université PARIS-SACLAY

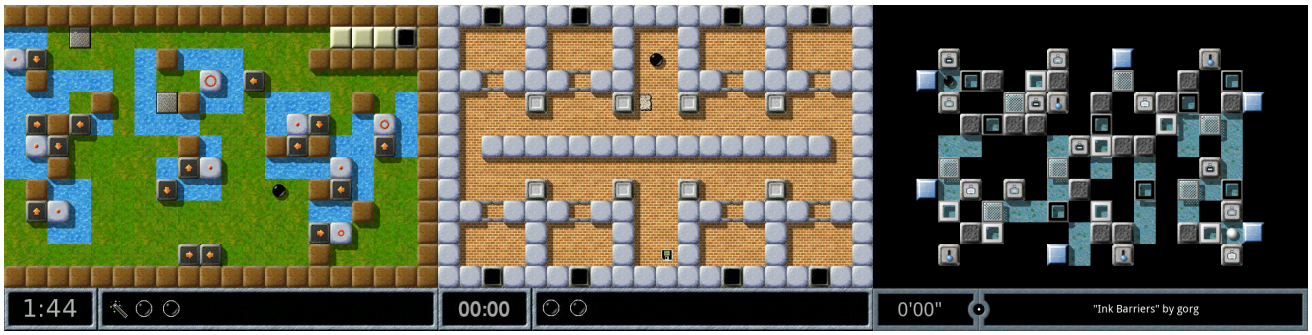
Projet de programmation objets

Cours de programmation objets

—Licence Informatique—

Redux

Le but de ce projet est de créer un jeu inspiré des jeux *Oxyd* et *Enigma*. Le projet devra être réalisé par équipe de 2 (binôme). Nous conseillons fortement que les deux membres de l'équipe soient de niveau similaire.



Oxyd est un jeu d'adresse et de réflexion, dans lequel le joueur contrôle une petite bille noire qui roule, touche des éléments pour les activer (les Oxyds s'ouvrent lorsqu'on les touche) et percute des objets pour les déplacer. Ce jeu qui ne tourne que sur des machines très anciennes, a été cloné sous le nom d'Enigma que l'on peut trouver sur `https://www.nongnu.org/enigma/`.

L'idée de base de Redux est simple : vous contrôlez une boule de billard noire (ou plutôt une sphère, vu sa fragilité et le bruit qu'elle fait en se brisant) qui doit trouver la sortie dans une sorte de labyrinthe. Certaine case du labyrinthe sont interactives par exemple : des blocs de bois mobiles, des lasers, des miroirs et des passages secrets. On trouve également des zones dangereuses, comme des gouffres, des sols friables (qui s'effondrent si la bille a roulé dessus plusieurs fois), des glissières, des bassins d'eau où l'on peut se noyer, des sables mouvants (dans lesquels la bille s'enfonce lentement) et divers pièges. Certains cases inversent les contrôles du joueur.

Nous vous proposons ici le jeu de base. À vous d'inventer de nouvelle possibilités et de créer de nouveau labyrinthe. On vous demande donc de créer un moteur de jeu simple mais suffisamment puissant pour laisser libre cours à votre imagination : conception de niveaux originaux, création de mécaniques inédites, détournement des règles existantes ou ajout d'énigmes plus complexes. Amusez-vous à concevoir, tester et affiner vos créations : le monde de Redux n'attend que vous pour s'enrichir de nouveaux défis.

Comportement de la bille

La bille est commandée par les mouvements de la souris, elle rebondis sur les murs et les obstacles et interagis avec les cases du labyrinthe.

La bille comporte une position (x, y) et une vitesse (v_x, v_y) qui sont des nombres réels (en Java : `double`). La vitesse absolue de la bille est donnée par la formule : $v := \sqrt{v_x^2 + v_y^2}$. À chaque tour de jeu, la bille avance en fonction de sa vitesse :

$$x = x + v_x \quad \text{and} \quad y = y + v_y \,. \tag{1}$$

La *direction de déplacement de la bille* est un vecteur de norme 1 qui est a les mêmes directions et sens que la vitesse de la bille. Ses coordonnées sont données par

$$d_x = \frac{v_x}{v} = \frac{v_x}{\sqrt{v_x^2 + v_y^2}} \quad \text{and} \quad d_y = \frac{v_y}{v} = \frac{v_y}{\sqrt{v_x^2 + v_y^2}} \,. \tag{2}$$

Note : La direction de déplacement de la bille n'est bien défini que si la bille se déplace. Si elle ne bouge pas on retournera $d_x = d_y = 0$.

