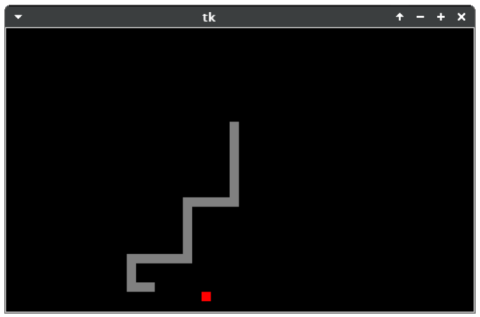
TP n° 15

Programmation du jeu Snake

Inspiré d'un TP de V.Picard.

Les jeux Snake font partie des jeux vidéos légendaires. Il s'agit de contrôler un serpent qui avance continuellement. Lorsqu'il mange des pommes, il grandit. S'il heurte un mur ou lui-même la partie est perdue.



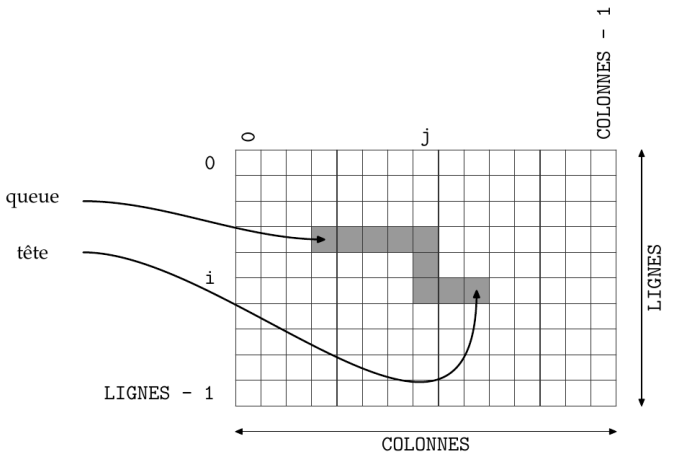
Pour vous faciliter le travail, on part d'un fichier squelette `snake_squelette.py` qui contient le code suivant :

- Les imports nécessaires pour le jeu.
- Les paramètres du jeu (que vous pouvez éventuellement modifier).
- Le code permettant de créer la fenêtre de jeu.
- Une fonction `couleurPixel(i, j, c)` permettant de changer la couleur du pixel de coordonnées *(i,j)*.
- Une section Initialisations permettant de définir les variables globales du jeu ainsi que le code d'initialisation.
- Des petites fonctions qu'il vous faudra écrire ou améliorer (voir la suite du sujet).
- La boucle principale du jeu dans la fonction `boucle()`.
- Le code permettant la gestion des événements claviers.
- À la fin les instructions pour démarrer le jeu.

1 Le modèle du jeu

La zone de jeu est une grille de pixels, représentée ci-dessus, de dimension LIGNES × COLONNES. Les pixels sont repérés par leur coordonnées *(i,j)* avec i l'indice de ligne et j l'indice de colonne. Le pixel de coordonnées *(0,0)* est celui en haut à gauche.

Le serpent est représenté à l'aide d'une liste de coordonnées *(i,j)*. Le premier élément de la liste correspond à la queue du serpent et le dernier élément correspond à la tête. Dans



l'exemple, le serpent correspond donc à la liste `serpent = [(3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (4, 7), (5, 7), (5, 9), (5, 9)]`

Dans le fichier squelette la variable globale `serpent` est définie avec une position initiale arbitrairement choisie.

Question 1 :

Chargez le fichier `snake_squelette.py` sous Pyzo et lancez le jeu ; vous devriez voir apparaître une fenêtre noire vide.

.....

Question 2 :

Dans la partie `Dessin initial du serpent`, écrire les lignes de code permettant de colorer les pixels correspondants à la position initiale du serpent. On utilisera la fonction `couleur Pixel(i,j, c)` et on pourra choisir la couleur "gray".

Lancez le jeu, vous devriez voir apparaître une fenêtre contenant le serpent immobile.

.....

2 Faire avancer le serpent

Question 3 :

Écrivez le contenu de la fonction `prochaineCase()` pour qu'elle retourne *pour l'instant* les

coordonnées de la case juste à droite de la tête du serpent.

.....

Question 4 :

Écrivez le contenu de la fonction `avance(i, j)`. Cette fonction commence par `global serpent` car elle devra modifier la variable globale `serpent`. Cette fonction fait avancer le serpent vers la case *(i,j)* donnée en argument ; pour cela il faudra :

- colorer la case *(i,j)*
- colorer en noir la case qui correspond à la queue du serpent
- ajouter dans serpent la case *(i,j)* à la tête
- enlever dans serpent la case qui correspondait à la queue

.....

Un jeu interactif fonctionne le plus souvent à l'aide d'une boucle principale qui s'exécute à intervalles réguliers. Cette boucle calcule le nouvel état du jeu après l'intervalle de temps, puis produit l'affichage du jeu (on parle de l'affichage d'une frame). S'il y a un nombre suffisant de frames par seconde, le jeu paraît animé.

Question 5 :

Programmez dans `boucle()` le comportement suivant : récupérer la case suivante *(i,j)* puis, après avoir vérifié qu'il n'y a pas collision (déjà codé), faire avancer le serpent à l'aide de la fonction `avance`.

Lancez le jeu , vous devriez voir le serpent avancer vers la droite. Lorsqu'il touche le bord de l'écran une erreur se produit car nous n'avons pas encore codé la fonction collision.

.....

