Épreuve blanche de type 2
12 mars 2021
Durée 5 heures

Épreuve proposée par Laure Gonnord, Nicolas Louvet, Nicolas Pronost

Aucun document autorisé en dehors des programmes de SNT et ISN.

Instructions **à lire attentivement!**

1. Les codes écrits seront **expliqués** (au besoin avec des exemples), *commentés*, indentés correctement. Lorsque la signature d'un algorithme n'est pas fournie, vous veillerez à bien justifier vos choix.
2. Les preuves seront **rédigées** avec soin.
3. Les réponses aux questions seront numérotées, et on évitera le grappillage.
   Dans toutes les questions, vous avez le droit de faire appel aux fonctions des questions précédentes, même si vous ne les avez pas écrites.

Partie 1
Enseigner la récursivité en classe de Terminale NSI (2 heures)

On s'intéresse ici à l'enseignement de la récursivité en classe de Terminale. Chaque fois qu'il sera nécessaire, vous vous reporterez au programme de Terminale, partie "Langage et Programmation" (pages 7 et 8).

1.1 Définition inductive et fonctions récursives

En classe de Terminale, on peut introduire la notion de fonction récursive en faisant un parallèle avec les définitions inductives, par exemple, la somme des entiers de 1 à n peut s'écrire de la façon suivante :

$$somme(n) = \begin{cases} 1 & \text{si n=1} \\ n + somme(n-1) & \text{sinon} \end{cases}.$$

On en déduit immédiatement une implémentation Python

```
def somme(n):
  if n == 1:
      return 1
  else:
      return n + somme(n-1)
```

dans laquelle l'analyse par cas de la définition est réalisée par la conditionnelle. À l'intérieur de la fonction, l'appel `somme(n-1)` fait référence à la fonction que l'on est en train de définir, c'est un appel *récursif*. La fonction est alors dite récursive.

Question 1

Dans un texte accessible aux élèves de Terminale accompagné d'un dessin de l'arbre d'appel, donner une explication de l'exécution de `somme(3)`.

Question 2

Donner une définition générale d'un cas de base et d'un cas récursif, et expliquer sur l'exemple.

Question 3

Donner 3 exemples de définitions récursives "classiques" qui permettent de montrer la diversité des cas de base, des cas récursifs, ...

Question 4

Quelles sont les conditions permettant de prouver la correction totale d'une fonction récursive" ?

1.2 De la récursivité sur les algorithmes de table

Dans le programme de Première NSI, l'algorithme de recherche dichotomique dans un tableau trié est présenté de manière itérative comme ci-dessous, et il est aussi demandé de savoir montrer sa terminaison sous la forme d'un *variant* de boucle.

```
def recherche_dichotomique(tab, val):
    gauche = 0
    droite = len(tab) -1
    while gauche <= droite:
        milieu = (gauche + droite) // 2
        if tab[milieu] == val:
            return milieu
        elif tab[milieu] > val:
            droite = milieu -1
        else:
            gauche = milieu + 1
    return -1
```

Question 5

Donner un exemple de couple d'entrée (tab, val) intéressant pour expliquer le déroulement de l'algorithme.

Question 6

Ajouter un *docstring* (contenant aussi des préconditions précises) et quelques commentaires explicatifs sur cette implémentation de l'algorithme de recherche dichotomique.

Question 7

Qu'est-ce qu'un variant de boucle ? Donner un tel variant pour cette implémentation.

Question 8

Prouver la correction totale de cet algorithme. Vous redonnerez la définition de la correction partielle et rédigerez comme pour une fiche correction à destination de vos élèves.

Question 9

Dans le cadre d'un cours, vous devez justifier l'importance de prouver la correction et la terminaison des algorithmes. Écrire *5 à 10 lignes* d'explications rédigées.

Question 10

On écrit le code à trou suivant comme préliminaire à une explication de la version récursive. Donner des explications de TD qui permettent de comprendre comment fonctionne cet algorithme et d'aider au remplissage des trous. Vous donnerez à la fin le contenu de ces trous.

```
def recherche_dicho_aux(tab, val, i, j):
    ...
        ... # cas de base: val non trouvée
    else:
        milieu = (i+j)//2
        if tab[milieu] == val:
            return milieu
        elif tab[milieu] > val:
            return recherche_dicho_aux(tab, val, ..., ...)
        else:
            return recherche_dicho_aux(tab, val, ..., ...)


def recherche_dicho(tab, val):
    return recherche_dicho_aux(tab, val, ..., ... )
```

Question 11

Expliquer à vos élèves pourquoi cet algorithme (récursif) termine.

Question 12

Citer un autre algorithme classique sur les tables qui s'écrit de manière similaire de manière récursive avec un cas de base et une fonction auxiliaire récursive (ou doublement récursive).

1.3 Récursivité et structures de données inductives : l'exemple des arbres

Définition des arbres, algorithmique On va dans cette partie traiter de la partie "paradigme de programmation", toujours du point de vue de la récursivité. On commence tout d'abord par faire remarquer que les programmes récursifs sont particulièrement adaptés au traitement des structures de données inductives, comme les listes ou les arbres. Les listes Python étant en fait implémentées par des tableaux dynamiques, les accès à des éléments quelconques sont permis par l'API liste en Python, ce qui ne rend pas l'exercice très compréhensible, alors on décide tout de suite de parler des arbres, et plus particulièrement des arbres binaires.

On rappelle ici une définition inductive (on peut dire récursive) des arbres binaires contenant des entiers aux noeuds :

```
type Arbre = Vide | Noeud(i:int, Arbre, Arbre)
```

Une *feuille* est un noeud qui n'a pas de successeur, ie `Noeud(untruc, Vide, Vide)`. Un noeud qui n'est pas une feuille est appelé *noeud interne*. Remarquons aussi qu'avec cette définition un noeud interne a toujours 2 fils (dont 1 qui peut être vide).
Ainsi la définition `Noeud(42, Noeud(1515, Noeud(70, Vide, Vide)), Vide)` correspond à l'arbre de la Figure 1.

Classiquement, étant donné un arbre `t` sous cette forme, on a accès à la valeur entière du noeud courant avec la notation `t.val`, ainsi qu'à son sous-arbre gauche (resp. droit) avec `t.fg()` (resp. `t.fd()`). La fonction `t.estvide()` renvoie `Vrai` si l'arbre est `Vide`, `Faux` sinon.