Après le déplacement de la bille (si il a bien lieu), la vitesse est diminuée par frottement avec le sol. En fonction de la case, la vitesse est diminuée de f, sans changer la direction du déplacement. La nouvelle vitesse est donc 0 si $v \leq f$ et sinon :

$$v_x \leftarrow v_x - f d_x = v_x \left(1 - \frac{f}{v}\right) \quad \text{and} \quad v_y \leftarrow v_y - f d_y = v_y \left(1 - \frac{f}{v}\right) \,. \tag{3}$$

Quand le joueur déplace la souris, on mesure de combien la souris à été déplacée $(s_x, s\ y)$ et on ajoute ces déplacements à la vitesse après les avoir multiplié par un *facteur d'accélération* f_a. La nouvelle vitesse est maintenant

$$v_x \leftarrow v_x + f_a s_x \quad \text{and} \quad v_x \leftarrow v_y + f_a s_y \,. \tag{4}$$

On peut changer le *facteur d'accélération* pour faire des terrains où il est plus difficile de se déplacer, voir changer son signe pour inverser les commandes.

Le labyrinthe

Le labyrinthe prendra la forme d'un ensemble de cases organisées dans un tableau à deux dimensions. Chaque case peut être soit une case intraversable (un mur), soit une case ordinaire, soit une sortie. En plus de la bille, les cases peuvent contenir des obstacles, que l'on peu déplacer et/ou détruire.

On conseille de s'inspirer de l'architecture mise en place au TP7 (en rendant abstrait les classes qui doivent l'être si ce n'est pas déjà le cas).

Interaction de la bille avec le plateau

La bille peux interagir avec les cases du plateau de deux manières différentes :

— Soit en entrant dans une case (par exemple entrée dans la case de sortie fait gagner le jeu) ;
— Soit en touchant une autre case (par exemple toucher une case de mur fait rebondir la bille).

Pour pouvoir étendre facilement le jeu en ajoutant de nouveaux types de case, les interactions seront gérées par la classe abstraite `Square` qui déclarera, entre autres, les méthodes suivantes. Les méthodes seront bien sûr implémenter par la classe correspondant au type précis de la case :

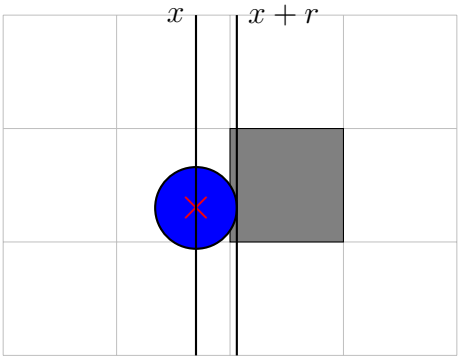
```
1  abstract public boolean isEmpty();
2  abstract public void enter(Ball b);
3  abstract public void leave(Ball b);
4  abstract public void touch(Ball b);
```

Note : Les types ci-dessus sont une indication. On peut rajouter des fonctions et des paramètres au fonction si besoin :

La case (i, j) du plateau est le carré $[i, i+1] \times [j, j+1]$. Pour savoir dans quelle case est la bille, il suffit de tronquer les coordonnées de la position de la bille en les convertissant en `int`. On peux ainsi savoir si la bille à changé de case pendant le déplacement en comparant la case avant et après le déplacement. On conseille de donner au plateau une méthode `getSquare` qui retourne la case du plateau en dessous des coordonnées données.

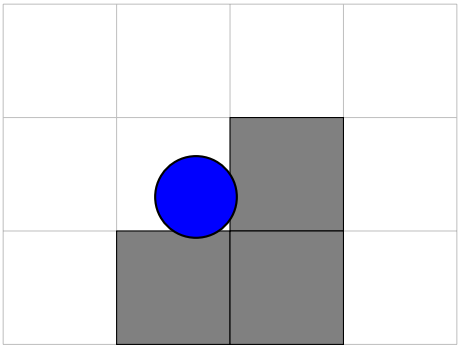
Pour savoir si la bille est rentrée en contact avec une case, il faut procéder en deux temps de la manière suivante. On va d'abord chercher s'il y a un contact avec le coté d'une case, s'il n'y a aucun contacts, on cherchera un contact avec un coin.

