LE GOUVERNEMENT
DU GRAND-DUCHÉ DE LUXEMBOURG
Ministère de l'Éducation nationale,
de l'Enfance et de la Jeunesse

EXAMEN DE FIN D'ÉTUDES SECONDAIRES CLASSIQUES
Sessions 2022

DISCIPLINE	SECTION(S)	ÉPREUVE ÉCRITE	
Informatique	CB	*Date de l'épreuve :*	02.06.22
		Durée de l'épreuve :	08:15 - 11:25
		Numéro du candidat :	

Partie au choix **Choisissez une question parmi les deux suivantes et indiquez votre choix avec un X.**			
Question	**Nb points**	**Sujet**	**Choix du candidat**
Question 1	14 points	Code de César	
Question 2	14 points	Fractions égyptiennes	
Partie obligatoire			
Question	**Nb points**	**Sujet**	**Obligatoire**
Question 3	46 points	Pacman	X

Créez sur votre ordinateur un dossier nommé suivant les instructions du professeur. Tous vos fichiers devront être stockés dans ce dossier. En particulier, si vous avez choisi la question 1, copiez dans ce dossier le fichier **`cypher.txt`**, dont l'emplacement est indiqué par votre professeur. En haut de chaque fichier Python, vous indiquerez en tant que commentaire votre lycée et votre numéro de candidat. A la fin de l'épreuve vous imprimerez et signerez les codes.

Partie au choix

Question 1 – Code de César (14 p.)

Cette question sera traitée dans le fichier **`q1.py`** de votre dossier. L'unique importation permise est **`ascii_uppercase`** du module **`string`**. On rappelle que **`ascii_uppercase`** est le string **`'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`** formé des 26 lettres majuscules de l'alphabet.

Le code de César est la méthode de chiffrement la plus ancienne de l'histoire. César l'a utilisée pour envoyer des messages codés à ses généraux. L'idée est simple : toute lettre du message original est remplacée par la lettre qui se trouve décalée de **`n`** places à droite dans l'alphabet. **`n`** est appelé le « shift » ou « décalage » du code. Les caractères spéciaux ne sont pas chiffrés.

Par exemple si **`n = 3`**, on remplace **`A`** par **`D`**, **`B`** par **`E`**, **`C`** par **`F`**, etc... Pour les dernières lettres, on reprend au début : ainsi **`X`** devient **`A`**, **`Y`** devient **`B`** et **`Z`** devient **`C`**.

Tableau d'encodage pour un shift de 3 lettres :

```
Texte clair : A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Texte codé :  D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
```

(1) Définir la fonction **`encryption_dict(shift)`**, qui retourne dans un dictionnaire le tableau d'encodage correspondant au **`shift`** donné en paramètre. Les clés du dictionnaire sont les 26 lettres majuscules. A chaque clé est associée la lettre majuscule du chiffrement. On veillera à ce que la fonction retourne également le résultat correct lorsque le shift est négatif, ce qui correspond à un décalage vers la gauche (utile pour le décodage). (3 p.)

(2) Le fichier **`cypher.txt`** contient un poème allemand chiffré à l'aide de la méthode de César. Le texte contient uniquement des lettres majuscules et des caractères spéciaux tels que l'espace ou les caractères de ponctuation. L'objectif de cette question est de décoder le poème et de sauvegarder le texte clair dans un fichier **`plain.txt`** (qui devra être créé dans votre dossier).

a) Dans le programme principal, ouvrir le fichier **`cypher.txt`** en mode lecture et le fichier **`plain.txt`** en mode écriture. Les variables contenant ces fichiers seront appelées **`cypher_file`** et **`plain_file`** respectivement. (0,5 p.)

b) Pour déterminer le shift du code, il faut d'abord faire une analyse de fréquence du texte chiffré, c.-à-d. compter le nombre de fois que chaque lettre majuscule y apparaît. Sachant que la lettre la plus fréquente dans le texte clair est le **`'E'`**, on pourra en déduire le shift.

