SUPINFO Academic Dept.

SUPINFO International University

INSTITUTE OF INFORMATION TECHNOLOGY

Algorithmique et programmation en Python

Second Mini - Projet

Just Get 10

Version 1.0
Last update: 14/12/2016
Use: Students/Staff
Author: Laurent GODEFROY

SOMMAIRE

1 PREAMBULE

Cet examen est à réaliser par groupes de deux étudiants. Dans l'unique cas où le nombre d'étudiants de la promotion est impair, un et un seul groupe de trois est autorisé.

Toute forme de plagiat ou utilisation de codes disponibles sur internet ou tout autre support, même de manière partielle, est strictement interdite et se verra sanctionnée d'un 0, d'une mention « cheater », et le cas échéant d'un conseil de discipline.

Ce mini-projet donnera lieu à des soutenances. Vos horaires de passages vous seront communiqués par votre campus.

Les soutenances sont également par groupes de deux. Elles dureront **20 minutes** pendant lesquelles vous montrerez à votre examinateur le bon fonctionnement de votre programme en en faisant la démonstration. Si vous n'avez pas implémenté le projet dans sa totalité, vous exposerez les parties fonctionnelles.

Pour appuyer votre présentation, vous devrez préparer un fichier de type Powerpoint, dans lesquels vous expliquerez les points du code que vous jugez les plus importants et significatifs. Il n'est pas nécessaire d'envoyer ce fichier à votre examinateur, ce dernier le découvrira le jour de la soutenance. Une communication précisant tout cela vous sera envoyé début janvier.

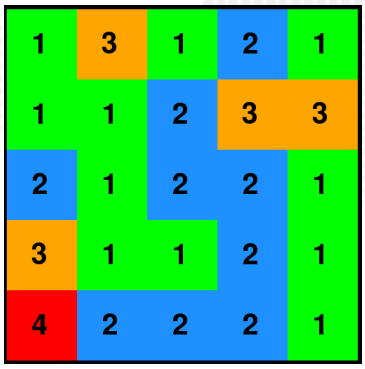
Un barème **indicatif** vous est donné dans la dernière partie de ce sujet.

2 GENERALITES SUR CE PROJET

Remarque importante : aucun code n'est demandé dans cette partie qui n'est qu'explicative.

Le but de ce mini-projet est d'écrire en langage Python un programme permettant de jouer au célèbre jeu de réflexion et de logique « Just Get 10 ». Ce jeu a été conçu par la companie Veewo et vous trouverez sa version originale ici.

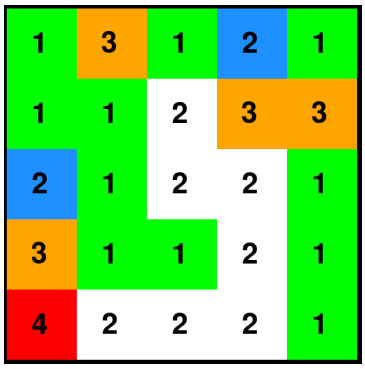
Initialement, des nombres de 1 à 4 sont aléatoirement disposés dans une grille de 5 lignes et 5 colonnes. Le nombre 1 a une probabilité plus grande d'apparition que le nombre 2, qui lui-même en a une plus grande que le nombre 3, etc. Voici un exemple de configuration initiale :



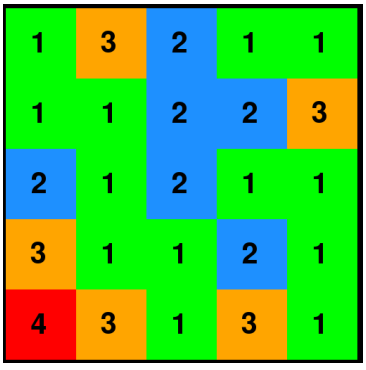
Le but est d'obtenir la valeur 10 en fusionnant successivement des cellules adjacentes de même valeur. Lorsque l'on fusionne un tel ensemble de cellules on obtient une cellule dont la valeur est incrémentée de 1. Les autres cellules de l'ensemble disparaissent, les cellules non impactées "tombent" par gravité, et les colonnes non pleines sont remplies aléatoirement avec les mêmes probabilités que lors de la création de la grille.