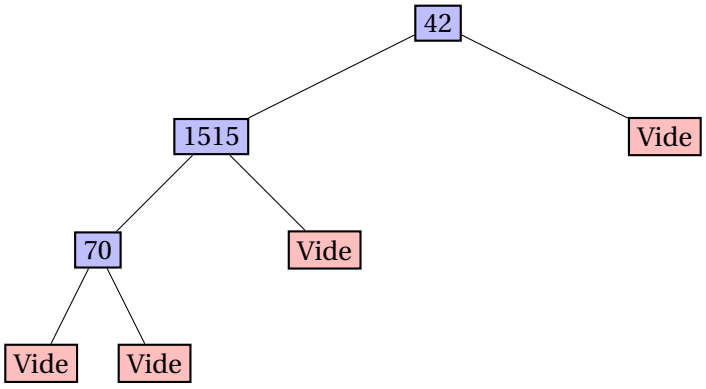


FIGURE 1 – Un arbre exemple

Question 13

Avec ces notations, écrire sans justification une définition récursive des fonctions suivantes :

(a) compter le nombre de noeuds *internes* d'un arbre donné.

(b) faire un parcours en profondeur infixe (qui effectue un traitement sur la valeur du noeud après avoir parcouru le sous-arbre gauche).

Attention ce n'est pas du Python qui est demandé ici, on demande quelque-chose qui ressemble à la définition de "somme".

Question 14

Donner une explication rédigée à destination de vos élèves de la définition de décompte des noeuds, en justifiant sa correction avec les critères de la question 4.

Implémentation Python Afin d'implémenter les fonctions récursives vues plus haut, on analyse ici les différents choix d'implémentation de la structure de données d'arbres binaires en Python :

— Dans les "listes" Python : l'arbre de la Figure 1 sera alors représenté par `[42, [1515, [70, None, None], None], None]`

— À l'aide de classes Python.

```
class Arbre :
    def __init__(self,val=None,fg=None,fd=None) :
        self.val = val
        self.fg = fg
        self.fd = fd
```

Question 15

Donner une ligne de Python permettant de déclarer une variable dont la valeur est l'arbre de la Figure 1 en utilisant l'implémentation par classe.

Question 16

Expliquer pourquoi dans les deux cas on peut construire des structures non conformes à notre définition inductive vue plus haut.

Question 17

Proposer des solutions pour chacune des deux implémentations, et les analyser.

On décide d'utiliser la classe Arbre pour implémenter quelques algorithmes classiques (ceux de la question 13)

Question 18

Un·e élève vous propose cette implémentation pour calculer le nombre de noeuds internes :

```
def nb_noeuds(self):
    if not(isinstance(self.val, int)):
        return 0
    else:
        return self.fg.nb_noeuds()+self.fd.nb_noeuds()
```

à l'exécution, il/elle obtient :

```
Traceback (most recent call last):
  File "tree.py", line 46, in <module>
    print(a2.nb_noeuds_bis())
  File "tree.py", line 31, in nb_noeuds
    return self.fg.nb_noeuds()+self.fd.nb_noeuds()
AttributeError: 'NoneType' object has no attribute 'nb_noeuds'
```

Expliquer cette erreur, donner une *indication* pour corriger. Critiquer aussi l'implémentation du cas de base et du cas récursif.

Question 19

Écrire une correction rédigée pour un programme qui affiche l'ensemble des noeuds internes d'un arbre, au fur et à mesure d'un parcours.

Le Petit Nicolas a écrit deux méthodes dans la classe `Arbre` pour parcourir un arbre et collecter dans une liste les valeurs présentes à chaque noeud dans l'ordre infixe. Il s'agit des méthodes `parcours0` et `parcours1` ci-dessous :

```
def parcours0(self):
  if self.fg == None and self.fd == None:
    return []
  else:
    return self.fg.parcours0() + [self.val] + self.fd.parcours0()

def parcours1_aux(self, l):
  if self.fg != None or self.fd != None:
    self.fg.parcours1_aux(l)
    l.append(self.val)
    self.fd.parcours1_aux(l)

def parcours1(self):
  l = []
  self.parcours1_aux(l)
  return l
```

Question 20

Dans ces deux méthodes, le Petit Nicolas teste si la méthode est appelée sur un cas terminal en comparant les données membres à `None` : pouvez vous lui proposer une manière plus claire de procéder?

Question 21

Comparez les mérites respectifs de ces deux parcours en terme de lisibilité et de complexité.

Question 22

Pour quel type d'arbre binaire est-il particulièrement utile de pouvoir collecter dans une liste des valeurs d'un arbre binaire dans l'ordre du parcours infixe?

Question 23

Justifier à vos élèves l'intérêt de passer en paramètre d'une fonction de parcours une fonction de traitement de données (programmation fonctionnelle [1]). Proposer une implémentation du parcours conforme à votre algorithme de la Question 13(b), de façon à pouvoir appeler :

```
t1.parcoursinfixe(lambda x:print(x))
```

pour imprimer votre arbre. Donner les limites d'application de votre implémentation.

1. On pourrait faire remarquer qu'il n'est pas clair que ce soit au programme de Terminale, mais bon.

Partie 2
Un projet de Terminale NSI autour du jeu Ricochet Robots
3 heures

Sur le modèle de l'épreuve 2 de 2020, en changeant le jeu.

Vous proposez à vos élèves de Terminale, en spécialité NSI [2], un projet qui réalise le jeu de Ricochet Robots. Dans ce projet, les élèves travailleront sur deux situations :

— Le joueur humain essaie une solution au problème et la machine vérifie que cette solution est correcte.

— La machine trouve la meilleure solution au problème.

L'objet de ce problème est de préparer une séquence de projet guidée, en suivant les étapes suivantes :

1. Étude des règles et d'une ressource pédagogique de niveau Collège.
2. Étude d'un code existant : structures de donnée, infrastructure, pour coder un jeu et les mouvements.
3. Étude d'une résolution simple existante et adaptation pédagogique.
4. Proposition d'une séquence et d'une évaluation associée.

Une rapide recherche sur le Web vous fournit les trois ressources suivantes disponibles en annexe :

1. Une page wikipédia en Français, avec les règles du jeu.
2. Une fiche à destination d'enseignant·e·s autour d'expérimentations pour comprendre les règles du jeu simplifiées.
3. Un dépôt github dans lequel on sélectionne un fichier Python.

