UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

Outils Info. Snake

```
# rendu
cd ~/prog-imp/
zip -r tpsnake.zip tpsnake/

# mise en place
mkdir ~/prog-imp/tpsnake
cd ~/prog-imp/tpsnake
emacs compte-rendu.txt &
```

Aides et Rappels

Il est indispensable de lire cette partie qui vous aide pour la réalisation du TP.

NB : si vous écrivez un programme qui prends trop de temps, vous pouvez l'interrompre avec `Ctrl+C`.

Structure typique d'un programme avec fonction « principale »

Un programme Python doit être proprement structuré avec 4 sections : 1) les imports des modules utilisés par le programme, 2) la définition des fonctions nécessaires, 3) la définition de la fonction principale, qui contient le code de base du programme, 4) l'appel de la fonction principale.

Voici un exemple qui trace trois carrés.

```
1
2  # 1) importation de bibliothèques
3  from qtido import *
4
5  # 2) définition des fonctions
6  def carre(fen, x, y, taille):
7      rectangle(fen, x, y, x+taille, y+taille)
8
9  # 3) définition de la fonction principale
10 def principale():
11     f = creer(600, 500)
12     carre(f, 100, 100, 100)
13     carre(f, 400, 100, 100)
14     carre(f, 225, 300, 150)
15     attendre_pendant(f, 1000)
16     exporter_image(f, "toto.png")
17
18 # 4) appel de la fonction principale
19 principale()
```

Technique de la simulation

Pour programmer une simulation, il faut d'abord réfléchir à ce qu'est l'état du système à simuler et comment il doit être stocké (quelles variables, avec quels noms et quel contenu). Il faut ensuite réfléchir à l'état initial du système (valeur au début de la simulation). Ensuite il faut programmer : 0) l'initialisation des variables auxiliaires, 1) l'initialisation de l'état du système, 2) la boucle de simulation avec une condition d'arrêt (si elle existe), 3) dans la boucle, l'affichage du système, 4) dans la boucle, la gestion des événements (temps, touches clavier, etc.) qui **n'affiche pas** et ne fait que modifier l'état.

Voici un exemple de la partie principale d'un programme de simulation d'un disque qui grandit avec le temps et prends une taille au hasard s'il devient trop grand.

```
    # 0) initialisation des variables auxiliaires
    f = creer(400, 400)
    # 1) initialisation de l'état et autre
    rayon = 50
    # 2) boucle de simulation avec condition d'arrêt
    while not est_fermee(f):
        # 3) affichage du système
        effacer(f)
        disque(f, 200, 200, rayon)
        # 4) gestion des événements : juste modifier l'état
        attendre_evenement(f, 10)
        ev = dernier_evenement(f)
        if ev == None: # temps
            rayon = rayon + 2
            if rayon > 200:
                rayon = random.random()*100
        elif ev == TOUCHE_ECHAP:
            exit()
```

Aide diverse sur Python et qtido

Utilisation de listes pour des coordonnées

On peut utiliser une liste Python pour représenter des paires, triplets, etc (des valeurs qui vont ensemble). Par exemple, la position en 2D d'une planète pourrait être représentée par `pos = [384.4, 0]`.

Utilisations avancées des listes : mots clés `in` et `del`

Quand on a une liste (ou équivalent), il est possible d'utiliser le mot clé spécial `in` pour savoir si une valeur apparaît dans cette liste. Ainsi, `123 in [10, 20, 123, 40]` vaut `True`, et `"toto" in [10, 20, 123, "bonjour"]` vaut `False`. Il est entre autre possible de tester si une valeur complexe est contenue dans une liste, par exemple `[1, 2] in ["toto", [1, 2], 1234]` vaut `True`, par contre `[1, 2] in ["a", 1, 2]` vaut `False`. Ce `in` peut être bien pratique dans ce TP. Attention, ce `in` ne doit pas être confondu avec `for ... in ...` qui a une sémantique totalement différente.

