SPACE INVADER

Le jeu demande d'installer une bibliothèque absente des distributions normales de Python (idle) : pygame ; avec cette bibliothèque, on s'écarte des fonctionnements basiques de ce langage. Cela devient plus difficile pour tous, professeur compris, de comprendre le fonctionnement du programme, ce qui pénalise un peu le projet.
Ceci dit le jeu fonctionne bien, et il offre un visuel et une dynamique intéressante. D'après le compte-rendu, on comprend qu'il y a eu un grand investissement des deux auteurs pour obtenir ce résultat et la fierté qu'ils ressentent d'avoir dépassé les multiples difficultés qui se sont présentées à eux est une première récompense, sans doute la meilleure.

De multiples fonctions de pygame sont utilisées que l'on ne cherchera pas à comprendre ici.
Pour gérer le temps, par exemple, il y a clock = pygame.time.Clock() mais une autre fonction est plus massivement utilisée, par exemple dans prochain_missile = time.time() + 1.5 où time.time() est une fonction du module time (un de ceux dont on a parlé en cours).
Les boucles for sont massivement utilisées dans tout le programme (52 en tout) ; elles servent à boucler sur des listes qui sont également très nombreuses. Le programme lui-même est long (833 lignes) et gagnerait à être raccourci.
On pourrait notablement simplifier le programme en utilisant des listes à deux indices :
au lieu de tirs, tirs2, tirs3, tirs4, tirs5, tirs6, utiliser tirs[i] avec i allant de 0 à 5.
De même, utiliser des listes pour les variables qui sont déclinées pour tous les ennemis :
compteur_bas, compteur_bas2, compteur_bas3, compteur_bas4, etc.
prochain_tir_ennemi, prochain_tir_ennemi2, prochain_tir_ennemi3, etc.
collision, collision2, collision3, collision4, collision5, etc.
collision_tir_ennemi2 collision_tir_ennemi2, collision_tir_ennemi3, etc.
compteur_cote, compteur_cote2, compteur_cote3, etc.
On peut aussi utiliser des listes pour les objets pygame définis :
ennemis, ennemis2, ennemis3, ennemis4, ennemis5, ennemis6, etc.
mort_ennemi, mort_ennemi2, mort_ennemi3, mort_ennemi4, , etc.
veritable_position_tir_ennemi, veritable_position_tir_ennemi2, etc.
Finalement, les principes du fonctionnement ne sont pas si compliqués, mais ils le paraissent du fait de la multiplicité et de la longueur.

Des commentaires sont donnés mais sont très limités, comme des titres aux différentes grandes sections du programme. Mais la plupart des actions ne sont pas commentées. Il faut deviner à quoi servent les différentes parties. Quel est le rôle, par exemple, de cette partie :

```
if event.type==KEYUP:
    if event.key==K_p:
        pygame.mixer.music.pause()
        pause = True
```

Sans doute est-ce pour mettre sur pause le jeu, mais cela pourrait être écrit dans le programme et aussi, éventuellement, dans un récapitulatif des différentes possibilités. Quelle est cette touche (K_p) : la flèche vers le haut ? C'est toujours bien d'indiquer ces informations dans le programme car même les auteurs risquent de les avoir oubliées après quelques temps…

Projet de note : 18/20
Détail de la note :

- préprojet : 2/2
- compte-rendu : 4/4 (narration intéressante des difficultés globalement rencontrées et des solutions adoptées, détails de deux difficultés majeures : déplacement des ennemis et réglage des collisions)

- Programme : 12/14 (fonctionnel, assez complexe notamment par la multiplication des objets et l'utilisation d'une bibliothèque spécifique, peu documenté)