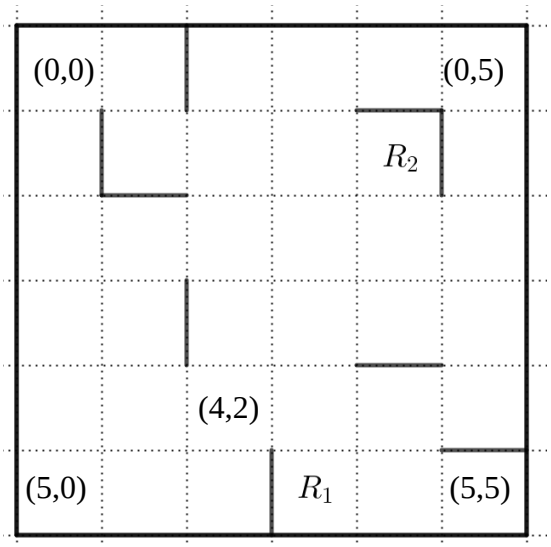
François AUBIN
Martin CANALS
Théo GIANI

Projet RICOCHETS ROBOT

Note technique N°3
Cette note annule et remplace la note N°1 en la complétant.
Janvier 2020

Sur la représentation des grilles de jeu.

1) La position d'une case est donnée par le tuple (x,y) de ses coordonnées .
x = numéro de ligne
y = numéro de colonne.
Exemple :
Repérage des cases du plateau



2) Le tableau est stocké sous forme de array numpy.

```
from numpy import array
grid = array([[1,2],[3,4],[5,6]) # convertit une liste de listes en array
grid.shape # tuple des dimensions (3,2)
```

T[i,j] contient un entier qui décrit la case (i,j) (voir point 4)

On a donc un stockage de la grille, ligne par ligne, en partant du haut.

3) Depuis la case **`T[i,j]`**
La case au NORD est **`T[i-1,j]`**
La case au SUD est **`T[i+1,j]`**
La case à l' EST est **`T[i,j+1]`**
La case à l'OUEST est **`T[i,j-1]`**

4) On utilise des constantes pour désigner les directions.
Ces constantes sont dans le fichier RRCONST.py

```
SOUTH = 0b0100
EAST = 0b0010
NORTH = 0b0001
WEST = 0b1000
```

T[i,j] contient un entier qui indique les murs autour de la case
Si la case (i,j) n'est pas bordée de murs alors **T[i,j] = 0**.
Si la case (i,j) a un mur au NORD et un à l'EST alors **T[i,j]=NORTH+EAST** (0b0011)

Pour tester si la case a un mur au SUD :

```
if T[i,j] & SOUTH :
```

5) Le déplacement d'un robot sur la grille.
Pour se déplacer un robot doit avoir connaissance du monde sur lequel il se déplace.
Ce monde est constitué de la grille de jeu et des autres robots. Le robot possède donc des références à ces objets.

Un robot étant a une position donnée **(i,j)** et voulant se déplacer dans une direction donnée **direction** appelle la méthode **cell_is_open** de l'objet **Grid** .
Cette méthode renvoie **True** si la case est ouverte dans la direction donnée et donc si le robot peut en sortir.
Le code de cette méthode est

```
def cell_is_open(self, pos, side) :
      return self.grid[pos] & side == 0
```

Si la case est ouverte le robot récupère la position de la case voisine.
Ceci se fait par l'intermédiaire d'une fonction **next_position** qui est construite par **step_builder**:

```
def step_builder(direction):
    """ renvoie une fonction qui calcule la prochaine position"""
    if direction == SOUTH:
        return lambda pos: (pos[0]+1, pos[1])
    if direction == NORTH:
        return lambda pos: (pos[0]-1, pos[1])
    if direction == EAST:
        return lambda pos: (pos[0], pos[1]+1)
    if direction == WEST:
        return lambda pos: (pos[0], pos[1]-1)

next_position = step_builder(direction)
```

La case voisine cible d'une position est donc

```
target = next_position(position)
```

Le robot fait appel à une autre méthode pour savoir si la case cible est libre :
Si la case est libre, la position du robot devient target

```
if robot.cell_is_free(target) : robot.position = target
```

La méthode est implémentée dans la classe Robot.