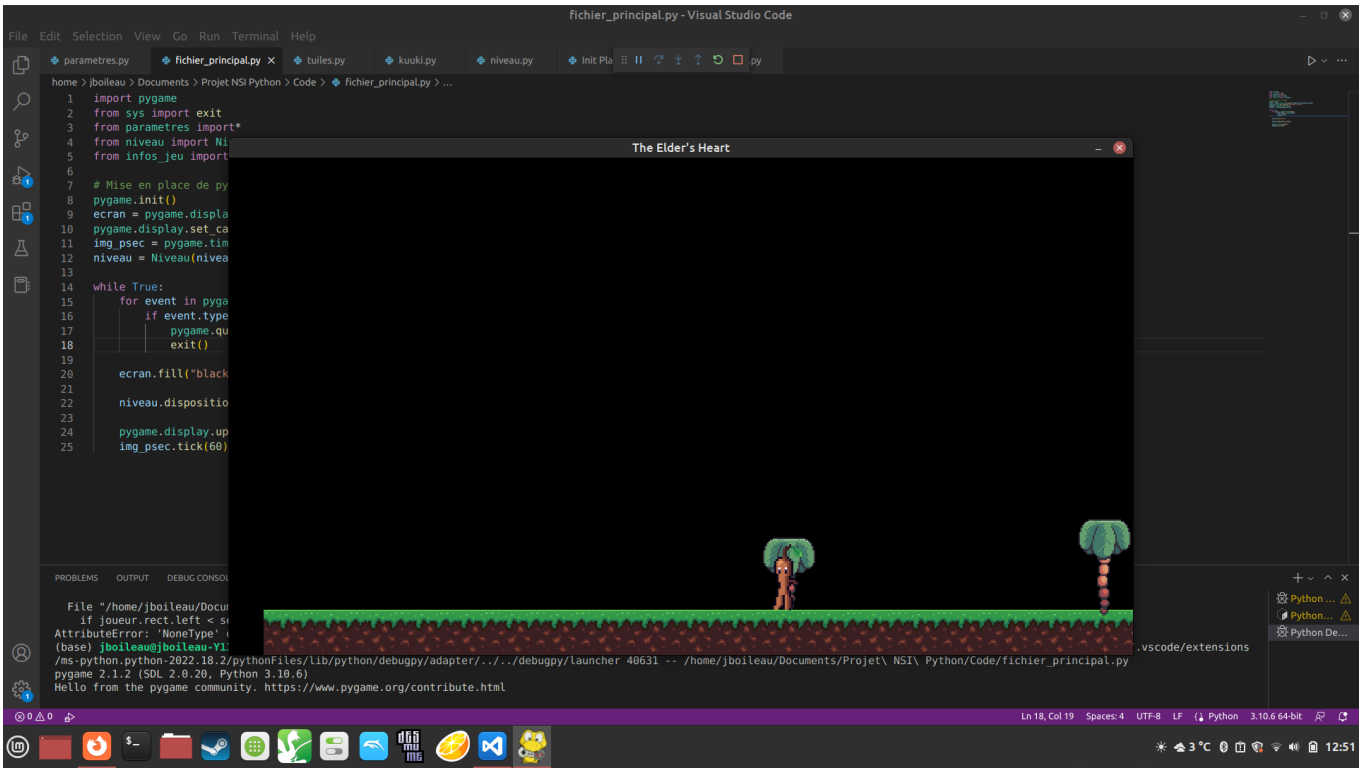
Compte-rendu de projet : difficultés rencontrées, résolutions, conclusion sur les apports de ce travail et rapide guide de compréhension de Tiled

Organisation du dossier de projet :

- Un dossier “Codes” avec l’ensemble des fichiers Python ainsi que le dossier pycache qui permet (normalement) d’importer les fichiers librement au sein du dossier.
- Un dossier “Niveaux” avec l’ensemble des tilesets et un dossier de fichiers csv et contenant aussi un fichier Tiled, ceci pour chaque niveau.
- Un dossier “Police” contenant un seul fichier, la police que j’utilise tout le temps.
- Un dossier “Visuels”, de loin le plus gros. Il est séparé en dossiers pour chaque type de texture. Tous les visuels ont été créés par moi seul sur le site en ligne Pixilart.

  Pour lancer le jeu, il faut lancer le programme du fichier nommé fichier_principal.

Quelques problèmes rencontrés (ceux qui m’ont demandé le plus de temps à résoudre)



- Lors du déplacement du rectangle de la caméra, seules les images des sprites bougent et non leur rectangle, ce qui donne lieu à des situations comme ci-dessus.
- Lorsque le joueur atteint les limites du rectangle de la caméra, sa vitesse semble augmenter et très vite puis il disparaît de l'écran.

