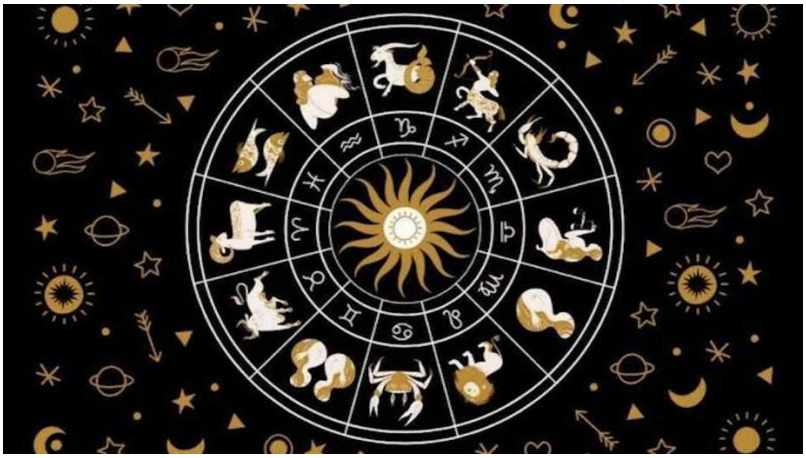
25 Juin 7486.

Le monde a connu de grandes avancées technologiques, mais la planète Terre voit ses jungles et déserts, ses vents et marées, ses lacs et océans, et sa lumière naturelle provenant du soleil, disparaître.

Les 12 signes du zodiaque, entités protégeant autrefois la Terre et dont chaque être humain est sous la protection, ont été capturés par les 4 éléments primordiaux venus se venger de l'humanité : l'air, le feu, la terre et l'eau.



C'est alors qu'une dernière entité, le Signe du Serpentaire, 13e signe du zodiaque, défie les 4 éléments afin de libérer les 12 autres Signes. Vous êtes choisi pour l'aider à faire face à cette menace. Serez-vous de taille ?

Partie 1 : Les préparatifs

Le Serpentaire vous donne 3 fichiers : Astroders.py, Enemies.py et Player.py.

Il a d'abord besoin que tu l'aides à rejoindre l'espace !

Fort heureusement, le Serpentaire t'a déjà préparé le terrain !

En effet, le fichier Astroders.py contient déjà du code.
Tu peux remarquer qu'une fenêtre s'est ouverte, et nous pouvons y voir le Serpentaire.
Prête bien attention aux commentaires soigneusement laissés à l'intérieur.
Ils sont repérables via le caractère '#'.
Ceux-ci expliquent en majorité le fonctionnement du code.
Si tu ne comprends pas certaines choses, n'hésite surtout pas à demander de l'aide aux autres participants ou aux cobras !
Pour le moment, sont contenu permet de créer une fenêtre, de créer le joueur, de pouvoir arrêter le programme avec la touche « Echap », et enfin d'afficher cette fenêtre avec une image en fond, et le Serpentaire en personne !

Pour tester cela, effectue la commande « python3 Astroders.py » pour démarrer le programme.
Tu devras utiliser cette commande dès que tu voudras le relancer.

Cependant, ce n'est pas en restant immobile que le Serpentaire réussira à sauver le monde…

Tu as dû remarquer que des ennemis ont été créés d'après le code, mais ils n'apparaissent pas.
En effet, la fonction « update » du joueur a besoin de ceux-ci pour fonctionner, mais nous verrons ça plus précisément tout à l'heure !

Il nous faut tout d'abord permettre au Serpentaire de bouger.
Pour cela, il faut aller dans le fichier « Player.py ».
La fonction « update » permet de réunir tout ce dont nous avons besoin pour afficher le Serpentaire et interagir avec lui.

Pour le moment, elle est malheureusement vide…
Mais pas d'inquiétude ! Le Serpentaire vous indique que Pygame vous permet une multitude de merveilleuses choses !
En effet, Pygame vous permet de détecter les entrées du clavier, dont les flèches directionnelles.
Pour vous aider, le Serpentaire vous indique ceci :
« La ligne « if pygame.key.get_pressed()[K_RIGHT] » te permet de vérifier si la flèche droite du clavier est appuyée.
Pour me déplacer, tu peux modifier ma position 'x', en l'augmentant selon ma vitesse.
Pour cela, la ligne « self.x += self.player_speed » te sera utile ! »
Après avoir suivi ses conseils et ajouté ces lignes dans « update », tu peux maintenant relancer le programme, et appuyer sur la flèche de droite. Le Serpentaire devrait maintenant bouger !
Cependant, il peut sortir de l'écran…
Pour cela, faites une condition lui permettant de ne se déplacer à droite seulement si sa position 'x' est inférieure ou égale à 1100 ! Cela devrait être un jeu d'enfant.
Maintenant, faites en sorte de pouvoir le déplacer vers la gauche, et de haut en bas (ces deux derniers étant évidemment en position 'y') !
Il vous indique que les valeurs pour qu'il ne déborde pas de l'écran vers la gauche, le bas et le haut sont respectivement 0, 800 et 0.
Il est à noter que les positions 'y' s'incrémentent vers le bas, et les 'x' vers la droite.