2.1 Recherche web, étude critique de ressources et des règles

Cette première partie étudie les deux premières ressources (Wikipédia, et la ressource à destination d'enseignant·e·s).

Question 1

Quel sous-problème du jeu ricochet est intéressant à étudier/coder, et pourquoi ? *deux lignes*

Question 2

Déduisez ce qu'est un bloqueur de la fiche enseignante, et proposer une formulation claire de son utilisation *deux lignes.*

Question 3

Étudier/critiquer la séance de prise en main des règles proposée par la fiche enseignante. Quelles compétences de niveau Seconde SNT sont mises en oeuvre dans cette séance ? Comment bien choisir les exemples ?

2. En annexe vous trouverez un extrait du programme de Terminale NSI

On désire maintenant améliorer cette fiche, notamment en donnant un format “textuel” pour décrire une solution potentielle et la vérification de cette solution pour une configuration donnée du jeu.

Question 4

Proposer à vos élèves une façon de décrire (sur papier) une solution dans le cas simplifié. *S'appuyer sur un exemple représentatif.*

Question 5

Proposer un texte à trou de bilan de votre séance qui explique comment vérifier une solution (décrite suivant le format de la question précédente) dans le cas simplifié. *Rédiger la solution de ce texte en rayant ce que vous supprimez dans celui-ci pour produire la version élève.*

Question 6

Dans cette séance, on s'aperçoit qu'il est plus simple de vérifier une solution que d'en calculer une. Donner 3 autres exemples simples *concrets* compatibles avec le programme de Terminale NSI de tels problèmes. *1 ligne par exemple.*

2.2 Prise en main et étude critique d'un code existant

Nous allons maintenant étudier le code source trouvé. Dans le dépôt sont présents les fichiers suivants :

— Le fichier `main.py` permet de lancer le logiciel et son interface graphique (ce fichier n'est pas fourni ici).
— Le fichier `model.py` comporte les structures de données nécessaires à la construction d'un plateau, le codage d'une configuration de jeu, et fournit une première fonction de résolution simple.
— Le fichier `ricochet.c` implémente une résolution plus efficace (et `ricochet.py` fournit un *binding* pour appeler les fonctions C de la résolution depuis Python : ce fichier n'est pas fourni ici.). Ce code n'est pas fourni ici.

Le code n'est absolument pas documenté (ni commenté) à part un README succinct d'utilisation de l'interface graphique.

Passage de Python 2 à Python 3 Le code Python est en Python2, on désire tout d'abord résoudre ce problème. Voici la première erreur que l'on obtient lors de l'exécution du code récupéré avec un interprète Python 3 :

```
$> python3 main.py
  File "main.py", line 21
    print ', '.join(''.join(move) for move in self.path)
          ^
SyntaxError: invalid syntax
```

Question 7

Comment “résoudre” cette erreur d'exécution ?

Ensuite, on obtient cette erreur :

```
 main.py:125: Deprecation Warning: an integer is required (got type float).
Implicit conversion to integers using __int__ is deprecated,
and may be removed in a future version of Python.
   dc.DrawCircle(x + size / 2, y + size / 2, size / 3)
```

Question 8

Comment “résoudre” cette erreur d’exécution ?

Après quelques autres manipulations de la sorte, nous arrivons à faire fonctionner l’affichage ainsi que la résolution d’un plateau “difficile” en 25 coups, comme dans la Figure 2. L’affichage, ainsi que la partie d’interaction avec l’utilisateur ne sont pas étudiés dans ce sujet. On pourra se poser la question de la pertinence de fournir ces lignes de code aux élèves, plus tard, dans la partie de synthèse. **Dans la suite, on se rapportera au code en annexe.**

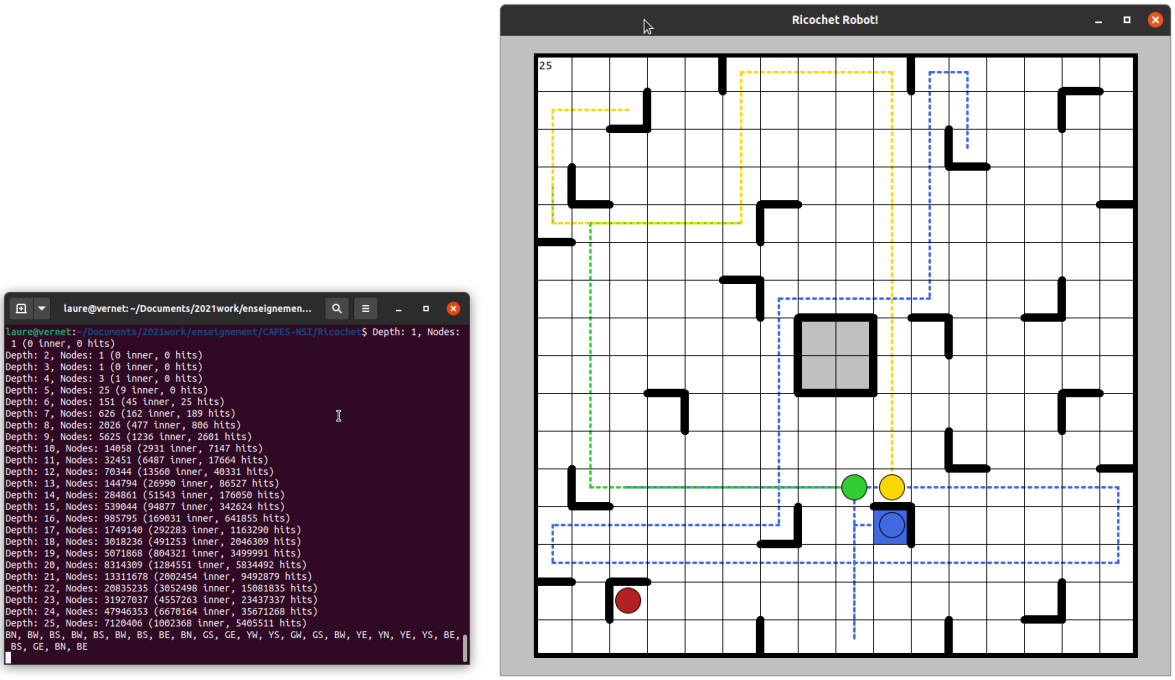


FIGURE 2 – Affichage graphique et résolution d’un plateau difficile

Codage d’une configuration, et des déplacements (model.py) Une configuration ici est la donnée d’un plateau de jeu et les positions de chaque robot (cf transparent “structures de données”). Le jeu codé ici est celui à 4 robots (de 4 couleurs différentes) et 16 objectifs appelés “tokens” (rond rouge, triangle rouge, . . .).

