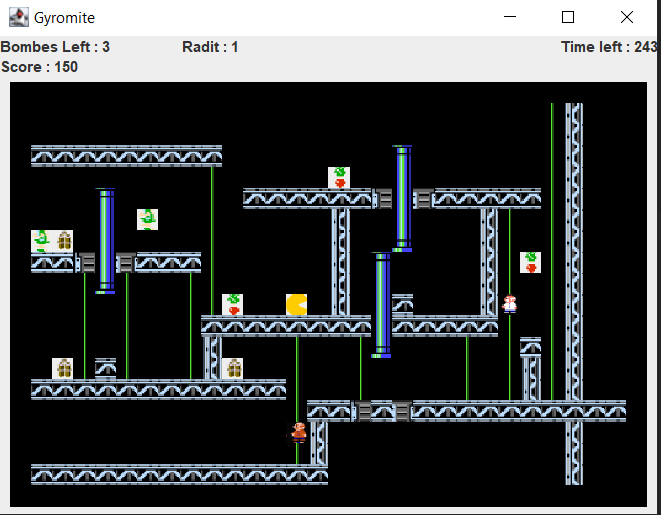


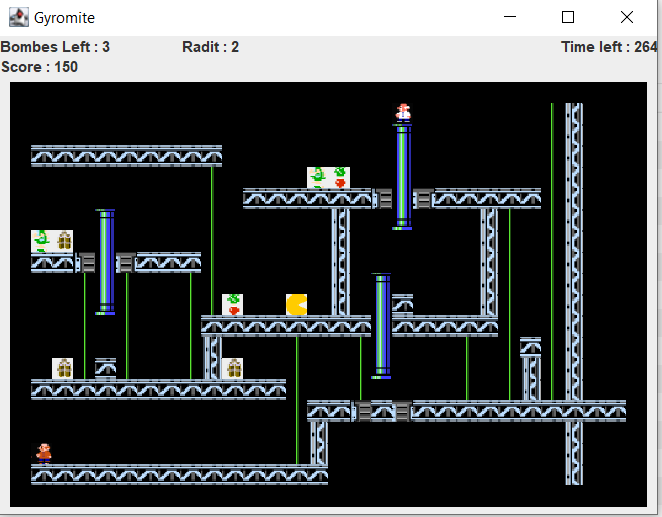
documentation UML

Nous avons repris le diagramme de classe donné avec le projet.

Partie en cours





d

