TP Snake

Le Snake est un jeu très simple. Vous êtes au contrôle d'un serpent et devez manger de la nourriture afin de grandir. À chaque fois que le serpent mange, il devient un peu plus long. Vous gagnez quand votre serpent occupe tout l'espace disponible, et vous perdez si vous vous mangez vous même ou si vous mangez un bord (quelle idée saugrenue).

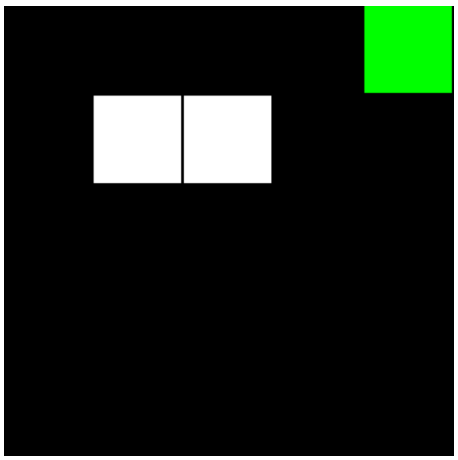

FIGURE 1 – Un exemple de Snake

I Création du Snake

Le jeu sera représenté par une liste contenant (dans cet ordre là) : le serpent, la position de la nourriture, le nombre de tours joués ainsi que la taille du plateau. Le serpent sera représenté par une liste de tuple contenant (encore dans cet ordre) : le tour d'ajout de la partie du corps, la position de la partie du corps. Avec cette représentation, l'exemple est donc :
[[(2, (1,1)), (3, (2,1))], (4,0), 3, 5]
(on est donc au $4^{\text{ème}}$ coup)

1. Créer une fonction `coordonnees_aleatoire` qui prend en argument la taille de la grille, et renvoit un couple de coordonnées (entières) aléatoires dans cette grille (on pourra utiliser la fonction `randint` du module `random`).
2. Créer une fonction `element_aleatoire` qui prend en argument une liste et renvoit un élément aléatoire de cette liste.
3. Créer une fonction `serpent` qui prend en argument le jeu (une liste contenant le serpent, la nourriture, le nombre de coups joués et la taille du plateau) et qui renvoit la liste des coordonées de chaque partie du serpent.
4. Créer une fonction `cree_jeu` prenant en argument la taille du plateau et renvoyant un nouveau jeu (le serpent commencera avec une seule partie de sont corps). On prendra soin de ne pas avoir le serpent et la nourriture au même emplacement.

II Interface graphique

Par la suite, on importe le programme Python joint avec ce TD à l'aide de la commande :

```
import snakeScreen as s
```

Ce programme a 3 utilités :

— créer une boucle qui va exécuter 5 fois par seconde la fonction donnée en argument en utilisant la commande : `s.start(fonction)`(veillez à ce que cette commande soit située sur la dernière ligne de votre programme)

— En utilisant la fonction précedente, python va faire apparaître une fenêtre carrée noire de 500 pixels de côté. Il est possible d'y afficher des carrés avec la commande :

```
s.canvas.create_rectangle(x1,y1,x2,y2,fill = "white")
```

`x1,y1,x2,y2` sont des entiers entre 0 et 499 représentant les coordonées de deux angles opposés du carré.

— Ce programme contient la variable *s.direction* qui prend la valeur 0 si l'on a appuyé sur ↑, 1 pour →, 2 pour ↓, 3 pour ← et la valeur **None** si aucune touche n'a été appuyée.