A noter que le terme "adjacent" n'inclut pas les cellules se touchant juste par le sommet, en diagonale.

On va expliciter cela sur un exemple. On peut donc sélectionner (elles vont apparaître en blanc) un ensemble de cellules adjacentes de même valeur en cliquant sur n'importe laquelle de ces cellules :



On choisit ensuite la cellule vers laquelle elles vont fusionner en cliquant dessus, par exemple ici celle située sur la cinquième ligne et deuxième colonne :



Les cellules sélectionnées avaient une valeur de 2, donc un 3 est apparu sur la cinquième ligne et deuxième colonne. Tous les autres 2 de l'ensemble sélectionné ont alors disparu. Les cellules restantes des colonnes 3 et 4 sont ensuite tombées par gravité, et ces colonnes incomplètes ont été remplies aléatoirement.

De proche en proche le but est donc d'atteindre 10. Si l'on ne peut plus jouer avant cela, la partie est perdue.

Vous aurez peut-être besoin de plusieurs lectures du sujet pour avoir une bonne vue d'ensemble du projet. Prenez donc le temps nécessaire à une bonne compréhension avant de commencer les codes demandés dans les parties suivantes.

Dans la partie 3 on implémentera les algorithmes nécessaires au déroulement du jeu.

Dans la partie 4 on rajoutera une interface graphique.

Enfin, dans la partie 5, on proposera quelques bonus et extensions possibles.

3 DES ALGORITHMES POUR JOUER A CE JEU

Il vous est fortement recommandé de lire l'intégralité de cette partie avant de commencer à coder. Les travaux demandés sont mis en évidence avec une couleur bleue.

Remarque importante : dans cette partie on ne travaillera qu'en mode console. La grille sera naturellement une liste à deux dimensions d'entiers.

3.1 INITIALISATION

Dans un fichier que l'on nommera « bases.py », implémenter les sous-programmes suivants :

- Une fonction prenant en paramètre un t-uple de trois réels (x_1,x_2,x_3) vérifiant $0<x_1<x_2<x_3<1$. Un nombre sera tout d'abord tiré aléatoirement entre 0 et 1 grâce à la fonction random. Si ce nombre est plus petit que x_1 la fonction retournera 4, s'il est entre x_1 et x_2, la fonction retournera 3, s'il est entre x_2 et x_3, la fonction retournera 2 et sinon elle retournera 1.

- Une fonction prenant en paramètre un entier **n** et un t-uple de trois réels **(x_1,x_2,x_3)** vérifiant **$0<x_1<x_2<x_3<1$**. Elle retournera une liste à deux dimensions de **n** lignes et **n** colonnes dont toutes les valeurs sont des entiers entre 1 et 4 obtenus grâce à la fonction précédente.
- Une procédure réalisant l'affichage des valeurs d'une liste à deux dimensions passée en paramètres. Cette procédure ne servira qu'à vérifier le bon fonctionnement des sous-programmes de cette partie 3.

Exemple de fonctionnement : le code suivant

```
n = 5
proba=(0.05,0.30,0.6)
gameBoard = newBoard(n,proba)
display(gameBoard,n)
```

doit conduire à un résultat similaire à celui-ci :

```
1 1 1 3 1
2 2 4 1 2
2 2 1 1 1
1 2 3 2 1
1 1 3 2 1
```

3.2 COUPS POSSIBLES

Dans un fichier que l'on nommera « possibles.py », implémenter les sous-programmes suivants :

- Une fonction prenant en paramètre un entier **n**, une liste à deux dimensions de **n** lignes et **n** colonnes, et deux entiers **i** et **j** tels que **$0\leq i<n$** et **$0\leq j<n$**. Elle retournera **True** si la cellule de coordonnées **i** et **j** possède une case adjacente de même valeur et **False** sinon.
- Une fonction prenant en paramètre un entier **n** et une liste à deux dimensions de **n** lignes et **n** colonnes. Elle retournera **True** si un coup est encore jouable sur la grille, c'est-à-dire si au moins une des cellules possède une case adjacente de même valeur. Elle retournera **False** sinon.
- Une fonction prenant en paramètre un entier **n** et une liste à deux dimensions de **n** lignes et **n** colonnes. Elle retournera la valeur maximale de la liste.