Solution : les deux problèmes ont été réglés d'un coup et il semble qu'ils étaient liés. Après au moins deux heures de comparaison minutieuse entre le code qui ne fonctionnait pas

correctement et celui dont j'avais tiré la logique de la caméra avec rectangle, je me suis rendu compte que je n'avais pas placé le sprite du joueur dans le GroupeCamera avec tous les autres sprites. En quoi cela pose problème ? Le joueur est le seul à ne pas être replacé en arrière lorsqu'il dépasse les limites du rectangle de la caméra. Conséquences : il n'est pas ramené en arrière lorsque tous les autres sprites sont déplacés dans le sens contraire de son déplacement (et donc il dépasse très vite la partie visible du niveau, c'est comme s'il allait deux fois plus vite), et deuxièmement, je ne comprends pas tout à fait pourquoi, mais alors que les images des autres sprites étaient déplacées, leur rectangle correspondant ne bougeait pas sur l'écran (ils restait au même emplacement). Ce problème était consécutif à la confusion entre deux logiques de caméra différentes. Une partie de chacune était implémentée dans le code, ce qui empêchait le bon fonctionnement de la caméra dont je voulais me servir.

Autre problème mineur rencontré plus tard : les animations ne se suivaient pas dans le bon ordre : la fonction importation_dossier() renvoyait une liste désordonnée. Un retour aux sources avec les fonctions de base sur les listes s'est alors imposé : sort() ...

Un problème au niveau de l'arrière-plan s'est dévoilé lorsque j'ai tenté de blit une image dans le fond : tous les éléments sur l'écran gardaient non seulement leur image présente mais aussi les images des anciennes positions lorsqu'ils se déplaçaient. Je ne sais pas si c'est la meilleure solution, mais en remplissant d'abord l'écran de noir avant de blit l'image de fond, cela a réglé le problème.

Un testeur en bêta (Soren) m'a dit qu'il serait judicieux d'ajouter une accélération pour éviter de tomber sur des pics lorsqu'on hésite à sauter et qu'on revient légèrement en arrière pour prendre de l'élan, ou même simplement pour que le déplacement du personnage paraisse plus naturel. C'était très facile à rajouter : je l'ai fait. Puis étant dans ma lancée, j'ai décidé de programmer le saut double, qui semblait bien plus simple à implémenter que le dash ou le saut mural (le dash m'a semblé peut-être faisable, j'ai essayé, mais il est très compliqué à rendre vraiment naturel et le saut mural je n'en parle même pas). Seulement, j'ai vite rencontré deux problèmes majeurs : le double saut n'est pas actif lorsque l'on tombe d'un rebord sans sauter, et il ne l'est pas non plus lorsque l'on tient la barre espace enfoncée juste au-delà de la limite de hauteur.
J'ai résolu le problème en l'attaquant autrement. J'ai changé le système de modification du vecteur de déplacement (particulièrement la place de celle-ci dans le programme) notamment. Un seul problème subsistait : si je sautais le maximum et que je gardais la barre espace enfoncée en arrivant au sol, je ne pouvais plus sauter. C'était dû à un booléen en trop dans la condition pour remettre la valeur initiale de la vitesse du saut (et donc permettre de faire un nouveau saut).
D'autres problèmes concernant le double saut ont suivi (saut automatique quand on garde la barre espace enfoncée, ce que je ne voulais pas, fonctionnement de la feuille enchantée, etc.), que j'ai réglés en changeant plusieurs conditions dans les instructions conditionnelles.

J'ai un problème mineur mais qui peut s'avérer embêtant si l'on se retrouve à l'exacte mauvaise position. Lorsque le bas droit ou le bas gauche rectangle du joueur se trouvent à la limite haut gauche ou la limite haut droit d'un autre sprite, l'état du joueur alterne rapidement entre sur

place et chute. Il faut que je voie s'il n'y a pas moyen de régler ce problème quelque part dans le code du joueur.
J'ai réglé le problème en créant un rectangle supplémentaire pour le joueur qui sert à gérer toutes les collisions. Le bémol est que les collisions verticales de chute sont toujours généreuses. Malheureusement, j'ai pensé que cela aurait nécessité une complète révision du système de collision, ce que je n'ai pas eu le temps de faire.
L'autre avantage de cette technique est que les collisions sont tout de même plus réalistes et cohérentes par rapport au personnage (pas de collision sur la tige de sa tête par exemple, ni à 20 cm de son corps).