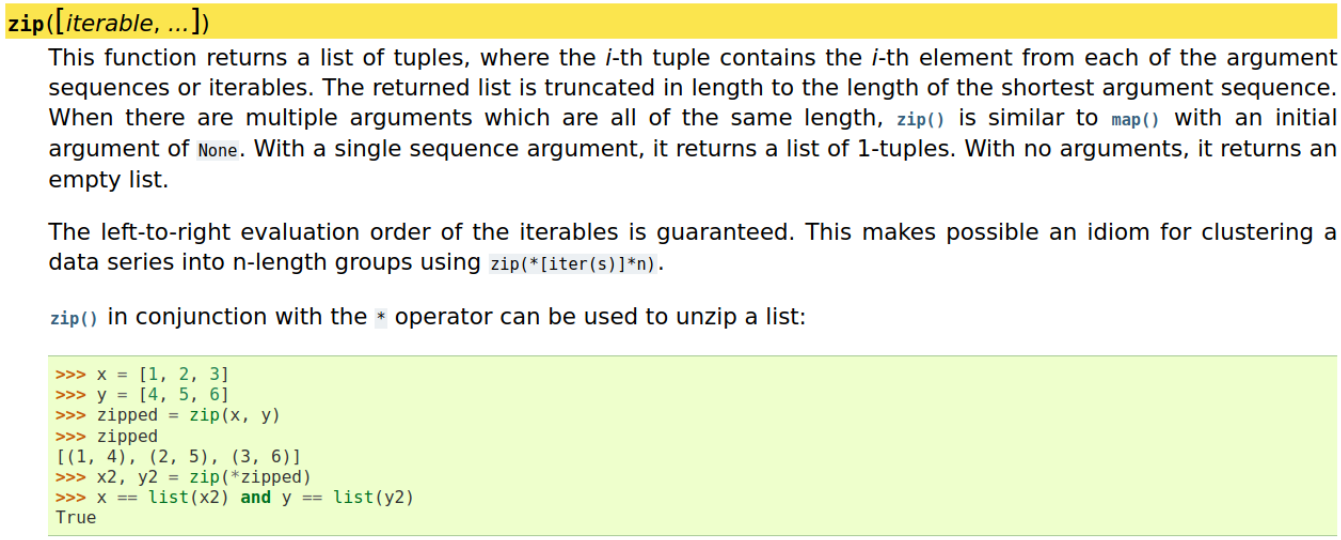
Question 9

Dans le code Python (`model.py`), expliquer comment est codée la configuration : un plateau de jeu, la position des robots (dessiner l’origine et les axes de la grille), l’objectif courant (couleur et forme). Donner le placement initial du robot rouge dans le jeu “hardest”. *Indication : regarder l’implémentation de la classe `Game` à la ligne 215, et vérifier à l’aide de la Figure 2. (10 lignes max)*

On donne la documentation de la fonction `zip` en Python 2 sur la figure 3 (https://python.readthedocs.io/en/v2.7.2/library/functions.html?highlight=zip#zip)

Question 10

Pour rendre la fonction `__init__` de la classe `Game` plus lisible par des élèves, donner un code équivalent à la ligne 229 : `self.robots = dict(zip(COLORS, robots))` sans utiliser la fonction zip.



zip([*iterable*, ...])

This function returns a list of tuples, where the *i*-th tuple contains the *i*-th element from each of the argument sequences or iterables. The returned list is truncated in length to the length of the shortest argument sequence. When there are multiple arguments which are all of the same length, `zip()` is similar to `map()` with an initial argument of `None`. With a single sequence argument, it returns a list of 1-tuples. With no arguments, it returns an empty list.

The left-to-right evaluation order of the iterables is guaranteed. This makes possible an idiom for clustering a data series into n-length groups using `zip(*[iter(s)]*n)`.

`zip()` in conjunction with the `*` operator can be used to unzip a list:

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> zipped = zip(x, y)
>>> zipped
[(1, 4), (2, 5), (3, 6)]
>>> x2, y2 = zip(*zipped)
>>> x == list(x2) and y == list(y2)
True
```

FIGURE 3 – Fonction zip en Python2

Question 11

Expliquer la fonctionnalité de la fonction `can_move` (ligne 254 de `model.py`) et écrire une spécification au format “docstring” qui pourrait aider vos élèves *5 lignes*.

Question 12

Expliquer comment est réalisé un mouvement (toujours dans `model.py`).

On suppose qu'on a instrumenté le code de résolution (que l'on verra dans la partie suivante), de façon à imprimer sur la sortie standard une solution. On comprend alors qu'une solution est décrite comme ceci :

$$[('R','E'),('R','N'),('R','W'),('R','N'),('Y','W'),('Y','N'),('Y','E')]$$

c'est à dire une liste (ordonnées) de couples : robot et direction

Question 13

On désire ajouter une fonction qui permet de vérifier une solution à partir de l'état **initial** des robots. Expliquer comment réaliser simplement une telle fonction dans le fichier `model.py`). *On fournira une analyse, implémentation, documentation. On explicitera clairement le type des variables utilisées.* On rappelle que l'information d'une case n de la grille se récupère avec `self.grid[n]`.

2.3 Étude d'une résolution simple en Python (model.py)

Le fichier principal `main.py` qui code l'interface utilisateur propose deux options pour la résolution automatique d'un problème :

1. `self.path = self.game.search()`
2. `self.path = ricochet.search(self.game, self.callback)`

On étudie ici la première solution basée sur l'usage des méthodes `search, _search` de la classe `model.py` (lignes 305, 313) dont le code est présent en annexe.

Question 14

On se pose la question de la signification de l'étoile dans l'appel à `do_move` ligne 328. Une rapide recherche sur stack overflow donne une réponse populaire (cf Figure 4). Remplacer la ligne 328 par du Python plus rapidement lisible par des élèves *Deux lignes.*

The single star `*` unpacks the sequence/collection into positional arguments, so you can do this:

