Jeu de Bataille Navale [sr07] - Examen

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Université Haute-Alsace

Version 20 mai 2018

Table des matières

Python - Jeu de Bataille Navale (Solution)

Mots-Clés Tableau bidimensionnel, Jeu ■
Requis Structures de base, Structures conditionnelles, Algorithmes paramétrés, Structures répétitives, Schéma itératif ■
Difficulté ● ● ○

Objectif
Cet exercice réalise un jeu de Bataille Navale contre l'ordinateur : l'ordinateur met des bateaux au hasard dans le terrain de jeu et l'utilisateur cherche à les couler en tirant dans les cases.

1 Jeu de Bataille Navale / pgbnavale (13 points)

1.1 Terrain de jeu (1 point)

Terrain de jeu
Dans ce jeu, le terrain de jeu est représenté par une grille. Une case du terrain peut contenir quatre valeurs différentes : initialement elle contient la mer ou un bateau (ici les bateaux ne feront qu'une case de long). Quand on tire sur la mer, la case contient alors un tir ; et quand on tire sur un bateau, la case contient alors un débris.

(0.25 point) Les différentes valeurs des cases seront représentés par des entiers. Définissez les constantes `BMer=1`, `BBateau=2`, `BDebris=3` et `BTir=4` représentant les valeurs que peuvent contenir les cases du terrain.

(0.75 point) Définissez les constantes `LGMAX=12` et `HTMAX=14` représentant les dimensions maximales d'un terrain, puis définissez le type `Terrain`, structure qui inclut :

- Un entier `hauteur` mémorisant la hauteur effective du terrain.
- Un entier `largeur` de la largeur effective du terrain.
- Un tableau bidimensionnel d'entiers `cases` de taille `HTMAX` lignes et `LGMAX` colonnes, le premier indice étant le numéro de ligne et le deuxième le numéro de colonne.

Validez vos définitions avec la solution.

Solution Python @[pgbnavale.py]

```python
# Mer
BMer = 0
# Bateau
BBateau = 1
# Débris
BDebris = 2
# Tir
BTir = 3
# Largeur maximale
LGMAX = 12
# Hauteur maximale
HTMAX = 14

class Terrain:
    """ Terrain de jeu """
    def __init__(self):
        self.hauteur = 0
        self.largeur = 0
        self.cases = [[0 for x in range(LGMAX)] for x in range(HTMAX)]
```

Écrivez une fonction `evalCase(tr,j,k)` qui renvoie la valeur de la case en `(j,k)` d'un `Terrain tr`.

Et écrivez une procédure `fixerCase(tr,j,k,valeur)` qui modifie la valeur de la case en `(j,k)` d'un `Terrain tr` en la fixant à `valeur`.

Validez vos opérations avec la solution.

Solution Python @[pgbnavale.py]

```
def evalCase(tr, j, k):
    """  Valeur d'une Case sur un Terrain

     :param tr: un Terrain
     :param j: indice de ligne
     :param k: indice de colonne
     :return: la valeur en tr[j][k]
    """
    return tr.cases[j][k]
```

```
def fixerCase(tr, j, k, valeur):
    """  Fixe une Case sur un Terrain

     :param tr: un Terrain
     :param j: indice de ligne
     :param k: indice de colonne
     :param valeur: valeur à fixer
    """
    tr.cases[j][k] = valeur
```

1.2 Affichage d'un terrain (1.5 points)

(0.5 point) Écrivez une fonction `symboleDe(x)` qui renvoie le caractère d'un entier `x` :

- `'~'` pour `BMer`
- `'~'` pour `BBateau` (l'utilisateur ne doit pas voir les bateaux)
- `'o'` pour `BTir`
- `'x'` pour `BDebris`

(1 point) Déduisez une procédure `afficherTerrain(tr)` qui affiche un `Terrain tr` comme dans l'exemple suivant :

```
~~~~~~~
~o~~x~~
~~x~~~~
~~oo~~~
```

Validez votre fonction et votre procédure avec la solution.

Solution Python @[pgbnavale.py]