Partie 2 : Des caducées en folie

Bravo ! Le Serpentaire peut maintenant se déplacer librement dans la fenêtre.
Mais même s'il est assez fort, il ne pourra pas vaincre les éléments avec la seule force de ses bras…
Pour cela, nous allons l'aider à envoyer des caducées, son symbole associés !

Tu as peut-être remarqué que certaines choses sont présentes dans « __init__ », dont « bullets », projectiles, en français.

Le Serpentaire vous donne ce morceau de savoir, toujours dans la fonction « update » :

```
time_now = time.time() * 1000 # création de chrono
if time_now - self.clock > self.shooting_speed * 1000: # a-t-il atteint 1 seconde ?
    self.bullets.append([self.bullet, self.x, self.y]) # j'ajoute un caducée
    self.clock = time_now # je relance le chrono à zéro
for bullet in range(len(self.bullets)): # pour tous mes projectiles
    self.bullets[bullet][2] -= self.bullet_speed # je les déplace en 'y'
```

Il vous indique comment cela fonctionne :

« Je crée un chrono, puis je vérifie si ce dernier atteint ma cadence de tir.
Si tel est le cas, je m'ajoute un nouveau caducée, et je relance le chrono à zéro.
Enfin, je déplace tous les caducées que je possède selon la vitesse de ceux-ci. »

Cela étant fait, il faut maintenant qu'ils puissent apparaître à l'écran.
C'est dans la fonction « display » que se font les affichages relatifs au Serpentaire.
Tu as peut-être déjà remarqué qu'une ligne est déjà présente : celle permettant d'afficher le Serpentaire !
En suivant le même principe que précédemment pour afficher tous les projectiles, voici comment afficher ses projectiles :

```
for bullet in self.bullets:
    screen.blit(bullet[0], (bullet[1], bullet[2]))  # bullet[1] est la position
'x', bullet[1] la position 'y'
```

Vous pouvez maintenant démarrer à nouveau le programme, et des caducées devraient être lancés toutes les secondes par votre signe protecteur !
Une petite chose est à rajouter par soucis de propreté.
En effet, les projectiles ne sont pas détruits ! On peut vite imaginer qu'au bout d'un certain temps une multitude de caducées flotteraient infiniment dans le vide spatial, mais surtout hors de la fenêtre, tout en étant calculés par l'ordinateur, pouvant entraîner de la latence…
C'est à nouveau le retour d'un morceau de savoir !

```
# permet de détuire les projectiles en dehors de l'écran, en re-stockant les
projectiles "vivants" en
# ayant retiré les projectiles "morts"
alive_bullets = []
for bullet in range(len(self.bullets)):
    if self.bullets[bullet][2] > -50: # le caducée est hors de l'écran de 50 pixels
        alive_bullets.append(self.bullets[bullet])
self.bullets = alive_bullets
```

Place-le donc dans « update ».

Partie 3 : L'invasion commence

Bien, notre signe préféré peut maintenant tirer !
Mais il serait maintenant temps de rattraper les 4 éléments.
C'est maintenant que le fichier « Enemies.py » intervient.

Procédons étape par étape !
En premier lieu, tu vas devoir afficher les éléments.
Contrairement à ce que l'on pourrait penser, ils ne sont pas au nombre de 4.
Enfin, si, mais pas exactement…
Chaque élément a le pouvoir de se diviser ! Ils apparaissent alors sous forme de « vagues d'ennemis ».
(tiens, cela me rappelle un certain jeu d'arcade de 1978…)

Regarde d'abord le contenu de « select_image».
Il contient le nom des différentes images utilisées pour afficher les éléments !
Cette fonction va nous permettre de retourner une image, le but étant que cette dernière soit aléatoire. Autrement dit, l'apparition des éléments se fait aléatoirement dans la vague.

