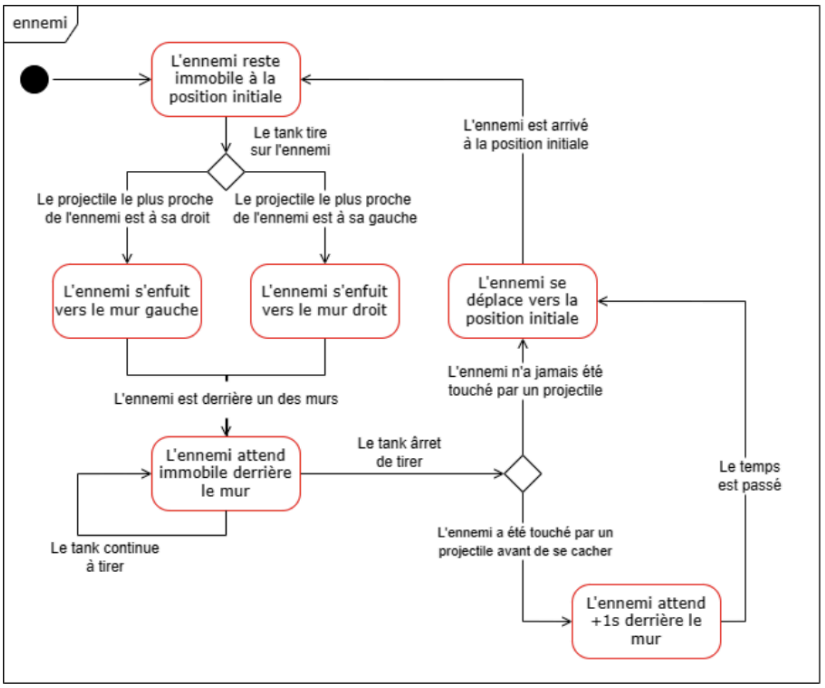
RAPPORT TP4

github : https://github.com/M3K0PZ/LOG725-TP

But : implémenter l'automate ci-dessous pour l'ennemi



Pour faire cet automate j'ai fait un système de SFM (Final state machine) vu lors d'un cours de programmation d'ai à Polytechnique Montréal.

Une classe machine à états s'occupe de mettre à jour les états, elle s'assure également qu'un seul état soit actif à la fois.
Les états sont des classes eux aussi, ils ont trois fontcions principales :

- enter : fonction qui joue quand on rentre dans l'état une fois
- update : fonction qui effectue le role de l'ia (chaque tick)
- check_conditions : qui vérifie si on a les conditions de sortie de l'état, et qui signale à la state machine de changer d'état

Z_StateTemplate_DO_NOT_USE.py

fichier template qui montre comment un état est à la base

Pourquoi :
une state machine permet l'ajout d'autant d'états qu'on veut, et de les gérer de manière indépendante, c'est plus modulable, clair et facile à implémenter pour de gros jeux.

Rajouté en plus des consignes :

- Un out of bounds pour les sprites ennemis au cas ou on transitionne mal

- Le fait que quand on fuie, si il y a plus de balle, on fuit toujours mais on attend la prochaine balle pour décider ou (plutot logique vu la lenteur des balles)

- La position initiale n'est pas vraiment au centre de l'écran pour l'ennemi (quand on se remet au centre on est décalé d'un offset par rapport au spawn)

- Chaque update connaît bullets, on aurait pu prendre le problème à l'envers, et lui mettre une valeur par défaut à None, mais normaliser les appels, avec des paramètres essentiels comme ai et bullets semble plus logique, car plus simple

- Utilisé pygame.time.wait et pas delay, car delay prive le processeur d'utiliser la ressource (effectue une interruption du thread) et wait laisse d'autres programmes utiliser la ressource pour d'autres besoins en attendant (plus performant, un peu moins précis, mais pour ce besoin c'est pas utile d'être précis à la ms près)