UNIVERSITE LIBRE DE BRUXELLES
École Polytechnique de Bruxelles

février 2014

INFO-H-100 : Introduction à la programmation
Projet 1 : Boggle – **Quelques erreurs récurrentes**

Lors du premier projet nous avons relevé certaines erreurs récurrentes. Pour vous permettre de ne plus les commettre à l'avenir, nous avons reproduit ci-dessous quelques extraits de code qui contiennent un ou plusieurs défauts. Nous vous proposons, en regard, des commentaires et une version "corrigée".

Exemple 1

```
def word_on_board(word, board):
    k=0
    for i in range(len(board)):
        for j in range(len(board[0])):
            if (word_in_place(word,board,i,j)):
                k+=1
    return k>0
```

Commentaires

- Il suffit que la sous-fonction `word_in_place` renvoie `True` une fois pour que la fonction `word_on_board` doive retourner `True`. Il est dès lors inutile de continuer à parcourir la grille après qu'un appel de `word_in_place` ait renvoyé `True`. Dans l'exemple ci-dessus, il aurait donc été plus judicieux d'utiliser deux boucles `while` imbriquées, au lieu de boucles `for`.
- La variable `k` est un entier, mais est utilisé comme un booléen. Veillez à utiliser des booléens plutôt que des entiers pour vérifier une condition.

Notre proposition

```
def word_on_board(word, board):
    found = False
    i = 0
    while i<len(board) and not found:
        j = 0
        while j<len(board) and not found:
            found = word_in_place(word,board,i,j)
    return found
```

Exemple 2

```
def play_boggle():
    dico=Bg.load_dico("dico.txt")
    tour,score=5,0
    while tour>0:
        score+=play_one_round(dico)
        tour-=1
    print("Votre score final est de ",score,plural("point",score))
    if (Boggle_mod.ask_yes_no("Voulez-vous jouer une partie de Boggle ? :)")):
        play_boggle()
```

Commentaires

- Une partie est composée d'exactement cinq tours. Ce nombre étant connu et constant, il est plus adéquat d'utiliser une boucle `for`.
- En fin de partie, le programme demande au joueur s'il veut jouer une autre partie. En cas de réponse positive, il y a un appel, au sein de la fonction `play_boggle`, à elle-même. Cette utilisation de la *récursivité*, même si elle fonctionne dans ce cas, n'est pas conseillée. Il aurait mieux valu utiliser une boucle `while` qui avait le même effet.

Notre proposition

```
def play_boggle():
    dico = Boggle_mod.load_dico("dico.txt")
    play = True
    while play:
```

```
        score = 0
        for tour in range(5):
            score += play_one_round(dico)
        print("Votre score final est de ",score,plural("point",score))
        play = Boggle_mod.ask_yes_no("Voulez-vous jouer une partie de Boggle ?:")
```

Exemple 3

```
def dictionary(word):
    """
    description: regarde si le mot est bien dans le dictionnaire
    input: word (string)
    output: x (bool)
    """
    file_name = "dico.txt"
    dico = Boggle_mod.load_dico(file_name)
    if word.upper() in dico:
        x = True
    else:
        x = False
    return x
```

Commentaires

– La lecture répétée du dictionnaire à partir du fichier n'est pas très efficace. Mieux vaut le charger initialement et le passer en paramètre aux fonctions qui en ont besoin.
– Le nom de variable `x` n'est pas très parlant. On n'en comprend pas la signification.
– La condition `if` est inutile. La variable `x` reçevant la valeur booléenne qui doit être renvoyée par la fonction, il aurraît suffit de retourner la valeur de l'expression booléenne du `if`.

Notre proposition

```
def in_dico(dico, word):
    """
    description: regarde si le mot est bien dans le dictionnaire
    input: dico (dictionnaire), word (string)
    output: (bool) True si le mot est trouvé dans dico.
    """
    return word.upper() in dico
```

Exemples 4 & 5

Les extraits suivants sont des exemples de code compliqué et difficilement lisible (même s'il s'exécute correctement et fournit les résultats attendus). Nous ne proposons pas de correction pour ces exemples.

```
def next_is_adjacent(letter, board_nia, line_nia, col_nia, memory_nia):
    verify_nia=False
    # Checks the space at the bottom.
    if line_nia!=3 and letter==board_nia[line_nia+1][col_nia] and not(line_nia+1,col_nia) in memory_nia:
        verify_nia, line_nia = True, line_nia+1
    elif line_nia!=0 and letter==board_nia[line_nia-1][col_nia] and not (line_nia-1,col_nia) in memory_nia:
        verify_nia, line_nia = True, line_nia-1
    elif col_nia!=3 and letter==board_nia[line_nia][col_nia+1] and not (line_nia,col_nia+1) in memory_nia:
    # Checks the space at the right.
        verify_nia, col_nia = True, col_nia+1
    elif col_nia!=0 and letter==board_nia[line_nia][col_nia-1] and not  (line_nia,col_nia-1) in memory_nia:
    # Checks the space at the left.
        verify_nia, col_nia = True, col_nia-1
    return(verify_nia, line_nia, col_nia)
```

et

```
def play_one_round(dico):
    board=shake()
    Boggle_mod.show_grid(board)

    word=writting_word()
    x=word_on_board(word, board)
    y=False
    z=True
    i=0
    if x==True:
        line_or_col=search_first_letter_on_word(word,board)
        print(line_or_col)
        while i<len(line_or_col) and z==True:
            z=word_in_place(word,board,line_or_col[0][i],line_or_col[1][i])
            i+=1
        if z==True:
            y=word_in_dico(word)
```

```
    s=0
    if  y==True:
        s=score(word)
    elif x==False:
        s=0
        print("Ce mot n'est pas dans le tableau")
    return s
```

Commentaires

– La lecture et la compréhension du premier exemple sont rendus très compliquées par le fait que, pour respecter la limite des 10 lignes :

1. l'expression booléenne teste trois conditions.
2. les assignations sont faites sous forme de tuples.

– Pour le deuxième exemple, nous avons surtout voulu mettre en évidence la difficulté de lire un code dont les noms choisis pour les variables sont abscons. En particulier, il est difficile de comprendre la signification "logique" des variables `x`, `y` et `z`.