Tu peux remarquer que la fonction prend en argument, en plus du « self » qui permet t'interagir avec les données de « __init__ », un 'x' et un 'y', qui sont donc des coordonnées.
Cette fonction sera utilisée plus tard pour créer les éléments à des emplacements précis pour les ordonnées en vague.
Nous pouvons déduire cela par le « return [img, x, y] ».
x et y étant donnés en argument, nous allons devoir créer img !
Pour cela, nous avons besoin d'une fonction nous permettant de retourner un chiffre alétoire.
Par chance, python possède de base une telle fonction : random !
Pour t'aider, le Serpentaire t'indique ceci :
« Tu as besoin de récupérer un chiffre entier, pour cela, tu dois utiliser random.randint(chiffre_minimum, chiffre_maximum). »
Dans « __init__ », tu peux déjà remarquer comment est chargée une image.
Ainsi, tu peux maintenant compléter cette ligne permettant de sélectionner une image aléatoirement, à ajouter juste après l'image_id :

img = pygame.image.load(image_id["Remplace-moi par la fonction random"])

Il faut maintenant ajuster la taille des images de 50 par 50.
Tu as également accès à cela dans « __init__ », et tu n'as plus qu'à appliquer la fonction utilisée pour cela à img !

Maintenant, plaçons les ennemis.
Cela se passe dans « create_enemies ».
Le Serpentaire t'indique ceci :
« Les 4 éléments apparaissent en formant 3 lignes de 10 éléments. Chaque élément est séparé de 100 pixels de haut et de large.».
Avec ces informations, complète le code déjà présent !
Oh, et afin de faire réapparaître une vague d'élément lorsque la précédente a entièrement disparue, ce qui servira pour plus tard mais surtout pour leur première apparition, ajoute ceci dans « update » :

```
if len(self.enemies) == 0:
    self.create_enemies()
```

Retire les commentaires dans le fichier « Astroders.py » lignes 34 et 39 afin de pouvoir utiliser les fonctions update et display.
Cela étant fait, relance le programme, et les éléments devraient apparaître !
Cependant, ils sont invincibles ! Mais heureusement, ils ne bougent pas et ne sont pas agressifs…

Partie 4 : Des ennemis envahissant

Il est maintenant temps de faire bouger les éléments !
Il paraît qu'ils ont l'habitude de se déplacer vers la droite jusqu'à un certain endroit, de descendre, puis de se déplacer vers la gauche jusqu'à un certain endroit, de descendre, et ainsi de suite…
(tiens, cela me rappelle toujours ce même jeu…)

Pour se faire, va dans la fonction « update ».
Dans « __init__ », tu peux remarquer les variables « curr_move » et « curr_dir ».
La première servira de compteur pour les mouvements, et la deuxième permettra de bouger vers la gauche ou vers la droite.
Le Serpentaire, comme à son habitude, vous indique :

« A chaque fois que la fonction est appelée, curr_move est incrémentée de 1.
Si curr_move est supérieur à nb_moves, alors la valeur de curr_dir passe de 1 à -1, ou de -1 à 1 selon, et curr_move repart de 0.
C'est ensuite que la position 'y' de tous les ennemis augmente de 10.
En parallèle, à chaque fois que la fonction est appelée, elle augmente la position x de tous les éléments par curr_dir, permettant alors de les faire passer de -1 ou de +1 et les faisant bouger vers la gauche ou vers la droite.»

Avec son aide, complète le code présent. Oh, et n'oublie pas de retirer les commentaires afin de permettre au programme de prendre en compte le code !

Maintenant, les ennemis devraient se déplacer !

Partie 5 : La guerre des astres

Bravo, tu es presque arrivé à la fin !
Le Serpentaire tire, mais pas les éléments. Et puis, tout le monde est invincible…
Il serait temps de régler tout ça !

Pour commencer, nous allons donner l'avantage au Serpentaire (après tout, tu es son chouchou), et nous commencerons par lui permettre de détruire les éléments à l'aide de ses caducées (ce qui est bien étrange après réflexion).

Pour ce faire, reste dans le fichier « Ennemies.py ».
La fonction « bullet_collide » permet de savoir si un élément de la vague se fait toucher par un caducée. La fonction est déjà prête, hormis les valeurs !
Les valeurs manquantes constituent ce qui s'appelle une « hitbox », que l'on pourrait traduire par « zone de touche » en français. Pour faire simple, c'est ce qui permet de savoir si une collision a lieu entre un caducée et un élément.
Tu peux décider par toi-même de créer ta propre hitbox, mais le Serpentaire t'a tout de même laissé ces valeurs comme exemple ou point de repère : 25, 75, 75, 100.

Maintenant, tu peux retourner dans le fichier « Player.py ».
Tu as sûrement dû remarquer que la fonction « bullet_collide » renvoie ce qui s'appelle une valeur booléenne, soit une valeur « Vraie ou Fausse » (True ou False).
En effet, elle permet tout simplement de dire si un élément s'est fait toucher ou non.
Naturellement, une condition 'if' vérifie si une condition est vraie.
Si l'on place une fonction retournant une valeur booléenne, si celle-ci est vraie, alors la condition sera remplie.