3.3 FUSION DE CELLULES

Dans un fichier que l'on nommera « merge.py », implémenter les sous-programmes suivants :

- Une procédure prenant en paramètre un entier **n**, une liste à deux dimensions de **n** lignes et **n** colonnes, un t-uple de deux entiers **(i,j)** tels que **0≤i<n** et **0≤j<n**, et une liste de t-uples de deux entiers vérifiant les inégalités précédentes. On supposera que lors d'un appel de cette procédure effectué depuis un autre sous-programme, la liste passée en paramètre ne contient initialement que le t-uple passé lui aussi en paramètre. A la fin de l'exécution de la procédure, cette liste contiendra les coordonnées (sous forme de t-uples) de l'ensemble des cellules adjacentes de même valeur que la cellule dont les coordonnées sont celles du t-uple passé en paramètre.
- Une procédure prenant en paramètre un entier **n**, une liste à deux dimensions de **n** lignes et **n** colonnes, et une liste de t-uples de deux entiers **(i,j)** tels que **0≤i<n** et **0≤j<n**. Elle incrémentera de 1 la valeur de la cellule dont les coordonnées sont le premier élément de la liste, et elle passera à 0 les valeurs des cellules dont les coordonnées sont les autres éléments de la liste.
- Une procédure prenant en paramètre un entier **n**, une liste à deux dimensions de **n** lignes et **n** colonnes, et un t-uple de trois réels **(x_1,x_2,x_3)** vérifiant **$0<x_1<x_2<x_3<1$**. Elle fera "tomber" par gravité les cellules situées au dessus de cellules dont la valeur est égale à 0. Enfin, elle remplira les colonnes incomplètes grâce à la première fonction de la sous-partie 3.1.

Exemple de fonctionnement : le code suivant (en appelant respectivement les trois procédures précédentes "propagation", "modification" et "gravity")

```
n = 5
proba=(0.05,0.30,0.6)
gameBoard = [[1,1,1,1,3],[1,2,4,1,1],[2,2,4,1,1],[4,2,2,3,3],[3,3,2,3,3]]
display(gameBoard,n)
current = (3,2)
L = [current]
propagation(gameBoard,L,current,n)
print(L)
print()
modification(gameBoard,L,n)
display(gameBoard,n)
gravity(gameBoard,n,proba)
display(gameBoard,n)
```

doit conduire à ce résultat :

```
1 1 1 1 3
1 2 4 1 1
2 2 4 1 1
4 2 2 3 3
3 3 2 3 3

[(3, 2), (4, 2), (3, 1), (2, 1), (1, 1), (2, 0)]

1 1 1 1 3
1 0 4 1 1
0 0 4 1 1
4 0 3 3 3
3 3 0 3 3

3 3 2 1 3
1 2 1 1 1
1 3 4 1 1
4 1 4 3 3
3 3 3 3 3
```

4 UNE INTERFACE GRAPHIQUE

Il vous est fortement recommandé de lire l'intégralité de cette partie avant de commencer à coder. Les travaux demandés sont mis en évidence avec une couleur bleue. On devra nécessairement utiliser la librairie graphique Pygame. L'usage d'une autre librairie ne sera pas pris

en compte.