1. **Contact et rebond sur le coté d'une case**. Supposons que le centre de la bille est aux coordonnées (x, y) qui correspondent à la case (i, j). Le rayon de la bille est r. Il y a un contact avec la case $(i-1, j)$, si $x - r < i$. De même, il y a un contact avec la case $(i+1, j)$ si $x + r > i + 1$.



Dans le cas de contact, il y a un rebond si la bille se rapproche de l'obstacle. Dans le cas de la figure ci-dessus, c'est le cas si $v_x > 0$. Alors, pour gérer un rebond, il suffira de changer le signe de v_x avec l'affectation $v_x \leftarrow -v_x$.

Note : Il peut y avoir rebond sur deux bords (l'un vertical, l'autre horizontal) en même temps :

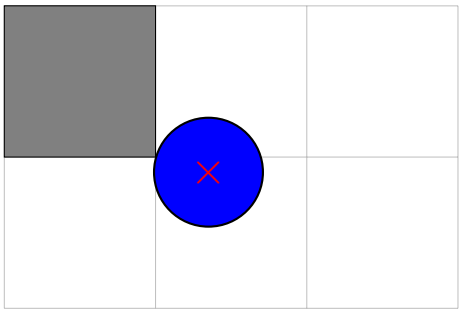


Dans le programme, on pourra gérer les deux rebonds l'un après l'autre en considérant qu'il y a deux rebonds indépendant, l'un vertical, l'autre horizontal.

2. **Contact et rebond sur le coin d'une case**. Tous d'abord, il ne peux y avoir rebond sur un coin que s'il n'y a *pas eu de rebond sur un bords*. Si l'on ne suis pas cette règle la bille va rebondir sur les coins entre les cases qui se touchent donnant des trajectoires irréalistes.

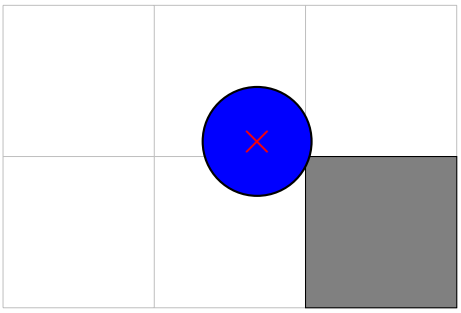
   Il y a un contact avec la case $(i-1, j-1)$ au point $(u, v) := (i, j)$, si

$$\sqrt{(x-u)^2 + (y-v)^2} < r\,.$$



   De même, il y a un contact avec la case $(i+1, j+1)$ au point $(u, v) := (i+1, j+1)$ si

$$\sqrt{(x-u)^2 + (y-v)^2} < r\,.$$



   Dans le cas, d'un contact avec un coin, la gestion du rebond est un peu plus difficile : On cherche la direction du contact : on note $r_c := \sqrt{(x-u)^2 + (y-v)^2}$ et

$$dc_x = \frac{x-u}{r_c} \quad \text{and} \quad dc_y = \frac{y-v}{r_c}\,. \tag{5}$$

   La vitesse de la bille dans cette direction est donnée par le produit scalaire :

$$v_{\text{coin}} := v_x \cdot dc_x + v_y \cdot dc_y \tag{6}$$

   La bille se rapproche de l'obstacle si v_{coin} est négatif. Si c'est le cas pour gérer le rebond, on va soustraire à la vitesse de la bille deux fois ce cette vitesse dans la direction du choc

$$v_x \leftarrow v_x - 2v_{\text{coin}}dc_x \quad \text{and} \quad v_y \leftarrow v_y - 2v_{\text{coin}}dc_y\,. \tag{7}$$

   ce qui aura pour effet de changer de signe dans la direction du choc.

Valeurs typiques des paramètres pour avoir un monde réaliste

Nous donnons ici des exemples de valeurs des paramètres. Ces valeurs sont indicatives pour que vous ayez une idée des ordres de grandeurs :

- rayon de la bille $r = 0.3$;
- vitesse typique $v = 0.02$, ne pas dépasser 0.2 pour avoir des déplacements réalistes ;
- facteur d'accélération $f_a = 0.001$ (accélération de la bille par pixel de déplacement de la souris ;
- facteur de friction $f = 0.005$.
- intervalle du timer entre $10ms$ (100 tours de jeu par seconde) et $50ms$ (20 tours de jeu par seconde). Attention : le code fourni en TP utilise la classe `javax.swing.Timer` qui est différente de `java.util.Timer`.