```
def sum(a, b):
    return a + b

values = (1, 2)

s = sum(*values)
```

FIGURE 4 – What does the star operator mean, in a function call? (stack overflow)

Question 15

Transformer la fonction `do_move` ligne 275 en une fonction à compléter (2 trous = 2 TODO), avec une spécification claire (en docstring) et des commentaires `#TODO` adéquats.

Question 16

Que peut-on modifier si on n'a pas envie d'utiliser les exceptions dans la fonction `do_move` précédente? *2 lignes*

Question 17

Expliquer les fonctionnalités respectives des fonctions `search` et `_search` (305 et 313), sous forme de *docstring* Python.

Question 18

Dans la fonction `_search`, à quoi sert la structure de données `memo`, et de quel type est-elle? À quoi sert le traitement spécial pour `depth == maxdepth - 1`? Comment est construit le chemin courant (variable `path`)?

Question 19

Donner une version à trou (2 ou 3 "trous") de la fonction `_search` accessible à des élèves de Terminale. *On n'oubliera pas les commentaires qui expliquent le travail à faire*

Question 20

On instrumente la fonction d'énumération `search` afin de faire imprimer le temps d'une énumération des différentes recherches (de profondeur fixée). Sur le plateau difficile, les 18 premières profondeurs donnent les temps décrits dans le tableau de la figure 5 . Donner "à la louche" une estimation du temps de calcul d'une solution pour ce plateau, sachant que la profondeur est 25.*Expliquer rapidement le calcul.*

prof.	1 à 9	10	11	12	13	14	15	16	17	18
t (sec)	<0,1	0,96	2,1	4,4	8,9	18	35	66	124	244

FIGURE 5 – Temps de résolution pour les recherches "naïves récursives" à profondeur max fixée.

Question 21

La question précédente est un peu difficile pour des élèves de Terminale. Donner des indications

pour rendre cette question faisable en autonomie. Cette question pourra éventuellement faire appel à l'usage d'une calculatrice graphique ou d'un tableur.

Question 22

Un·e élève produit le fichier décrit dans le Listing 1.

Listing 1 – main_test.py

```
import model

def main():
    game = model.Game(None) # random game
    res = game.search()
    print(res)
    print(game.check_solution(res))
    print(game.check_solution(res))
```

pour tester sans l'interface graphique sa résolution (fonction `search`), et il/elle obtient :