- Créer une liste **`frequencies`** avec 26 entiers initialement égaux à 0. (0,5 p.)
- Parcourir le texte chiffré et compter le nombre d'apparitions de chaque lettre : le 1er élément de **`frequencies`** devra contenir le nombre d'apparitions du **`'A'`**, le 2e élément devra contenir le nombre d'apparitions du **`'B'`** etc., jusqu'au dernier élément, qui devra contenir le nombre d'apparitions de **`'Z'`**. Fermer ensuite le fichier **`cypher.txt`**. (3 p.)
- Rechercher maintenant l'indice de la lettre avec la fréquence maximale dans le texte chiffré (on admet qu'il y en a une seule) et en déduire le shift du code, à stocker dans une variable **`shift`**. (2 p.)

c) A l'aide de la fonction **`encryption_dict`**, créer un dictionnaire **`d`** qui permet de réaliser le déchiffrement. ***Aide*** : déchiffrer = appliquer le shift opposé au texte. (1 p.)

d) Rouvrir le fichier **`cypher.txt`** et le parcourir à nouveau, ligne par ligne. Chaque ligne doit être décodée et le texte clair doit être écrit dans le fichier **`plain.txt`**. Le formatage du texte chiffré doit être respecté : tous les caractères du texte clair (y compris les caractères spéciaux) doivent se trouver au même endroit que dans le texte chiffré. Pour terminer, fermer les deux fichiers **`cypher.txt`** et **`plain.txt`**. (4 p.)

Question 2 – Fractions égyptiennes (14 p.)

Cette question sera traitée dans le fichier **`q2.py`** de votre dossier. ***Importations permises en cas de besoin*** : **`floor`**, **`ceil`** du module **`math`**.

Une **fraction égyptienne** est une fraction du type $\frac{1}{n}$,

- de ***numérateur*** égal à 1 et
- de ***dénominateur*** un entier $n \geq 2$.

Les premières fractions égyptiennes sont donc : $\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}$ ...

Les anciens Égyptiens ne connaissaient que les fractions de ce type.

Le but de la question est de décomposer une fraction quelconque dans l'intervalle $]0,1[$ en une somme de fractions égyptiennes par l'algorithme dit « **glouton** » (all. : « gefräßig »). Par exemple :

$$\frac{5}{7} = \frac{1}{2} + \frac{1}{5} + \frac{1}{70}$$

(1) Les fractions (qu'on peut supposer toutes positives) seront implémentées sous la forme de tuplets **`(a,b)`** d'entiers, avec **`b`** $\neq 0$.

a) Ecrire une fonction **`gcd(a,b)`** qui retourne le pgcd (plus grand commun diviseur) de deux entiers positifs **`a`** et **`b`**. (2 p.)

Rappel (algorithme d'Euclide) : $\gcd(a,b) = \begin{cases} a \text{ si } b = 0 \\ \gcd(b, a \bmod b) \text{ sinon} \end{cases}$

b) Ecrire une fonction **`simplify(f)`**, qui pour une fraction donnée **`f`**, retourne la fraction simplifiée. (N.B. : **`f`** est un tuplet !) (2 p.)

c) Ecrire une fonction **`inverse(f)`**, qui retourne l'inverse d'une fraction donnée **`f`** non nulle. (1 p.)

d) Ecrire une fonction **`sub(f1,f2)`**, qui retourne la différence simplifiée de deux fractions données **`f1`** et **`f2`**. (2 p.)

e) Ecrire une fonction **`are_equal(f1,f2)`**, qui retourne **`True`** si les deux fractions **`f1`** et **`f2`** sont égales, **`False`** sinon. (1 p.)

(2) Soit maintenant une fraction positive $f = \frac{a}{b}$ avec $a < b$, donc un nombre rationnel de $]0,1[$.

On demande d'écrire une fonction **`egyptian_decomposition`** qui retourne dans une liste **`terms`** des fractions égyptiennes dont la somme est égale à f (cf. exemple d'exécution de la question 3). Le candidat peut choisir de présenter une solution ***récursive*** ou ***itérative*** :

- Dans le cas d'un algorithme **`récursif`**, la fonction a 2 paramètres : la fraction f et la liste **`terms`**, qui doit être vide à l'appel.
- Dans le cas d'un algorithme **`itératif`**, la fonction a un seul paramètre : la fraction f. La liste **`terms`** est alors une variable locale de la fonction.