5. Créer une variable global *jeu* qui prend la velur de sortie de `cree_jeu(5)` (Ou un autre chiffre que 5) puis créer une fonction `fenetre` sans argument dans laquel on accéde à la variable *jeu* avec le mot clef `global`.(C'est cette fonction que l'on vas donner en argument à `s.start(fonction)`)
6. Créer une fonction `carre` prenant en argument deux coordonnées (`x` et `y`), ainsi que la taille de la grille et qui va dessiner un carré sur l'écran correspondant à la case de la grille de coordonnées (x,y).
7. Modifier la fonction `fenetre` afin d'afficher le serpent et la nourriture sous forme de carré sur l'écran. (vous pouvez modifier la fonction `carre` pour permettre de choisir la couleur du carré et afficher la nourriture en vert en utilisant la couleur **"lime"**).
8. Écrire une fonction qui prend en argument une liste et qui renvoie l'indice du maximum(ne renvoit qu'un seul indice s'il y a plusieurs maximum).
   Faire une autre fonction qui renvoie l'indice du minimum (elle ressemble fortement à la première fonction...).
9. Créer une fonction qui prend en argument une liste et qui supprime le minimum en temps constant (cette fonction ne doit rien renvoyer, les modifications doivent être apportées par effets de bords).
10. Créer la fonction `deplacement_permit` prenant en argument le jeu et un couple de coordonnées et renvoie **True** si la case est dans la grille et est libre (la nourriture n'occupe pas de case, mais le serpent oui) et **False** sinon.
11. Créer la fonction `recommencer` sans argument et qui va modifier la variable globale *jeu* pour créer un nouveau jeu, et va donner la valeur **None** à la variable *s.direction*.
12. Écrire une fonction `pas` prenant en argument le jeu et joue un coup :
    — La tête du serpent est la partie la "plus jeune" de sont corp (l'élément maximum de *jeu[0]*)

— Il faut ajouter une nouvelle partie au corps du serpent (juste à gauche, en haut, à droite, ou en bas de la tête) en fonction de la valeur de `s.direction` qui indique la direction dans laquelle se dirige le serpent.
— Si cette partie du serpent est sur la même case que la nourriture, on modifie aléatoirement les coordonnées de la nourriture. Sinon on enlève la "plus vieille" partie du corps du serpent. (l'élément minimum de *jeu[0]*)
— Si l'emplacement de la nouvelle partie du serpent n'est pas valable, le jeu recommence, de même si le jeu est fini. Une fois cette fonction écrite, appelez-la dans la fonction `fenetre` (afin qu'elle s'exécute 5 fois par seconde), lancez le jeu et essayez de gagner.

III Pour aller plus loin (le vrai TD...)

L'objectif de cette partie est de créer une stratégie automatique capable de jouer (plus ou moins bien) au Snake.

13. Donnez une stratégie permettant de gagner à tous les coups une partie de Snake (même si la partie est longue, le seul but est de la finir).

Algorithme du plus court chemin (une version parmi tant d'autres)

14. Créer une fonction `convertir_grille` prenant en argument le jeu et renvoyant une matrice carrée (de même taille que le plateau) contenant la valeur **False** pour les cases sur lesquelles se trouve une partie du serpent et **True** pour les autres.
15. Écrire une fonction `voisins` prenant en argument une grille du jeu et une position, et renvoyant la liste des cases voisines de la position sur lesquelles il est possible de se rendre (pas de serpent ou de bordure...)
16. Créer une fonction `front_voisin` qui prend en argument une grille du jeu, ainsi qu'une liste de positions et qui renvoie une liste contenant toutes les cases voisines accessibles de toutes les positions données en arguments (il est inutile de faire figurer plusieur sfois la même case)
    La fonction doit aussi modifier la grille par effet de bord et mettre la valeur **False** à toutes les cases correspondantes positions renvoyées par la fonction.
17. À l'aide de la dernière fonction, créer la fonction `plus_court_chemin` prenant en arguments la grille du jeu, une position de départ et une position d'arrivée et renvoyant la taille du plus court chemin séparant les deux positions. (renvoie **False** si ces deux positions ne peuvent pas se rejoindre).
18. Créer finalement la fonction `strategie` prenant en argument le jeu, et modifiant la valeur *s.direction* de manière à ce que le serpent se dirige vers la nourriture.(Si la nourriture ne peut pas être atteinte, il faut diriger le serpent vers une position legale. Si aucune des directions n'est legale, peu importe la valeur de *s.direction*, la partie est perdue d'avance. la fonction ne modifie donc pas la valeur de *s.direction*)