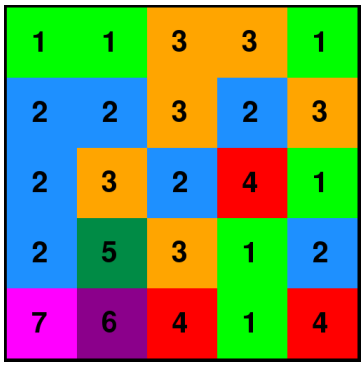
Remarque 1 : dans cette partie on continuera bien sûr à modéliser le plateau de jeu par une liste à deux dimensions d'entiers.

Remarque 2 : on se restreindra dans cette partie à des plateaux de 5 lignes et 5 colonnes, c'est-à-dire aux règles originelles du jeu.

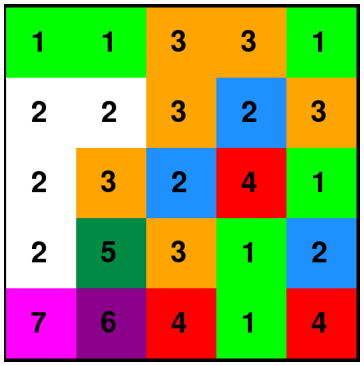
Remarque 3 : vous êtes libre de désigner votre jeu comme bon vous semble tant que les fonctionnalités sont présentes. Les captures d'écran de cette partie ne sont là que pour vous donner une idée.

4.1 AFFICHAGE DE LA GRILLE

Le but de cette partie est d'implémenter les procédures permettant d'afficher la grille. Chaque valeur aura sa propre couleur, donc après quelques tours de jeu vous devrez avoir un affichage ressemblant à celui-ci :



Une fois sélectionné, un ensemble de cellules adjacentes de même valeur devra être mis en évidence, avec du blanc par exemple :

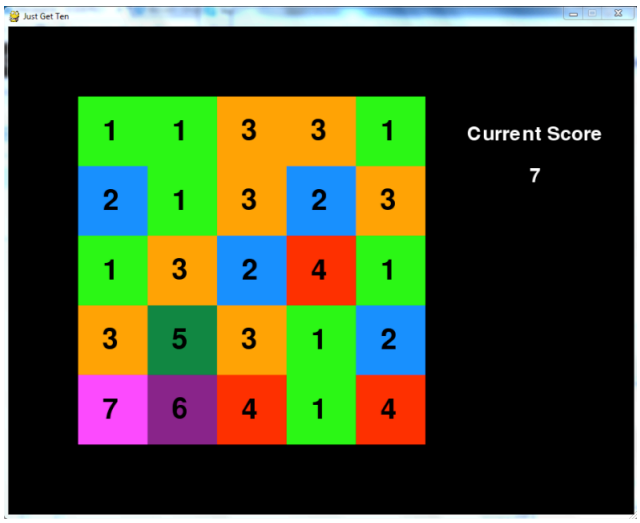


A noter que la sélection d'un ensemble de cellules ne sera pas effectué par les procédures de cette sous-partie mais par celle de la suivante.

Dans un fichier que l'on nommera « justGetTenGUI.py », implémenter les sous-programmes suivants :

- Une procédure prenant en paramètre une liste à deux dimensions représentant la grille, la surface sur laquelle elle sera dessinée, les coordonnées d'une cellule et un booléen. Selon la valeur de ce dernier, la procédure affichera la valeur de la cellule en question (au bon endroit évidemment) soit sur un fond de sa couleur associée, soit sur un fond blanc.
- Une procédure prenant en paramètre une liste à deux dimensions représentant la grille, la surface sur laquelle elle sera dessinée, une liste de coordonnées de cellules et un booléen. Selon le principe précédent, cette procédure affichera la valeur des cellules dont les coordonnées figurent dans la liste passée en paramètre.
- Une procédure prenant en paramètre une liste à deux dimensions représentant la grille, son nombre **n** de lignes et de colonnes, et la surface sur laquelle elle sera dessinée. Cette procédure affichera la valeur de chacune des cellules de la grille sur un fond de sa couleur associée.
- Une procédure score affichant la valeur maximale de la grille.