Voici une description de l'algorithme «**glouton** » à implémenter :

- simplifier f ;
- si $f = 0$, on retourne la liste **`terms`** car il n'y a plus rien à décomposer ;
- si f est déjà une fraction égyptienne, on retourne la liste **`terms`** à laquelle on ajoute f ;
- sinon :

  a) on cherche la plus grande fraction égyptienne $\frac{1}{n}$ telle que $\frac{1}{n} < \frac{a}{b}$ (d'où le nom de l'algorithme), ce qui revient à chercher le plus petit entier n tel que $n > \frac{b}{a}$;

  b) on ajoute cette fraction $\frac{1}{n}$ à la liste **`terms`** ;

  c) on remplace f par $f - \frac{1}{n}$ et on continue avec la décomposition de cette nouvelle fraction, soit par un appel récursif, soit dans une boucle. (5 p.)

(3) Dans le programme principal, tester la fonction **`egyptian_decomposition`** avec $f = \frac{5}{7}$. La liste avec les termes égyptiens doit être affichée sur l'écran, et l'on obtient :

```
[(1, 2), (1, 5), (1, 70)]
```
(1 p.)

Partie obligatoire

Question 3 - Pacman (46 p.)

Le but de cette question est de programmer une version simplifiée du jeu « ***Pacman*** ». Elle sera traitée dans le fichier **`q3.py`** de votre dossier. Le pacman (en jaune) se déplace à l'intérieur d'un labyrinthe et mange un maximum de pac-gommes (en vert) tout en évitant de heurter l'un des quatre fantômes (en bleu). Les seules importations permises sont : **`pygame`**, **`sys`**, **`pygame.locals`** et les fonctions **`randint`**, **`randrange`** et **`choice`** du module **`random`**.