```
[('R', 'N'), ('Y', 'N'), ('Y', 'E'), ('Y', 'N'),
('Y', 'W'), ('Y', 'N'), ('Y', 'W'), ('Y', 'S')]
True
False
```

Comment est-ce possible d'obtenir ces deux résultats différents ? Comment remédier à ce problème ? *Indication : quelles sont les structures de données modifiées lors d'un appel à `check_solution` ?*

2.4 Synthèse : production de la séquence "Projet" (1 heure au moins)

Il est ici demandé de passer du temps à réfléchir avant de rédiger proprement la partie. On rappelle qu'il s'agit d'un **projet de Terminale** NSI.

Question 23

Proposer une séance de découverte débranchée, dont l'objectif est de découvrir **l'ensemble des règles du jeu Ricochet** en vue du projet. Donner un timing approximatif et les compétences associées. *1/2 page à 1 page. On ne reproduira pas la fiche fournie en annexe, et on écrira des "vraies" compétences en terme d'algorithmique et de programmation.*

Question 24

(**Question de synthèse, à rédiger sérieusement**) Proposer un déroulement du projet long de NSI, avec des jalons précis et les compétences associées à ce projet. Vous expliquerez aussi comment vous gérez les difficultés de niveau et les différences d'avancée de projet. *On attend au moins une page de description. La question de l'évaluation du code est traitée dans la question suivante.*

Question 25

Proposer un barème pour l'évaluation **finale** du **code** dans le cadre du projet. *Expliquer les différents éléments que vous comptez évaluer ainsi que le nombre de points attribués pour chacun de ces éléments 10/15 lignes. Donner une compétence difficile à évaluer uniquement avec le produit final.*

Question 26

Proposer 3 questions de QCM à poser en devoir "sur table", capitalisant sur le projet, et permettant d'évaluer des compétences algorithmiques/langages apprises durant le projet. *Justifier proprement.*

Ricochet Robots

Annexes

Ricochet Robots (**Rasende Roboter** pour la première édition en allemand) est un jeu de société créé par Alex Randolph et illustré par Franz Vohwinkel, édité en 1999 par Hans im Glück / Tilsit.

Le jeu est composé d'un plateau, de tuiles représentant chacune une des cases du plateau, et de pions appelés « robots ». La partie est décomposée en tours de jeu, un tour consistant à déplacer les robots sur un plateau afin d'en amener un sur l'une des cases du plateau. Les robots se déplacent en ligne droite et avancent toujours jusqu'au premier mur qu'ils rencontrent.

On peut aussi bien y jouer seul qu'à un grand nombre de participants.

Ricochet Robots
jeu de société

Exemple de partie en cours

Auteur	Alex Randolph
Éditeur	Hans im Glück
Date de 1re édition	1999
Autre éditeur	Tilsit
Joueur(s)	1 à 99
Âge	à partir de 10 ans
Durée annoncée	30 minutes

habileté physique	réflexion décision	générateur de hasard	info. compl. et parfaite
✗ Non	✓ Oui	✗ Non	✓ Oui

Sommaire

Matériel

Première édition

En 1999, le jeu s'appelle Rasende Roboter et contient :

- 4 plateaux double-face à assembler ;
- 4 pions de couleurs différentes représentant les robots ;
- 4 tuiles *robot* de couleurs identiques à celles des robots ;
- 17 tuiles *objectif* distribuées en quatre groupes de quatre tuiles de couleur

identique à celle d'un robot, et une tuile multicolore ;
- 1 sablier.

Le plateau de jeu représente un quadrillage sur lequel figurent certaines cases spéciales, les cases *objectif*, c'est-à-dire les cases où doivent être amenés les robots. Ce plateau de jeu est composé de quatre parties recto-verso, permettant ainsi d'obtenir 96 plateaux de jeu différents. Une version en anglais, titrée *Ricochet Robot*[1], est éditée par Abacus / Rio Grande Games.

Deuxième édition

La deuxième édition sort en 2003 chez Abacus / Rio Grande Games, sous forme d'une boîte bleue titrée uniquement *Ricochet Robots*. Elle comprend un robot supplémentaire, de couleur noire. De plus certaines cases comportent des *murs* sur deux de leurs côtés, représentant les obstacles que rencontreront les robots[2].

Troisième édition

La troisième édition est une réédition de la boîte d'origine sous le nom de *Ricochet Robots*, avec un robot supplémentaire, de couleur argent. Les plateaux sont différents et compatibles avec l'édition précédente[3].

Objectif

L'objectif du jeu consiste à récupérer des tuiles *objectif* en amenant un des robots sur une case particulière du plateau.

Règles du jeu

À chaque tour, un des joueurs retourne une tuile *objectif*. Le but est alors d'amener le robot de la couleur de la tuile sur la case *objectif* dont le symbole est identique à celui de la tuile. Si c'est la tuile multicolore qui est tirée, l'objectif est alors d'amener n'importe quel robot sur la case multicolore du plateau.

Les joueurs jouent simultanément, chacun réfléchissant sur le moyen d'amener le robot en utilisant les règles de déplacement. Lorsque l'un d'entre eux pense avoir trouvé une solution, il annonce en combien de mouvements il compte réaliser l'objectif puis il retourne le sablier. Les autres joueurs ont jusqu'à la fin du sablier pour proposer de meilleures solutions, utilisant moins de mouvements.

Après l'écoulement du sablier, le joueur qui a la solution comptant le moins de mouvement montre sa solution et remporte la tuile. S'il échoue dans sa démonstration, le joueur qui proposait le nombre de mouvements immédiatement supérieur montre sa solution, etc. jusqu'à ce qu'une solution soit valide.

Règles de déplacement

Sur le plateau, les robots se déplacent en ligne droite et le plus loin possible avant de rencontrer un obstacle. Durant leur tour, les joueurs peuvent utiliser les quatre robots comme ils le souhaitent.

Une fois mis en mouvement, le robot ne peut s'arrêter ou repartir dans une autre direction que lorsqu'il rencontre un obstacle. Les obstacles peuvent être :

- les bords du plateau

- les *murs* symbolisés sur le plateau
- un autre robot

Chaque déplacement de robot compte pour un mouvement, quel que soit le nombre de cases parcourues.

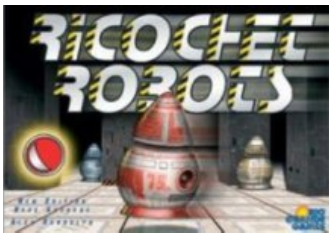
Cas particulier

Si, après avoir retourné une tuile *objectif*, il s'avère que la solution est atteignable en un seul mouvement, les joueurs devront ignorer cette solution et s'efforcer d'en trouver une autre.

Conditions de victoire

Le joueur qui possède le plus de tuiles *objectif* en fin de partie remporte la victoire.

ACADÉMIE
DE NANCY-METZ
Liberté
Égalité
Fraternité



Ricochet Robots

Article mis en ligne le 4 juillet 2019
dernière modification le 13 octobre 2019

par **Sébastien Lozano**

Ricochet Robots

Idée de départ : Variations autour du jeu **Ricochet Robots**.
But du jeu : Un robot doit atteindre une cible. Il peut se déplacer verticalement ou horizontalement mais ne s'arrête que lorsqu'il rencontre un obstacle : mur, autre robot... Il pourra alors changer de direction. Le joueur (ou l'équipe) qui utilise le moins de mouvements pour rejoindre la cible gagne.
Organisation : Travail de groupe pour comprendre et assimiler les règles. Défi en fin d'heure et prolongement par un projet de programmation algorithmique sur **Scratch**.

Protocole

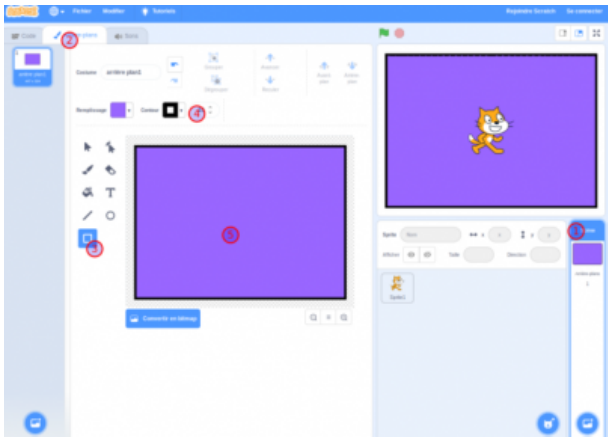
- Support pour le travail de groupe : Documents trouvés sur **http://primaths.fr**, en pièce jointe à la fin de l'article.

- ◦ les élèves sont en groupes par 4.
  ◦ chaque binôme reçoit un plateau et peut dessiner dessus. Les plateaux sont imprimables avec un lien à la fin de l'article.
  ◦ je mets un chronomètre de 1min30 en route
  ◦ à l'issue du temps imparti, je note le nombre de coups de chaque équipe, 1 déplacement vaut 1 coup, l'utilisation d'1 bloqueur vaut 1 coup. En cas d'égalité, je tire au sort l'équipe qui viendra exposer sa solution.
  ◦ l'équipe vient présenter sa solution au tableau sur lequel est vidéo-projeté le plateau en cours. Elle gagne 5 points si sa proposition est correcte et perd 1 point en cas d'erreur.

- Une fois les règles intégrées, je lance une petite compétition entre mes classes grâce au site **Prise 2 tête - Ricochet** sur lequel est proposé une variante numérique du jeu. Un nouveau défi est proposé chaque jour, les participants peuvent enregistrer leur score et consulter le classement le lendemain.

- Enfin, on peut lancer le projet de coder un pseudo-jeu sous **Scratch**. Les étapes suivantes peuvent permettre de décomposer la tâche :

Premier temps *environ 1h selon les élèves*
une scène simple, constituée par un rectangle noir qui permettra de contrôler l'arrêt des déplacements du robot.



scene simplifiée

Codage des déplacements grâce aux flèches du clavier dans les 4 directions, attention la gestion de l'orientation au moment du changement de direction est à anticiper.

Gestion de l'orientation

un bouton de réinitialisation pour ramener le robot au centre du rectangle par exemple.

Second temps *ajouter une cible et gérer l'affichage d'un message de victoire lors du contact avec celle-ci.*

Troisième temps *un compteur pour le nombre de coups*

Quatrième temps *qui peut être en travail facultatif à la maison, le design global de la scène et les débuggages éventuels*

Les notions algorithmiques/mathématiques travaillées

- Avancer d'un négatif c'est reculer : à partir de la 5eme mais l'idée peut être soumise dès la 6eme
- Répéter jusqu'à un test d'arrêt,
- Variable pour le compteur du nombre de mouvements, événements déclencheurs parallèles, boucle infinie, les effets de bords sous scratch avec les capteurs de couleur, ...

Si vous testez dans vos classes, vous pouvez, en cliquant sur le lien qui suit, faire un retour en écrivant à **sebastien.lozano@ac-nancy-metz.fr**

Avec la précieuse relecture d'**Audrey MICONI**

15 plateaux simplifiés

model.py 1/5
~/Documents/2021work/enseignement/CAPES-NSI/Ricochet/ 28/12/2020