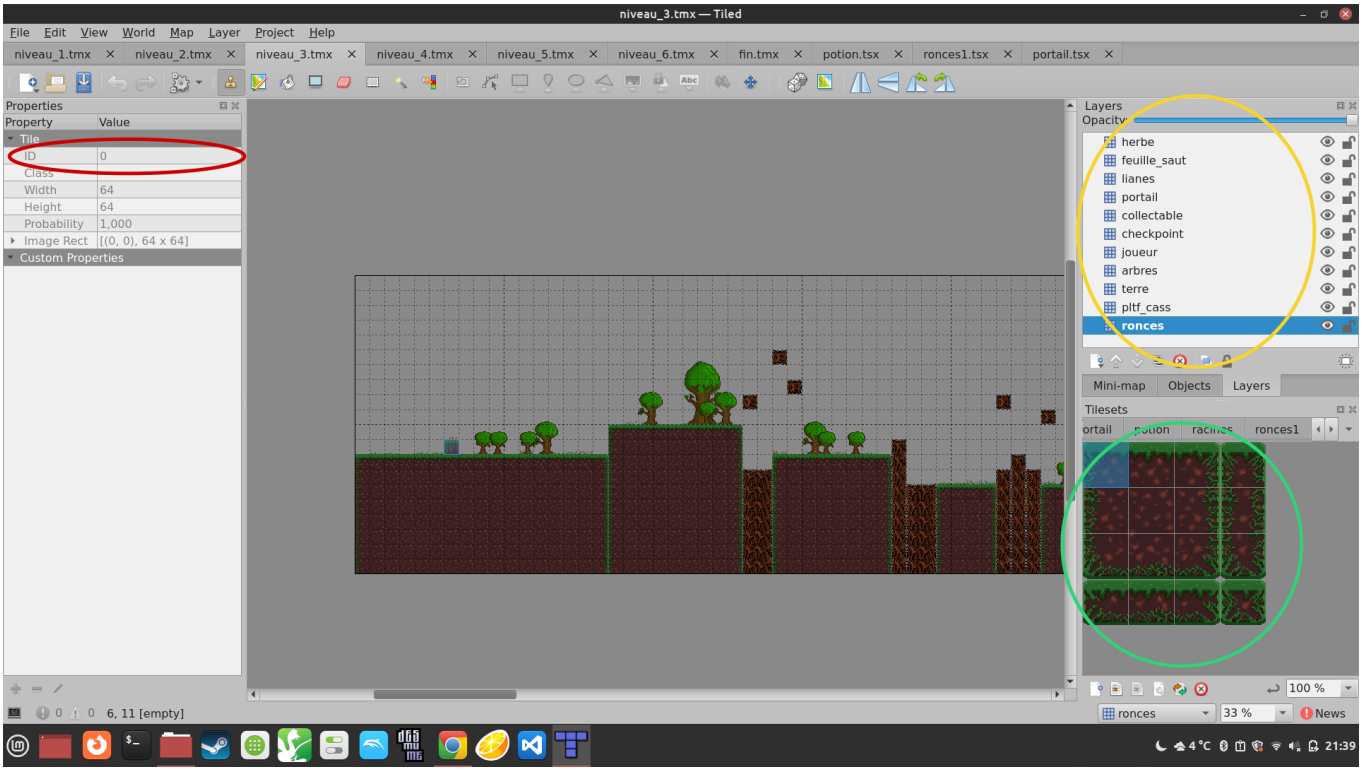
J'avais aussi de grosses pertes de frames lorsque j'affichais les rectangles transparents. C'est là que la méthode convert_alpha() s'est révélée indispensable car réduisant considérablement l'utilisation des performances.

Enfin, ce qui m'a pris largement le plus de temps (un week-end entier, le dernier), c'est de réviser tout mon code pour le simplifier et le commenter de manière à ce qu'il puisse être compris.

Apports :

J'ai travaillé uniquement seul, avec peu de recours à des aides extérieures hormis pour apprendre les bases de création de niveau et de logique d'un platformer. Ainsi, j'ai pu découvrir ce que c'était de réaliser un projet de A à Z par moi-même. Le fait de créer des mécaniques tout seul m'a permis de découvrir le processus de réflexion qui précède le développement puis l'aboutissement de celles-là. Malgré quelques cheveux arrachés çà et là, j'ai pris beaucoup de plaisir (et de temps) à créer des visuels et m'en servir pour créer quelque chose de joli, fonctionnel, et souvent interactif avec le joueur. C'est d'ailleurs cela le plus intéressant et important dans la programmation d'un jeu à mon avis : inventer, créer et gérer les interactions joueur-objet.
Sûrement, ce projet était un peu ambitieux et comporte quelques bugs (normalement mineurs), et il m'a pris beaucoup plus de temps que je n'aurai voulu au point d'empiéter quelque peu sur les autres matières à certains moments, mais je n'ai pas les mots pour exprimer à quel point je suis heureux d'avoir accompli ce travail.
Aussi, peut-être inutile de le préciser, mais j'ai eu besoin de beaucoup plus de fonctions que ce que j'avais annoncé dans mon document de présentation…

Rapide guide de compréhension de Tiled :



- En rouge : l'ID de la tuile utilisée. C'est le nombre qui sera introduit dans le fichier csv à la place où la tuile a été posée dans l'éditeur de niveau (au centre).
- En jaune : l'ensemble des différentes couches de textures. Chaque couche comporte des images placées à certaines positions sur l'éditeur de niveau. Cette position détermine ensuite la ligne et la colonne où la valeur de l'ID de l'image sera placée. Chaque couche donne son nom à un fichier csv. Pour le niveau 3, il y a donc onze fichiers csv.
- En vert : les "tilesets". Ce sont les groupements d'images que je peux placer aux positions que je veux sur l'éditeur de niveau.

L'avantage indéniable de Tiled, c'est que c'est une manière très visuelle et donc très simple et efficace de créer des fichiers csv pour s'en servir dans des jeux. En quelques commandes de clavier, on peut exporter les fichiers et les utiliser directement avec le code approprié. Le logiciel comporte de multiples outils extrêmement utiles, dont je ne connais qu'une petite partie.