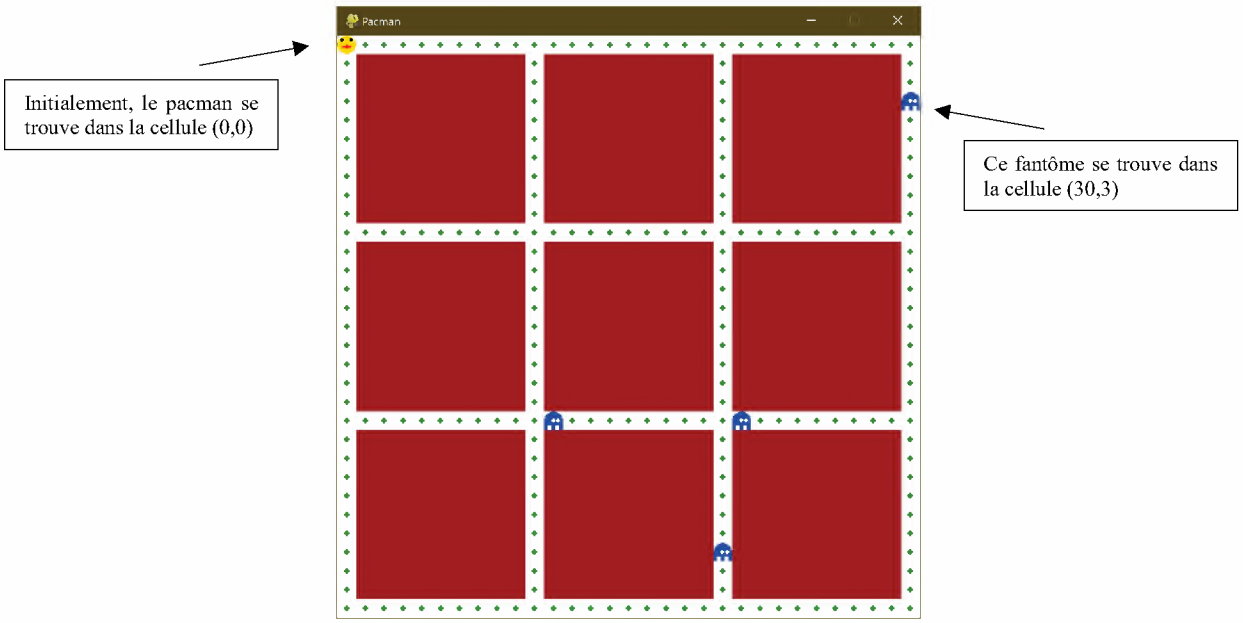

fig. 1 : Jeu au lancement

➢ INITIALISATIONS (2 p.)

(1) Effectuer les initialisations nécessaires pour créer une fenêtre d'application avec un écran carré **`screen`** de dimensions **`620 x 620`** pixels, rempli initialement de la couleur **`'brown'`**. L'écran sera subdivisé dans la suite en **`31 x 31`** cellules carrées ayant chacune **`620//31 = 20`** pixels de côté. On utilisera les constantes **`SIZE = 620`**, **`N_CELLS = 31`** et **`DIM = SIZE // N_CELLS`** dans le code. Le programme devra encore fonctionner correctement si l'on modifie ces constantes. L'entête de la fenêtre est **`'Pacman'`**. Le jeu tourne avec une fréquence de rafraîchissement **`FPS = 25`**. Pour afficher du texte sur l'écran **`pygame`**, on stockera dans la variable **`font`** la police **`'Arial'`** de taille **`16 p`**. (2 p.)

➢ CLASSES (28 = 8+10+10 p.)

(2) Programmer la classe **`Maze`**, représentant un labyrinthe. Cette classe a un seul attribut : **`cells`**. C'est un dictionnaire dont les clés sont les couples de coordonnées **`(x,y)`** des cellules de l'écran avec **`x`** et **`y`** variant de **`0`** à **`N_CELLS-1`**. La valeur associée à une clé **`(x,y)`** peut être l'un des trois strings suivants :

- **`'w'`**, lorsqu'il s'agit d'une cellule de type **`'wall'`** que ni le pacman, ni les fantômes ne peuvent visiter. Ces cellules sont de couleur **`'brown'`**.
- **`'r'`**, lorsqu'il s'agit d'une cellule de type **`'road'`**. Le pacman et les fantômes peuvent se déplacer sur ces cellules. Ces cellules sont dessinées en blanc.
- **`'f'`**, lorsqu'il s'agit d'une cellule de type **`'road with food'`**, donc d'une cellule de type **`'road'`** contenant en plus de la nourriture (pac-gomme). Ces cellules sont aussi

dessinées en blanc et contiennent en leur centre un disque vert (`'green'`) de rayon 3 pixels (bord compris) représentant le pac-gomme. Le bord du pac-gomme est vert foncé (`'darkgreen'`) et a une épaisseur de 1 pixel.

Maze
`cells : {(int,int):str}`

`__init__()`
`draw()`
`is_valid((int,int)) : bool`

Méthodes :

a) Le constructeur `__init__` initialise l'attribut `cells` de telle sorte que toutes les cellules dont les abscisses ou les ordonnées sont divisibles par 10 sont des cellules de type `'road with food'`, toutes les autres cellules étant de type `'wall'`. (3 p.)

b) La méthode `draw` dessine le labyrinthe sur l'écran pygame en respectant les consignes ci-dessus. ***Attention*** : pour dessiner une cellule, il faudra d'abord déterminer les coordonnées en pixels de son coin supérieur gauche. (3,5 p.)

c) La méthode booléenne `is_valid` retourne `True` si le couple `(x,y)` fourni en paramètre représente une cellule valide (donc pas de type `'wall'`) à l'intérieur du labyrinthe. (1,5 p.)

(3) Programmer la classe `Ghost`, représentant un fantôme. Les fantômes peuvent se déplacer sur toutes les cellules de type `'road'` ou `'road with food'`. Ils peuvent se croiser sans problème (puisqu'ils sont immatériels …) et ne mangent rien.