```python
import itertools
import random

# Directions
NORTH = 'N'
EAST = 'E'
SOUTH = 'S'
WEST = 'W'

DIRECTIONS = [NORTH, EAST, SOUTH, WEST]

REVERSE = {
    NORTH: SOUTH,
    EAST: WEST,
    SOUTH: NORTH,
    WEST: EAST,
}

OFFSET = {
    NORTH: -16,
    EAST: 1,
    SOUTH: 16,
    WEST: -1,
}

# Masks
M_NORTH = 0x01
M_EAST  = 0x02
M_SOUTH = 0x04
M_WEST  = 0x08
M_ROBOT = 0x10

M_LOOKUP = {
    NORTH: M_NORTH,
    EAST: M_EAST,
    SOUTH: M_SOUTH,
    WEST: M_WEST,
}

# Colors
RED = 'R'
GREEN = 'G'
BLUE = 'B'
YELLOW = 'Y'

COLORS = [RED, GREEN, BLUE, YELLOW]

# Shapes
CIRCLE = 'C'
TRIANGLE = 'T'
SQUARE = 'Q'
HEXAGON = 'H'

SHAPES = [CIRCLE, TRIANGLE, SQUARE, HEXAGON]

# Tokens
TOKENS = [''.join(token) for token in itertools.product(COLORS, SHAPES)]

# Quadrants
QUAD_1A = (
    'NW,N,N,N,NE,NW,N,N,'
    'W,S,X,X,X,X,SEYH,W,'
    'WE,NWGT,X,X,X,X,N,X,'
    'W,X,X,X,X,X,X,X,'
    'W,X,X,X,X,X,S,X,'
    'SW,X,X,X,X,X,NEBQ,W,'
    'NW,X,E,SWRC,X,X,X,S,'
    'W,X,X,N,X,X,E,NW'
)

QUAD_1B = (
    'NW,NE,NW,N,NS,N,N,N,'
    'W,S,X,E,NWRC,X,X,X,'
    'W,NEGT,W,X,X,X,X,X,'
    'W,X,X,X,X,X,SEYH,W,'
    'W,X,X,X,X,X,N,X,'
    'SW,X,X,X,X,X,X,X,'
    'NW,X,E,SWBQ,X,X,X,S,'
```


model.py 2/5
~/Documents/2021work/enseignement/CAPES-NSI/Ricochet/ 28/12/2020

```python
    'W,X,X,N,X,X,E,NW'
)

QUAD_2A = (
    'NW,N,N,NE,NW,N,N,N,'
    'W,X,X,X,X,E,SWBC,X,'
    'W,S,X,X,X,X,N,X,'
    'W,NEYT,W,X,X,S,X,X,'
    'W,X,X,X,E,NWGQ,X,X,'
    'W,X,SERH,W,X,X,X,X,'
    'SW,X,N,X,X,X,X,S,'
    'NW,X,X,X,X,X,E,NW'
)

QUAD_2B = (
    'NW,N,N,N,NE,NW,N,N,'
    'W,X,SERH,W,X,X,X,X,'
    'W,X,N,X,X,X,X,X,'
    'WE,SWGQ,X,X,X,X,S,X,'
    'SW,N,X,X,X,E,NWYT,X,'
    'NW,X,X,X,X,S,X,X,'
    'W,X,X,X,X,NEBC,W,S,'
    'W,X,X,X,X,X,E,NW'
)

QUAD_3A = (
    'NW,N,N,NE,NW,N,N,N,'
    'W,X,X,X,X,SEGH,W,X,'
    'WE,SWRQ,X,X,X,N,X,X,'
    'SW,N,X,X,X,X,S,X,'
    'NW,X,X,X,X,E,NWYC,X,'
    'W,X,S,X,X,X,X,X,'
    'W,X,NEBT,W,X,X,X,S,'
    'W,X,X,X,X,X,E,NW'
)

QUAD_3B = (
    'NW,N,NS,N,NE,NW,N,N,'
    'W,E,NWYC,X,X,X,X,X,'
    'W,X,X,X,X,X,X,X,'
    'W,X,X,X,X,E,SWBT,X,'
    'SW,X,X,X,S,X,N,X,'
    'NW,X,X,X,NERQ,W,X,X,'
    'W,SEGH,W,X,X,X,X,S,'
    'W,N,X,X,X,X,E,NW'
)

QUAD_4A = (
    'NW,N,N,NE,NW,N,N,N,'
    'W,X,X,X,X,X,X,X,'
    'W,X,X,X,X,SEBH,W,X,'
    'W,X,S,X,X,N,X,X,'
    'SW,X,NEGC,W,X,X,X,X,'
    'NW,S,X,X,X,X,E,SWRT,'
    'WE,NWYQ,X,X,X,X,X,NS,'
    'W,X,X,X,X,X,E,NW'
)

QUAD_4B = (
    'NW,N,N,NE,NW,N,N,N,'
    'WE,SWRT,X,X,X,X,S,X,'
    'W,N,X,X,X,X,NEGC,W,'
    'W,X,X,X,X,X,X,X,'
    'W,X,SEBH,W,X,X,X,S,'
    'SW,X,N,X,X,X,E,NWYQ,'
    'NW,X,X,X,X,X,X,S,'
    'W,X,X,X,X,X,E,NW'
)

