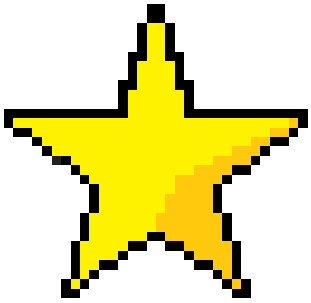
Dustopia - Compte rendu 5

Lysandre TERRISSE, Elora ROGER, Anouk OMS, Thomas DAILLY

Séance du 27 mars 2023



Oui, les traits que fait Paint ne sont pas symétriques.

Table des matières

1 Introduction

Cette séance a été en grande majorité dédiée à la mise en commun des codes. Malgré le fait qu'il y a peu à dire là dessus, nous pouvons quand même présenter d'autres fonctionnalités que nous avons ajouté. Nous avons principalement rajouté le déplacement des monstres, la transition entre les niveaux et des boutons interactifs pour différents menus.

2 Création des boutons

Pour le menu de Game Over, nous avons crée une classe `Bouton`, qui sera par ailleurs utilisée dans d'autres sections du jeu. Pour faire ces boutons, nous avons importé une nouvelle police d'écriture `Kanit-Regular.ttf`. L'un des problèmes que nous avons eu lors de l'implémentation de cette fonctionnalité était de donner une bonne taille au texte pour qu'il puisse rentrer dans le bouton. Dans Pygame (comme dans l'extrême majorité des cas), la taille de la police d'écriture s'exprime en pts, et non en pixels. Pour résoudre ce problème, au lieu de choisir des pts différents pour que des textes différents entrent dans deux boutons de taille identique, nous avons décidé de faire l'inverse. Tous les textes des boutons ont la même taille en pts. Ce sont plutôt les boutons qui s'adaptent selon la longueur du texte. Mais un problème persiste : Comment faire pour déterminer la taille du texte en pixels, si elle est exprimée en pts ? Pour cela, il suffit juste de calculer la surface qui va être affichée, de récupérer son rectangle, et d'appeler la méthode `get_width()` et `get_height()`. Plus simplement encore, nous avons découvert qu'il existe une méthode `font.size()`, qui appliquée à une police d'écriture (objet `font`) et une chaîne de caractères, fait le même processus.

Pour l'instant, le bouton pour recommencer la partie ne fonctionne pas. Mais, une fois les méthodes `str()` de chaque classe définie, la fonction de sauvegarde sera de nouveau opérationnelle, ce qui permettrait à ce bouton de recharger le début de la partie.

3 Transition entre les niveaux

Depuis la dernière fois, nous avons fait une animation en fin de niveau. Cette implémentation a été faite dans le fichier `animations.py`. Lorsque le joueur s'approche du susuwatari à sauver, ils se mettent tous les deux à sauter sur place l'un après l'autre durant l'espace de quelques secondes. Pendant ce temps, un fond noir apparaît petit à petit, jusqu'à devenir opaque. À la fin de l'animation, le niveau suivant est chargé. Cette petite modification nous permet désormais d'avoir une vraie progression dans le jeu.

Pour faire cette transition, nous avons eu besoin d'utiliser une surface différente de celles que nous utilisons couramment, pour pouvoir supporter la transparence (voir la sitographie).