Ghost
`x : int` `y : int`

`__init__(x,y)`
`draw()`
`move(maze:Maze)`

La classe a deux attributs : `x` et `y`, représentant les coordonnées de la cellule dans laquelle se trouve le fantôme.

Méthodes :

a) Le constructeur `__init__` initialise les coordonnées `x` et `y` sur les valeurs des paramètres correspondants. (1 p.)

b) La méthode `draw` dessine le fantôme dans sa cellule : (5 p.)

- dessiner d'abord le disque bleu inscrit dans la cellule (tangent aux bords intérieurement) ;
- recouvrir la moitié inférieure de la cellule par un rectangle bleu ;
- dessiner en bas de la cellule les deux rectangles blancs de sorte que le fantôme obtient trois « pieds » de largeur `DIM//5` et de hauteur `DIM//4`.
- dessiner finalement les deux yeux du fantôme : ce sont des disques blancs de centres `(DIM//2,DIM//2)` et `(3*DIM//4,DIM//2)` respectivement et de rayons `DIM//8`.

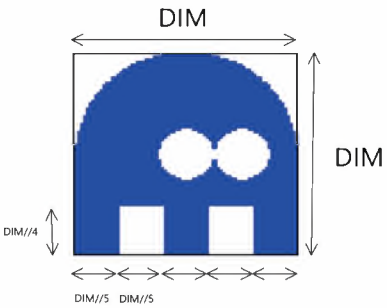

fig. 2

N.B. : a) Les coordonnées des centres des disques blancs sont données relativement au coin supérieur gauche de la cellule ! b) Les bords noirs ne sont pas à dessiner !

c) La méthode **`move`** déplace le fantôme aléatoirement vers une cellule voisine du labyrinthe **`maze`**. Pour cela, on choisit au hasard l'une des quatre cellules au nord, au sud, à l'est ou à l'ouest de la cellule actuelle du fantôme. Aussi longtemps que cette cellule n'existe pas dans le labyrinthe ou est inaccessible (c.-à-d. de type **`'wall'`**), on rechoisit. Le fantôme sera déplacé sur la première cellule valide qu'on trouve. (Vu la forme du labyrinthe, l'une au moins des cellules voisines est toujours accessible !) (4 p.)

(4) Programmer la classe **`Pacman`**, représentant un pacman.

`Pacman`
`x : int` **`y : int`** **`is_alive : bool`** **`score : int`**
`__init__(x,y)` **`draw()`** **`collides_with(ghost:Ghost)`** **`move(maze:Maze,dx:int,dy:int)`**

La classe a les attributs suivants :

- **`x`** et **`y`** : les coordonnées de la cellule où se trouve le pacman,
- **`is_alive`** : **`True`** ou **`False`** suivant que le pacman est en vie ou mort,
- **`score`** : le nombre de pac-gommes que le pacman a mangées.

Méthodes :

a) Le constructeur **`__init__`** initialise les coordonnées **`x`** et **`y`** sur les valeurs des paramètres correspondants. Il initialise **`is_alive`** à **`True`** et **`score`** à 0. (1 p.)

b) La méthode **`draw`** dessine le pacman dans sa cellule. (4 p.)

- Son visage est un disque inscrit dans la cellule (tangent intérieurement aux bords) de couleur **`'gold'`** si le pacman est en vie, de couleur **`'red'`** s'il est mort, c.-à-d. lorsqu'il vient de heurter un fantôme.
- Les yeux sont deux disques noirs de centres **`(DIM//4,DIM//4)`** et **`(3*DIM//4,DIM//4)`** respectivement et de rayons **`DIM//16`**.
- La bouche est une ellipse inscrite dans un rectangle dont les dimensions et la position sont précisées sur la figure ci-dessous. Elle est de couleur **`'red'`** si le pacman est en vie et de couleur **`'gold'`** s'il est mort.