Question 6 :

Écrivez le contenu de la fonction `collision` qui retourne `True` lorsque le serpent heurte quelque chose. Pour le moment une collision se produit si les coordonnées *(i,j)* se situent en-dehors de la zone de jeu (le serpent heurte un mur) ; on implémentera plus tard les collisions du serpent avec lui-même.

Lancez le jeu, cette fois ci quand le serpent heurte le jeu s'arrête et un message "PERDU !!!" s'affiche dans la console.

.....

3 Contrôle du serpent à l'aide du clavier

Question 7 :

Créez une variable globale `direction` initialisée à "droite". Modifier le contenu des fonctions `toucheHaut`, `toucheBas`, `toucheDroite` et `toucheGauche` pour qu'elles modifient le contenu de la variable `direction` de façon appropriée. Ces fonctions afficheront aussi la nouvelle direction, par exemple avec `print("HAUT")`. dans toucheHaut pour vérifier que tout fonctionne correctement.

.....

Si on lance le jeu, il ne se passe rien de nouveau excepté l'affichage des messages quand on appuie sur les touches fléchées du clavier.

Question 8 :

Modifiez la fonction `prochaineCase(i, j)` pour que le calcul de la case suivante tienne maintenant compte de la position de la tête du serpent mais **aussi** de la variable direction.

Lancez le jeu, le serpent devrait être contrôlable par les touches fléchées ; le jeu s'arrête lorsque le serpent heurte l'un des murs. Par contre on peut remarquer que le serpent peut se traverser lui-même.

.....

Question 9 :

Faites des ajouts à la fonction `collision(i, j)` pour tester si le serpent va avancer sur lui-même. Autrement dit, si les coordonnées *(i,j)* passées en argument correspondent à une des cases de la liste `serpent` alors la fonction retourne aussi `True`.

Lancez le jeu et testez que le serpent est bien contrôlable et que la partie s'arrête lorsque le serpent heurte un mur ou lui-même.

.....

4 Mangez des pommes !

Question 10 :

Ajoutez une variable globale `pomme` initialisée (par exemple) à *(10, 8)*. Puis compléter la partie `Dessin initial de la pomme` pour colorer cette case en rouge.

En lançant le jeu vous devriez voir une pomme mais rien ne se passe quand le serpent essaie de la manger.

.....

Question 11 :

Complétez la fonction `mange(i, j)` qui est similaire à `avance(i, j)` mais dans laquelle case *(i,j)* est supposée être une pomme. Dans ce cas, contrairement à `avance`, on ne supprime pas la case de queue du serpent, ce qui aura pour effet de le faire grandir d'une unité.

.....

Question 12 :

Modifiez la boucle principale pour qu'elle fonctionne ainsi :

- on calcule *(i,j)* les coordonnées de la case suivante.
- si la case suivante produit une collision, le jeu est perdu,
- sinon, si la case suivante est celle de pomme alors on fait manger le serpent,
- sinon on fait avancer le serpent.

Lancez le jeu : désormais le serpent peut manger la pomme et grandir d'une unité. Toutefois on peut constater que la pomme reste présente (invisible) et le serpent grandit chaque fois qu'on repasse sur cette pomme invisible.

.....

Question 13 :

Complétez la fonction `nouvellePomme()`. Cette fonction calcule des coordonnées aléatoires pour la nouvelle pomme à l'aide de `randint` qui prend comme arguments 2 entiers et renvoie un entier aléatoire entre ses deux arguments inclus. Si ces coordonnées ne conviennent pas (le

serpent est déjà dessus), on en choisit de nouvelles jusqu'à en trouver qui conviennent. Une fois des coordonnées acceptables trouvées, on modifie la variable `pomme` et on colore la case correspondante.

.....

Question 14 :

Ajoutez dans la boucle principale, l'appel à la fonction `nouvellePomme()` pour régénérer une pomme au bon moment.

.....

Félicitations, votre jeu est maintenant fonctionnel !

5 Score

Question 15 :

Ajoutez une variable globale `score` et le code associé, qui compte le nombre de pommes mangées. Le score final sera affiché lorsque la partie se termine.

.....

6 Améliorations possibles

Voici quelques pistes à implémenter si vous en avez envie, par forcément dans l'ordre indique :

- **Anti-demi-tour** : Vous remarquerez que le serpent peut revenir sur ses pas, ce qui fait perdre immédiatement la partie : par exemple il se dirige vers la droite et le joueur appuie sur gauche. Pour éviter cela, modifiez les fonctions `toucheXXX` pour qu'elles soient inactives si elles correspondent à un demi-tour.
- **Niveaux**:Programmez différents niveaux de difficulté qui correspondront à des vitesses de jeu différentes. Lorsque le joueur atteint un certain score le niveau de jeu augmente.
- **Barre d'énergie** : Le serpent dispose d'une barre d'énergie. Lorsqu'il avance il perd un peu d'énergie. Lorsqu'il tourne il en perd un peu plus. Lorsqu'il mange une pomme il regagne de l'énergie. Si l'énergie arrive à 0 la partie est perdue. Cela incite le joueur à récupérer les pommes rapidement et sans zigzags.
- **Pommes empoisonnées** : Au bout d'un certain nombre de frames (pas trop petit quand même), la pomme devient verte et empoisonnée (date limite de consommation). Si on la mange on perd une ou deux longueurs de serpent ; si le serpent a une longueur nulle, on perd. Quand une pomme devient verte, une nouvelle pomme rouge apparaît. Les pommes vertes restent présentes jusque la fin de la partie (il peut donc y en avoir plusieurs).