Comme `l.append(element)` permet de modifier `l` en lui ajoutant un élément, `del l[i]` permet de modifier `l` en lui supprimant l'élément d'indice `i`.

Affichage et re-affichage de la fenêtre qtido

Pour des raisons de vitesse d'affichage, qtido ne rafraîchi pas le contenu de la fenêtre à chaque fois que l'on trace dedans. Le ré-affichage n'est garanti que lorsque qu'une des fonctions suivantes est appelée : `attendre_pendant`, `attendre_evenement`, `re_afficher`. Si rien ne semble être affiché, le problème est peut être que ces fonctions ne sont jamais appelées.

De plus, il ne faut pas oublier d'appeler `effacer` pour repeindre en noir toute la fenêtre. Tant que `effacer` n'est pas appelée, les nouvelles figures sont tracées par dessus les anciennes.

But du TP

Le but est de concevoir un « serpent ». Il est recommandé de suivre les étapes (exercices) et de copier le fichier de l'exercice précédent pour commencer chaque nouvel exercice (penser à bien ouvrir, modifier et

exécuter le nouveau fichier). Les étapes servent à réaliser des sous parties du serpent, en pouvant vérifier à chaque fois que ce qui est fait semble fonctionner. Par exemple, au départ nous allons ignorer le score, la pomme et le temps qui passe.

Exercice 1 – Affichage du serpent

Pour cet exercice, travaillez dans un fichier 'serpent-1.py', directement dans le dossier `tpsnake`.

En ignorant le score et la pomme, faite un programme (pas forcément une simulation) qui défini un état initial pour le serpent et l'affiche. Vous serez donc amené à vous poser la question de comment représenter l'état du serpent, comment le stocker en Python et comment l'afficher dans une fenêtre qtido.

Exercice 2 – Ajout de la pomme

Pour cet exercice, travaillez dans un fichier 'serpent-2.py', directement dans le dossier `tpsnake`.

Faite que votre programme gère aussi la position et l'affichage de la pomme, dont la position sera tirée au hasard. Ce tirage d'une position au hasard sera réutilisé par la suite : vous pouvez donc directement définir une fonction pour cette fonctionnalité.

Exercice 3 – Simulation

Pour cet exercice, travaillez dans un fichier 'serpent-3.py', directement dans le dossier `tpsnake`.

Faite que votre programme soit maintenant une simulation mais que le serpent ne bouge pas. Pour vérifier que votre simulation est bien animée, pour l'instant, vous pouvez à chaque pas de temps (par exemple toutes les 400 millisecondes) replacer la pomme au hasard.

Exercice 4 – Déplacement du serpent

Pour cet exercice, travaillez dans un fichier 'serpent-4.py', directement dans le dossier `tpsnake`.

À chaque pas de temps, selon la touche appuyée, faites que le serpent se déplace dans la direction correspondante. Il continue tout droit si aucun touche n'est appuyée entre temps.

Exercice 5 – Gestion des bords

Pour cet exercice, travaillez dans un fichier 'serpent-5.py', directement dans le dossier `tpsnake`.

Si la tête du serpent sort d'un des 4 cotés de l'écran, la partie est perdue. Une alternative est de le faire « traverser » vers l'autre coté de l'écran.

Exercice 6 – Gestion de la pomme

Pour cet exercice, travaillez dans un fichier 'serpent-6.py', directement dans le dossier `tpsnake`.

Si la tête du serpent arrive sur la pomme, le serpent doit grandir et la pomme réapparaître. Le serpent peut grandir tout de suite (plus simple) ou quand la pomme mangée arrive au bout du serpent.

Exercice 7 – Autres fonctionnalités (ordre au choix)

- perdre quand le serpent s'auto-mange,
- gestion et affichage du score,
- ne pas faire re-apparaître la pomme sur le serpent,
- (opt) taille en nombre de case du niveau choisie à l'aide des paramètres du programme,
- (opt) accélérer la partie au fur et à mesure,
- (opt) afficher les cases du serpent de couleur différentes (ex: une case sur deux plus claire).