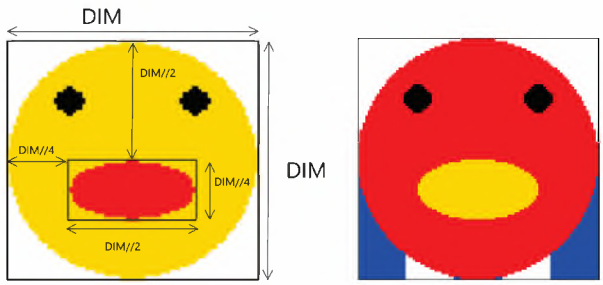

fig. 3 : pacman vivant resp. mort (venant de heurter le fantôme en arrière-plan)

N.B. : a) Les coordonnées des centres des disques sont données à nouveau relativement au coin supérieur gauche de la cellule ! b) Les bords noirs ne sont pas à dessiner !

c) La méthode `collides_with` retourne `True` si le pacman se trouve dans la même cellule que le fantôme `ghost` fourni en paramètre, `False` sinon. (1 p.)

d) La méthode `move` déplace le pacman dans le labyrinthe `maze` (paramètre) vers une cellule voisine en fonction des paramètres `dx` et `dy`.

(***N.B.*** : Lors de l'appel de cette méthode, on choisira toujours :
- `dx = ±1` et `dy = 0` pour un déplacement horizontal,
- `dx = 0` et `dy = ±1` pour un déplacement vertical.)

Le pacman peut uniquement être déplacé s'il est en vie (sinon la méthode n'a aucun effet) et il peut uniquement être déplacé sur une cellule qui n'est pas du type `'wall'`. Si la nouvelle cellule contient de la nourriture, le pacman la mange, c.-à-d. son score est incrémenté de 1 et la nourriture doit disparaître par la suite. (***N.B.*** : Cette méthode n'a pas besoin de tester les éventuelles collisions avec un fantôme !) (4 p.)

➢ FONCTIONS (4 p.)

(5) Ecrire la fonction `get_random_pos(maze)` qui choisit une cellule aléatoire du labyrinthe `maze` fourni en paramètre. La cellule choisie ne doit pas être du type `'wall'` et elle ne doit pas se trouver ni dans la 1re ligne, ni dans la 1re colonne du labyrinthe. La fonction retourne le tuplet de coordonnées `(x,y)` de la cellule choisie. (2,5 p.)

(6) Ecrire la fonction booléenne `still_some_food(maze)` qui vérifie s'il y a encore de la nourriture dans le labyrinthe `maze` ou non. (1,5 p.)

➢ POGRAMME PRINCIPAL (12 p.)

(7) Ecrire le programme principal :

a) Créer une instance `maze` du labyrinthe, une instance `pacman` du pacman et 4 fantômes du type `Ghost`, qui sont placés dans une liste `ghosts` : le pacman est positionné en `(0,0)`. La position initiale des fantômes est choisie au hasard à l'aide de la fonction `get_random_pos`. (2 p.)

b) Ecrire la boucle principale :
- Lorsque le joueur clique sur la croix de fermeture, le programme et l'environnement `pygame` sont quittés correctement.
- S'il y a encore de la nourriture, le pacman est déplacé à l'aide des flèches de direction du clavier. Aussi longtemps que le joueur maintient une flèche de direction enfoncée, le pacman doit continuer à se déplacer vers la cellule voisine visée.
- Le pacman doit interrompre sa marche dès qu'il heurte un fantôme. Le pacman et tous les fantômes sont déplacés une fois à chaque itération, à condition que le pacman est en vie et qu'il y a encore de la nourriture. Toutes les collisions du pacman avec l'un des fantômes doivent être détectées immédiatement : en cas de collision le pacman meurt. Aucun croisement du pacman avec les fantômes n'est permis.
- Tous les objets sont dessinés une fois à chaque itération.
- Le jeu est terminé si le pacman a mangé toutes les pac-gommes ou lorsqu'il meurt. Les fantômes arrêtent alors de bouger et le pacman ne répond plus aux flèches de direction. Dans le premier cas le message affiché en vert au centre de l'écran est : `'YOU WON! SCORE:...'`, sinon le message affiché en orange est `'YOU DIED! SCORE:...'`, les trois points dans ces strings devant être remplacés par le score du pacman. (10 p.)