Exemple de case interactive

Le labyrinthe peut comporter un grand nombre de cases interactives différentes. La liste ci-dessous donne un certain nombre d'exemple. On ne demande pas de les implémenter tous !

- un trou dans laquelle la bille tombe faisant perdre le joueur ;
- une case qui inverse les commandes du joueur quand la bille rentre dedans ;
- une case qui électrocute mortellement la bille quand le joueur la touche ;
- des cases «téléporteur» qui vont par paire et transporte instantanément la bille sur la case associée quand la bille rentre dans la case ;
- une case patinoire où le joueur ne contrôle plus la bille qui se déplace en ligne droite ;
- inversement une case terrain lourd où il est difficile d'avancer ;
- une case sens unique où l'on ne peut entrer que par un bord et sortir par le bord oppose ; on rebondis sur la case si l'on essaye de rentrer par le mauvais bords ;
- une case tapis-roulant qui accélère le joueur dans une direction ;
- une avec un capteur qui ouvre une porte à un autre endroit du labyrinthe ;
- À vous d'imaginer d'autre type de cases...

Conseils pour la progression de votre travail

Nous vous conseillons de travailler par étapes. Veillez à bien sauvegarder votre travail (par exemple avec Git) à chaque étape du projet :

1. On commence avec un labyrinthe vide entouré de murs. Lecture du fichier décrivant le labyrinthe et affichage du labyrinthe ;
2. Affichage et déplacement de la bille à vitesse constante ;
3. Rebond de la bille sur les murs du bords de la grille ;
4. Contrôle de la bille par la souris ;
5. Frottement de la bille sur le sol ;
6. Ajout d'obstacle dans la grille et rebonds sur ces obstacles ;
7. Ajoute de nouveau type de case : trou et sortie ;
8. Améliorations éventuelles...

Organisation et évaluation du projet

Deux séances de TP (l'une dirigée, l'autre libre) porteront sur le projet. N'hésitez pas à poser des questions à vos encadrants de TP pour régler les problèmes que vous ne parvenez pas à surmonter. Il est indispensable de commencer à travailler sur le projet dès maintenant.

L'évaluation du projet se fera au travers d'une soutenance orale. L'organisation de la soutenance est la suivante :

— Avant votre horaire de passage, vous vous installez sur une machine, chargez votre programme et vérifiez rapidement que tout fonctionne comme la dernière fois que vous l'avez testé (il est de votre responsabilité de vous être assuré avant la soutenance que votre projet fonctionne sur la machine que vous utilisez pour la soutenance) ;

— les deux membres du binôme ont 3 minutes pour présenter ensemble le résultat de leur travail (organisation du code, visualisation des résultats, discussion sur les problèmes rencontrés...) ;

— ensuite l'examinateur a 7 minutes pour juger de votre maîtrise de la programmation. À partir de ce moment, vous serez considérés individuellement, et un effort important sera fait pour mettre en avant clairement votre implication dans le projet. Un étudiant ne peut prétendre être noté sur un code qu'il ne réussit pas à expliquer.

— Un étudiant absent à la soutenance aura zéro sur vingt.

Le principal critère d'évaluation est le bon fonctionnement de votre projet.

ATTENTION : Il vaut mieux avoir écrit moins de code s'il se comporte correctement, que toutes les fonctionnalités demandées où rien ne marche vraiment !

En particulier, rappelez-vous que «le mieux est l'ennemi du bien». Si vous souhaitez améliorer votre projet, soumettez-le (*submit* en anglais) dans Gitlab de manière à pouvoir revenir en arrière si vous n'arrivez pas à faire marcher la suite. Git retient toutes les versions que vous avez soumises. En cas de besoin, n'hésitez pas à demander sur le forum comment revenir en arrière dans l'historique.

Nous prendrons également en compte :

— Qualité de présentation...

— Types abstraits (tous les attributs sont privés) : pertinence, choix alternatifs, utilisation dans le code...

— Qualité du code : modularité, lisibilité, pertinence des commentaires...

— La qualité et la pertinence des tests...

— Originalité : gestion de problèmes rencontrés, affichage d'informations intermédiaires...

Attention ! Les critères ci-dessus ne seront pas évalués en regardant seulement le code, mais également par la manière dont vous répondez aux questions lors de la soutenance. Il est autorisé de se faire aider à comprendre ou débugguer certains points, mais il faut avoir *écrit soi-même, compris et être capable d'expliquer le code que l'on présente*. La note des deux membres du binôme est individuelle et dépend de leurs réponses aux questions et de leur implication dans le projet.