Kenzo Carneiro

**Pygame**

Voici le début de la fonction main de mon projet de groupe de terminale :



Pygame est un module assez simple, il est divisé en plusieurs modules chacun ayant un rôle précis : display pour la fenêtre, mixer pour l’audio, image pour la gestion d’image, etc etc.

Ci-dessus, on voit un peu de syntaxe pygame. Tout d’abord, il faut initialiser l’affichage, rien de plus simple ; hauteur et largeur. On définit ensuite le titre de la fenêtre, et on initialise la clock (voir plus loin). On initialise aussi le mixer, c’est-à-dire le gestionnaire de son sur Pygame (on fera ça tard dans le dev de Age of, c’est pas très important).

Remarque inutile : Oui notre projet s’appelais Pythontale, c’était un délire en gros on créait un fangame Undertale sauf que l’histoire (si on avait eu le temps de la développer lol) devait tourner autour de… et bas de Python en fait. C’est pour cela que le seul et unique ennemi du jeu s’appelle « Conwayboss », c’est un boss qui suit les patterns du jeu de la vie de Conway.

pygame.image permet de charger des images, on les convertit ensuite en format pygame, et puis plus tard, il pourrait être nécessaire de convertir ses images en « rectangles » :



Cela permet, à partir des dimensions de bases des images, de créer une hitbox simple selon ces dimensions. En gros, la hitbox = l’image entière. En effet, des fonctions permettent ensuite de déterminer si 2 rectangles se superposent, grâce à ça, on en déduit si il y a collision ou non.

A noter que l’interdiction de déplacement en cas de collision avec un mur est parfois délicate à gérer (c’est pour cela qu’il y a beaucoup de jeu dans lesquels il existe des bugs où on peut passer à travers les murs). Dans notre projet, le système était simple : on déplace le joueur, si collision il y a on replace le joueur sur le bord du mur à chaque fois. Ce système peut poser des problèmes en fct° de l’épaisseur du mur et de la vitesse du joueur. Pour age of, ça ne sera pas un prob car ttes les unités iront à peu près à la même vitesse.

Pygame, comme la plupart des moteurs de jeu, fonctionne avec un système simple de « boucle général ». C’est la boucle qui EN GROS est répété à chaque tick (à peu près équivalent aux FPS), et qui permet au jeu de tourner en continu :



Dans cette boucle, pas mal de choses se passent :

* On s’occupe de l’affichage, fill permet de remplir l’affichage d’une couleur et **blit** permet de dessiner des sprites (image d’objets avec lesquels on intéragit : le joueur, PNJ, batiments, etc) à l’écran.
* Ensuite, on regarde si un « event » a lieu, par exemple : appuie sur une touche du clavier, appuie sur la croix rouge pour quitter l’application, etc.
* (tplist est spécifique à notre projet, il permet de changer de zone dans le jeu, c’est pas important pour nous.)
* Les deux dernières lignes sont très importantes : sans elles, le jeu ne se lance même pas. pygame.display.update() met à jour l’affichage, c’est-à-dire qu’il AFFICHE le jeu. Sans cette commande l’affichage est mis à jour dans le code, mais n’est jamais réellement affiché sur l’écran, l’utilisateur ne verra donc strictement rien si à chaque itération on ne modifie pas ça.
* La dernière ligne, clock.tick(60), définit le nombre de tics (équivaut en gros au FPS) du jeu. Attention cependant, augmenter les tics ne donnera pas plus d’FPS au jeu, enfin si, mais il accélèrera surtout l’entièreté du jeu !!! Ce n’est donc pas exactement comme les FPS.