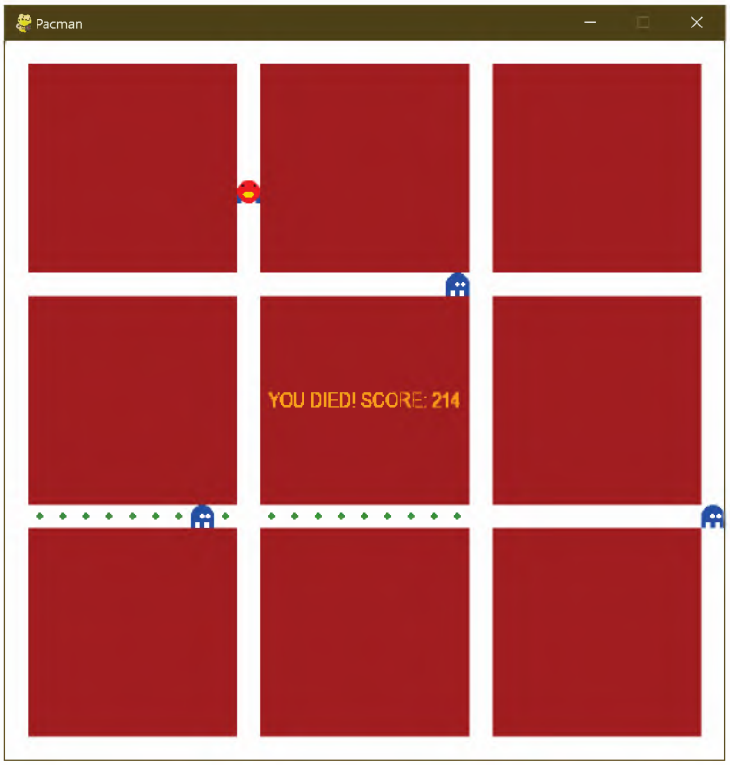

fig. 4 : Le pacman a heurté un fantôme, jeu perdu !

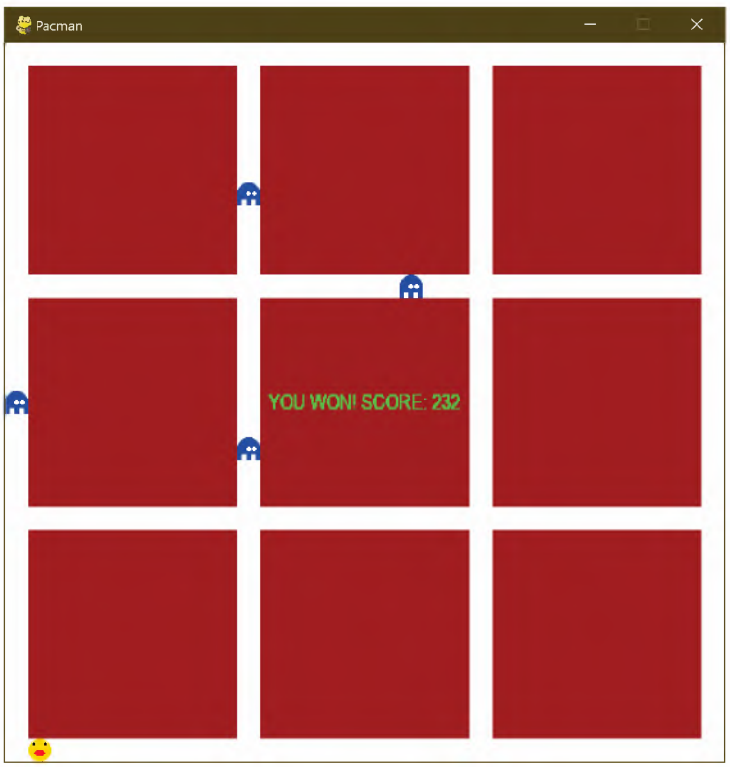

fig. 5 : Le pacman a mangé tout, jeu gagné !