Ainsi, ceci permet de supprimer le projectile du joueur s'il touche un ennemi, tandis que du côté de l'ennemi, lorsqu'il renvoi « True », il aura de son côté supprimé un élément.
Ajoute ceci dans « update » :

```
# gère les projectiles du joueur
for bullet in range(len(self.bullets)):
    self.bullets[bullet][2] -= self.bullet_speed  # déplace le projectile
    if enemies.bullet_collide(self.bullets[bullet][1], self.bullets[bullet][2]):
# supprime le projectile
        # s'il touche un ennemi
        del self.bullets[bullet]
        break
```

Tu peux relancer le programme.
Normalement, le Serpentaire devrait pouvoir éliminer les éléments !

C'est superbe, mais dans la réalité, les éléments ne sont pas si inoffensifs.
Ils envoient des étoiles, et parfois des météorites !

Retourne dans « Ennemies.py ».
Pour te faciliter la tâche, les fonctions « shot » et « update_shot » sont déjà présentes, il te suffit de supprimer les commentaires de ces fonctions, dans « create_ennemies » afin de leur associer les chronos et aux lignes 91 et 92 afin de les utiliser dans « update », ainsi que les commentaires dans « bullet_collide », permettant de supprimer et d'arrêter les ces mêmes chronos à la destruction de l'élément.
En échange, lis-bien ce que font ces fonctions à l'aide des commentaires !

Relance le programme, et les éléments devraient maintenant lancer des projectiles.
Mais, le Serpentaire est invincible !
Toujours dans notre réalité, les éléments sont censés faire mal…

Partie 6 : Le dernier rempart de l'humanité

Ça y est, tu es arrivé à la phase finale !

Il faut maintenant blesser notre adoré signe du Serpentaire, sans quoi le combat n'aurait aucune intensité…

Pour cela, toujours dans « Enemies.py », regarde la fonction « player_collision ».
Tu vas devoir la compléter !
Cette fois-ci, tu ne devrais pas avoir trop de mal à le faire.
Il te suffit de recréer une hitbox.
Voici quelques valeurs de base si tu le souhaites : 50 pour le joueur et les météores, et 50 pour la largeur des étoiles et 30 pour sa longueur.

Maintenant que tu as pris en compte les collisions, il faut qu'elles aient un impact (sans mauvais jeu de mots). Pour cela, retourne dans « Player.py ».

Ton dernier défi consiste à utiliser le résultat de « player_collision » pour réduire les points de vie du Serpentaire.
Comme tu as dû le voir dans cette dernière fonction, les étoiles infligent 1 point de dégât et entrer en collision avec les éléments te tue en un coup (en retournant -1), et une collision avec une météorite réduit de 3 tes points vies, qui sont à la base… de 3. Le programme devrait donc s'arrêter si le Serpentaire subit suffisamment de dégâts.

Pour finir, affichons les points de vies du meilleur signe du zodiaque pour faciliter le gameplay !
Pour cela, ajoute ce code dans « update » :

```
# crée du texte à partir du montant de point de vie du joueur
self.hp_text = "HP : " + str(self.hp)
```

Pour afficher le texte, ajoute ceci dans « display » :

```
white = (255, 255, 255) # la couleur blanche au format RGB
text_img = self.font.render(self.hp_text, True, white) # affiche le texte
screen.blit(text_img, (0, 0))
```

Relance le programme, et tu devrais voir les PV du Serpentaire s'afficher !

Partie 7 ??? : A toi de jouer !

Tout d'abord : bravo !
Tu as permis au Serpentaire de pouvoir défendre l'humanité.
Maintenant, libre à toi de modifier le jeu selon tes désirs !
Tu as normalement dû apprendre certaines choses, et tu peux modifier à ta guise des parties du programme ou ajouter de nouvelles choses facilement comme du texte, de nouveaux projectiles, etc…
Pourquoi ne pas voter à la fin de la journée pour le meilleur jeu ?

Voici quelques exemples d'ajouts sympathiques ! :

Faire en sorte que chaque élément ajoute un bonus selon son type, comme plus de PV, augmentation de la vitesse de déplacement, augmentation de la vitesse de tir, et un bonus aléatoire entre ces trois-là, et augmenter la puissance des éléments à chaque vague (plus de chances de faire apparaître des météores, et une cadence de tir améliorée).

Une condition de défaite si un élément atteint le bas de l'écran.

Un score et un écran de Game Over et / ou une possibilité de victoire, ou bien un menu, etc…

De nouveaux ennemis ou personnages jouables.

N'hésite surtout pas à demander de l'aide à un Cobra si tu as du mal à appliquer certaines de tes idées !

Maintenant, c'est à toi de décider du destin de l'humanité et de celui du signe du Serpentaire !