QUADS = [
    (QUAD_1A, QUAD_1B),
    (QUAD_2A, QUAD_2B),
    (QUAD_3A, QUAD_3B),
    (QUAD_4A, QUAD_4B),
]

# Rotation
ROTATE_QUAD = [
```

model.py **3/5**
~/Documents/2021work/enseignement/CAPES–NSI/Ricochet/ 28/12/2020

```
    56, 48, 40, 32, 24, 16,  8,  0,
    57, 49, 41, 33, 25, 17,  9,  1,
    58, 50, 42, 34, 26, 18, 10,  2,
    59, 51, 43, 35, 27, 19, 11,  3,
    60, 52, 44, 36, 28, 20, 12,  4,
    61, 53, 45, 37, 29, 21, 13,  5,
    62, 54, 46, 38, 30, 22, 14,  6,
    63, 55, 47, 39, 31, 23, 15,  7,
]

ROTATE_WALL = {
    NORTH: EAST,
    EAST: SOUTH,
    SOUTH: WEST,
    WEST: NORTH,
}

# Helper Functions
def idx(x, y, size=16):
    return y * size + x

def xy(index, size=16):
    x = index % size
    y = index / size
    return (x, y)

def rotate_quad(data, times=1):
    for i in range(times):
        result = [data[index] for index in ROTATE_QUAD]
        result = [''.join(ROTATE_WALL.get(c, c) for c in x) for x in result]
        data = result
    return data

def create_grid(quads=None):
    if quads is None:
        quads = [random.choice(pair) for pair in QUADS]
        random.shuffle(quads)
    quads = [quad.split(',') for quad in quads]
    quads = [rotate_quad(quads[i], i) for i in [0, 1, 3, 2]]
    result = [None for i in range(16 * 16)]
    for i, quad in enumerate(quads):
        dx, dy = xy(i, 2)
        for j, data in enumerate(quad):
            x, y = xy(j, 8)
            x += dx * 8
            y += dy * 8
            index = idx(x, y)
            result[index] = data
    return result

def to_mask(cell):
    result = 0
    for letter, mask in M_LOOKUP.items():
        if letter in cell:
            result |= mask
    return result

# Game
class Game(object):
    @staticmethod
    def hardest():
        quads = [QUAD_2B, QUAD_4B, QUAD_3B, QUAD_1B]
        robots = [226, 48, 43, 18]
        token = 'BT'
        return Game(quads=quads, robots=robots, token=token)
    def __init__(self, seed=None, quads=None, robots=None, token=None):
        if seed:
            random.seed(seed)
        self.grid = create_grid(quads)
        if robots is None:
            self.robots = self.place_robots()
        else:
            self.robots = dict(zip(COLORS, robots))
        self.token = token or random.choice(TOKENS)
        self.moves = 0
        self.last = None
    def place_robots(self):
        result = {}
```

model.py **4/5**
~/Documents/2021work/enseignement/CAPES–NSI/Ricochet/ 28/12/2020

```
235         used = set()
236         for color in COLORS:
237             while True:
238                 index = random.randint(0, 255)
239                 if index in (119, 120, 135, 136):
240                     continue
241                 if self.grid[index][-2:] in TOKENS:
242                     continue
243                 if index in used:
244                     continue
245                 result[color] = index
246                 used.add(index)
247                 break
248         return result
249     def get_robot(self, index):
250         for color, position in self.robots.iteritems():
251             if position == index:
252                 return color
253         return None
254     def can_move(self, color, direction):
255         if self.last == (color, REVERSE[direction]):
256             return False
257         index = self.robots[color]
258         if direction in self.grid[index]:
259             return False
260         new_index = index + OFFSET[direction]
261         if new_index in self.robots.itervalues():
262             return False
263         return True
264     def compute_move(self, color, direction):
265         index = self.robots[color]
266         robots = self.robots.values()
267         while True:
268             if direction in self.grid[index]:
269                 break
270             new_index = index + OFFSET[direction]
271             if new_index in robots:
272                 break
273             index = new_index
274         return index
275     def do_move(self, color, direction):
276         start = self.robots[color]
277         last = self.last
278         if last == (color, REVERSE[direction]):
279             raise Exception
280         end = self.compute_move(color, direction)
281         if start == end:
282             raise Exception
283         self.moves += 1
284         self.robots[color] = end
285         self.last = (color, direction)
286         return (color, start, last)
287     def undo_move(self, data):
288         color, start, last = data
289         self.moves -= 1
290         self.robots[color] = start
291         self.last = last
292     def get_moves(self, colors=None):
293         result = []
294         colors = colors or COLORS
295         for color in colors:
296             for direction in DIRECTIONS:
297                 if self.can_move(color, direction):
298                     result.append((color, direction))
299         return result
300     def over(self):
301         color = self.token[0]
302         return self.token in self.grid[self.robots[color]]
303     def key(self):
304         return tuple(self.robots.itervalues())
305     def search(self):
306         max_depth = 1
307         while True:
308             #print 'Searching to depth:', max_depth
309             result = self._search([], set(), 0, max_depth)
310             if result is not None:
311                 return result
312             max_depth += 1
```

model.py **5/5**
~/Documents/2021work/enseignement/CAPES–NSI/Ricochet/ 28/12/2020

```
313     def _search(self, path, memo, depth, max_depth):
314         if self.over():
315             return list(path)
316         if depth == max_depth:
317             return None
318         key = (depth, self.key())
319         if key in memo:
320             return None
321         memo.add(key)
322         if depth == max_depth - 1:
323             colors = [self.token[0]]
324         else:
325             colors = None
326         moves = self.get_moves(colors)
327         for move in moves:
328             data = self.do_move(*move)
329             path.append(move)
330             result = self._search(path, memo, depth + 1, max_depth)
331             path.pop(-1)
332             self.undo_move(data)
333             if result:
334                 return result
335         return None
336     def export(self):
337         grid = []
338         token = None
339         robots = [self.robots[color] for color in COLORS]
340         for index, cell in enumerate(self.grid):
341             mask = to_mask(cell)
342             if index in robots:
343                 mask |= M_ROBOT
344             grid.append(mask)
345             if self.token in cell:
346                 token = index
347         robot = COLORS.index(self.token[0])
348         return {
349             'grid': grid,
350             'robot': robot,
351             'token': token,
352             'robots': robots,
353         }
end
```