```
def symboleDe(x):
    """  Caractère visuel d'une valeur de case

     :param x: valeur
     :return: le caractère de x correspondant
    """
    if x == BMer or x == BBateau:
        return "~"
    elif x == BTir:
        return "o"
    else:
        return "x"
```

```
def afficherTerrain(tr):
    """  Affichage d'un Terrain

     :param tr: un Terrain
    """
    for j in range(0, tr.hauteur):
        for k in range(0, tr.largeur):
            print(symboleDe(evalCase(tr, j, k)), end="")
        print()
```

1.3 Initialisation d'un terrain (4.5 points)

(1 point) Écrivez une procédure `demanderTaille(tr)` qui saisit les dimensions (largeur et hauteur) d'un `Terrain tr` en vérifiant qu'elles sont correctes, à savoir :

- La largeur dans `[1..LGMAX]`.
- La hauteur dans `[1..HTMAX]`.

(1.5 point) Au départ le terrain est vide, c.-à-d. qu'il ne contient que de la mer `BMer`.

Écrivez une procédure `initialiserTerrain(tr)` qui initialise les cases d'un `Terrain tr` à la mer (la taille du terrain est supposée avoir été préalablement demandée).

Validez vos procédures avec la solution.

Solution Python @[pgbnavale.py]

```
def saisirEntier(binf, bsup):
    """  Saisie controlée d'un entier dans un intervalle

     :param binf: borne inférieure
     :param bsup: borne supérieure
     :return: un entier dans [binf..bsup] (avec binf <= bsup)
    """
    n = binf - 1
    while not binf <= n <= bsup:
        print("Entier dans [", binf, "..", bsup, "]? ", sep="", end="")
```

```
        n = int(input())
    return n
```

```
def demanderTaille(tr):
    """  Saisie des dimensions d'un Terrain

     :param tr: un Terrain
    """
    print("Largeur du terrain: ", end="")
    tr.largeur = saisirEntier(1, LGMAX)
    print("Hauteur du terrain: ", end="")
    tr.hauteur = saisirEntier(1, HTMAX)
```

```
def initialiserTerrain(tr):
    """  Initialisation d'un Terrain à BMer

     :param tr: un Terrain
    """
    # Les dimensions sont valides
    for j in range(0, tr.hauteur):
        for k in range(0, tr.largeur):
            fixerCase(tr, j, k, BMer)
```

(2 points) Il faut ensuite mettre des bateaux au hasard, sachant qu'ici les bateaux ne font qu'une case.

Écrivez une procédure `mettreBateaux(tr,nbt)` qui met au hasard `nbt` bateaux dans un `Terrain tr`. Attention, il faut avoir `nbt` bateaux dans `nbt` cases différentes (on pourrait tirer au hasard plusieurs fois la même case). Pour cela, faites un tirage sécurisé :

```
Pour chacun des nbt bateaux Faire
   Tirer les coordonnées d'une case au hasard jusqu'à trouver une case libre
   Mettre un bateau dans la case libre
FinPour
```

Outil Python

Le package `random` définit la fonction `randint(a,b)` qui renvoie un entier pseudo-aléatoire compris dans l'intervalle `[a..b]`.

Validez votre procédure avec la solution.

Solution Python @[pgbnavale.py]

```
def mettreBateaux(tr, nbt):
    """  Mise des bateaux sur un Terrain

     :param tr: un Terrain
     :param nbt: nombre de bateaux à mettre
    """
    for n in range(nbt):
        while True:
            j, k = rnd.randint(0,tr.hauteur-1), rnd.randint(0,tr.largeur-1)
```

```
            if evalCase(tr, j, k) == BMer: break
        fixerCase(tr, j, k, BBateau)
```

1.4 Effectuer un tir (3.5 points)

(0.5 point) Définissez le type `Case`, structure qui inclut deux entiers : `lig` et `col` représentant la position d'une case.

Validez votre définition avec la solution.

Solution Python @[pgbnavale.py]