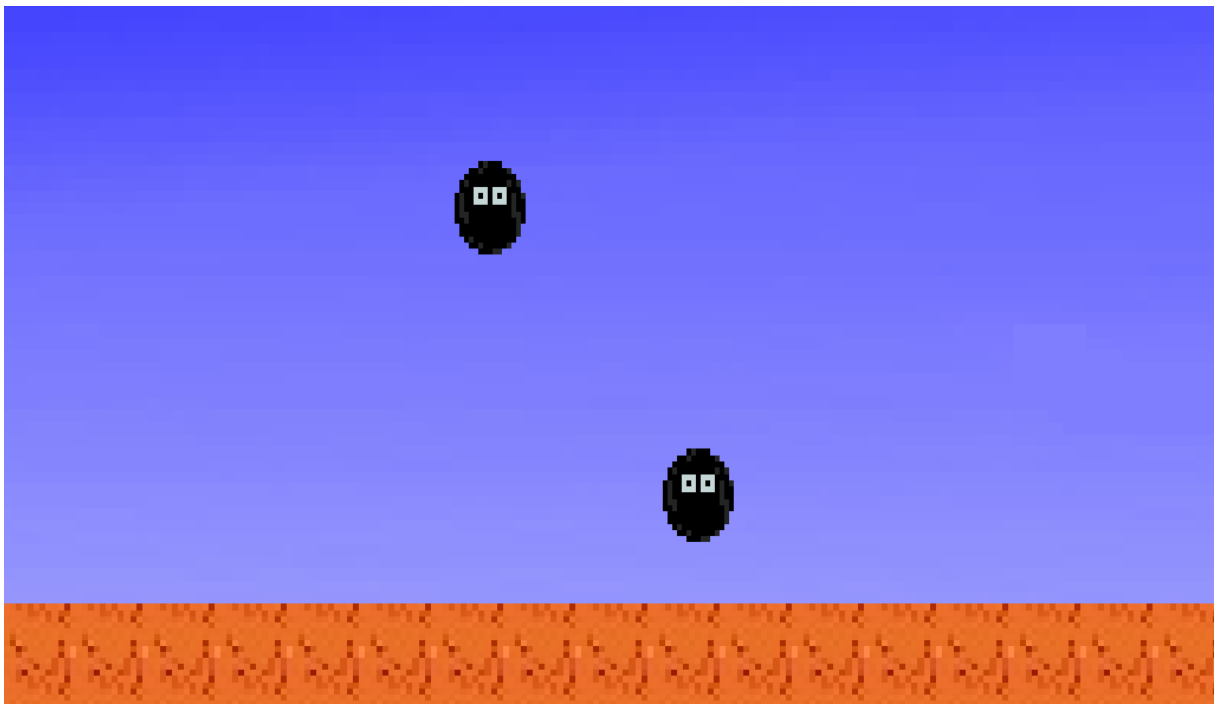


Figure 1: Kaku qui fait un `JUMP Relatif 0` avec son ami

4 Deplacement des Monstres

Au cours de la semaine, nous avons implémenté les déplacements des monstres :

On parcourt `listeEntites` de l'`etatDuJeu`, si l'entite est un monstre, alors on on lui donne une vitesse de 0,02. La vitesse a été choisit de manière afin que le joueur puisse éviter le monstre mais sans qu'il n'aille trop lentement.

Les monstres ne se déplacent qu'à l'horizontale : ils avancent dans une direction et, lorsqu'ils rencontrent un obstacle, ils font demi-tour.

Pour ne pas surcharger le nombre de monstres présents sur la carte, nous avons besoin de faire disparaître certains monstres. Nous hésitons actuellement sur quelle méthode de suppression serait la plus adaptée. Nous avons plusieurs possibilités :

- Avoir un nombre maximum de monstre généré par chacun des générateurs

  Suggestion d'origine

  Le problème étant que certains générateurs généreront plus de monstre que les autres et que, le nombre généré étant aléatoire, les générateurs ne peuvent être placés à n'importe quel endroit

- Avoir un nombre maximum de monstres sur le jeu (`nbMonstreMax`)

  Une fois que chacun des générateur à généré un monstre, on compte le nombre de monstres sur le jeu. Si celui-ci est supérieur à `nbMonstreMax`, alors on enlève les premiers qui ont été générés (depuis le début du jeu) avec la fonction pop() de Python

  Le problème étant que les premiers générateurs risquent de générer plus que les autres car la génération des monstre se fait grâce à une boucle.

- Avoir des générateurs qui ne comptent pas le nombre de monstre qu'ils ont créé, mais plutôt en faire disparaître au bout d'un certain temps.

La plupart de ces suggestions implique la création d'une fonction `disparition` qui créerait une animation lors de la disparition des monstres. On a choisi de faire en sorte que les monstres s'enfoncent dans la terre, ce qui veut dire que nous devons faire augmenter leurs ordonnées $\boldsymbol{y}$ avant de les supprimer de `listeEntites` de l'`etatDuJeu`

5 Sitographie

- Comment utiliser des surfaces transparentes en Pygame
  https://stackoverflow.com/questions/6339057
- Comment obtenir les dimensions d'un texte en Pygame
  https://stackoverflow.com/questions/25149892
- La police Kanit Regular sur Google Fonts
  https://fonts.google.com/specimen/Kanit