- Votre fenêtre de jeu ressemblera donc à celle-ci :



4.2 LA PROCEDURE DE JEU

Implémenter dans le fichier précédent une procédure gérant la succession de coups du joueur :

- Cette procédure prend en paramètre une liste à deux dimensions représentant la grille, son nombre **n** de lignes et de colonnes, la surface sur laquelle elle sera dessinée, et un t-uple de trois réels (x_1,x_2,x_3) vérifiant $0<x_1<x_2<x_3<1$.

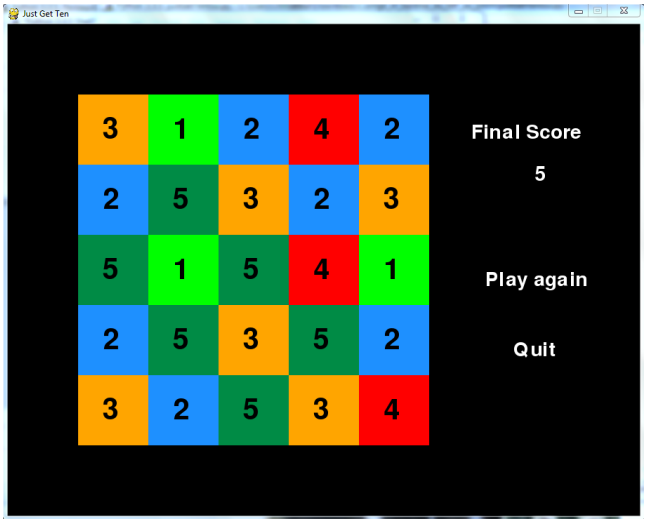
- Tant qu'il est reste au moins un coup possible, le joueur est invité à jouer :
  - Un clic sur une cellule "isolée" ne produit rien.
  - Un clic sur n'importe quelle cellule d'un ensemble de cellules adjacentes de même valeur colorie cet ensemble en blanc.
  - Un clic sur une des cellules d'un ensemble déjà sélectionné produit la fusion vers cette cellule. La grille est alors modifiée selon les règles du jeu.
  - Si un ensemble est sélectionné, un clic sur une cellule n'y appartenant pas le recolore, et décolore éventuellement un autre ensemble.
  - A tout moment le joueur peut quitter le jeu en cliquant sur la croix ou en appuyant sur la touche d'échappement.
- Après chaque coup le score est mis à jour.

4.3 LA PROCEDURE PRINCIPALE

Implémenter dans le fichier précédent une procédure permettant de jouer à Just Get 10 :

- Cette procédure devra créer une surface graphique, initialiser une grille selon un t-uple de probabilités et faire jouer le joueur.
- A la fin de la partie on proposera au joueur le choix entre rejouer ou quitter le jeu.

Voici un **exemple** de fin de partie :



5 DES BONUS

Vous êtes libre d'implémenter zéro, un ou plusieurs des bonus suivants.

5.1 SAUVEGARDER UNE PARTIE

Rajouter une option permettant de sauvegarder l'état d'une partie dans un fichier texte.
Implémenter également une possibilité de reprendre le cours d'une partie mémorisée dans un fichier texte.

5.2 UNE ANIMATION POUR UN 10

Rajouter une animation visuelle et sonore (se documenter sur ce dernier point) lorsque le joueur atteint 10. Faire en sorte que la partie puisse ensuite reprendre.

5.3 DIFFERENTS MODES DE JEU

Proposer le choix entre différents modes de jeu. En voici quelques exemples :

- Grille plus petite (4 lignes et 4 colonnes)
- Grille plus grande (6 lignes et 6 colonnes)
- Temps limité pour chaque coup
- Temps limité pour atteindre 10

…

6 BAREME INDICATIF

Ce barème peut-être amener à évoluer, il n'est donc qu'i**ndicatif**.

- Partie 3: 20 points
- Partie 4 : 20 points

- Bonus : 10 points

Ce qui fait un total de 40 points, votre soutenance étant évaluée sur 20 points. La note totale sera alors ramenée sur 20 points par proportionnalité.