```
class Case:
    """ Position d'une case """
    def __init__(self):
        self.lig = 0
        self.col = 0
```

(1 point) Écrivez une fonction `caseValide(tr,c)` qui teste et renvoie `Vrai` si les coordonnées d'une `Case c` sur un `Terrain tr` sont valides, `Faux` sinon. La fonction doit tester que la `Case` est sur le `Terrain`, et dans l'affirmative, qu'elle n'a pas déjà été tirée : ceci signifie que sa valeur doit être `BMer` ou `BBateau`.

Validez votre fonction avec la solution.

Solution Python @[pgbnavale.py]

```
def dansIntervalle(n, a, b):
    """  Prédicat d'un entier dans un intervalle

     :param n: un entier
     :param a: un entier
     :param b: un entier
     :return: Vrai si n est dans [a..b] (a <= b), Faux sinon
    """
    return (a <= n <= b)
```

```
def caseValide(tr, c):
    """  Prédicat de case valide

     :param tr: un Terrain
     :param c: une Case
     :return: Vrai si c est valide sur tr
    """
    # la Case doit être sur le Terrain
    b = (dansIntervalle(c.lig, 0, tr.hauteur - 1) and dansIntervalle(c.col, 0,
    tr.largeur - 1))
    # Si oui, evalCase est licite
    if b:
```

```
        valeur = evalCase(tr, c.lig, c.col)
        # Et la valeur doit être BMer ou BBateau
        b = (valeur == BMer or valeur == BBateau)
    return b
```

(1 point) Déduisez une procédure `demanderCaseTir(tr,c)` qui saisit les coordonnées de la `Case` de tir dans `c` sur un `Terrain tr`.

Validez votre procédure avec la solution.

Solution Python @[pgbnavale.py]

```
def demanderCaseTir(tr, c):
    """  Saisie des coordonnées d'un tir

     :param tr: un Terrain
     :param c: une Case
    """
    c.lig, c.col = -1, -1
    while not caseValide(tr, c):
        c.lig = int(input("Ligne du tir? "))
        c.col = int(input("Colonne du tir? "))
```

(1 point) Écrivez alors une fonction `tirer(tr,c)` qui effectue le tir dans la case de coordonnées `Case c` sur un `Terrain tr` en affichant les résultats du tir (`bateau coule` ou `dans l'eau`). La fonction renverra 0 si le tir est dans l'eau, et 1 pour un bateau coulés.

Validez votre fonction avec la solution.

Solution Python @[pgbnavale.py]

1.5 Jeu (2.5 points)

(2.5 points) Finalement, écrivez une procédure `jouer` qui organise un jeu de bataille navale. Il faut :

- Demander les dimensions du terrain.
- Initialiser le terrain.
- Mettre les bateaux : Le nombre de bateaux sera un cinquième du nombre de cases.
- Affichez le terrain.
- Jouer `TantQue` tous les bateaux ne sont pas coulés :
  - Demander à l'utilisateur sa case de tir.
  - Faire le tir et afficher le résultat du tir.

— Ré-afficher le terrain.

- Affichez le pourcentage de réussite des tirs qui vaut : `100*nbt/ntirs` où `nbt` est le nombre de bateaux et `ntirs` celui du nombre de tirs effectués par l'utilisateur.

Validez votre procédure avec la solution.

Solution Python @[pgbnavale.py]

```
def jouer():
    """ Réalise un jeu """
    tr = Terrain()
    demanderTaille(tr)
    initialiserTerrain(tr)
    nbateaux = (tr.hauteur * tr.largeur) // 5
    mettreBateaux(tr, nbateaux)
    afficherTerrain(tr)

    print("Nombre de bateaux a trouver: ", nbateaux, sep="")
    ntirs = 0
    ncoules = 0
    while ncoules < nbateaux:
        c = Case()
        demanderCaseTir(tr, c)
        ncoules += tirer(tr, c)
        afficherTerrain(tr)
        ntirs += 1
    pourcent = (nbateaux * 100.0) / ntirs
    print("Pourcentage de reussite = ", pourcent, sep="")
```

Testez.

2 Références générales

Comprend [] ■