Sullyvan LASSAGNE--BUCKWELL

Rapport : Projet jeu du pendu

I – Présentation

Mon projet est un jeu de pendu qui se démarque des autres sur quelques points :

- L'utilisateur peut **entrer** les lettres directement depuis son clavier ou ouvrir une boite de dialogue à l'aide d'un bouton pour taper un mot ou une lettre
- Le jeu est **rejouable** sans devoir réexécuter le programme (bouton cliquable à la fin d'une partie)
- Le nombre de **chances** est personnalisable (on peut commencer avec certaines parties du pendu déjà dessine)
- La **difficulté** du mot est au choix, en mode normal il n'y a pas de caractères spéciaux
- J'ai essayé de pousser **la gestion des erreurs** au maximum, l'utilisateur peut faire beaucoup de bêtises sans que le programme ne plante ou ne bug
- Il possède également un **timer** (temps) ainsi qu'un compteur des chances restantes et erreurs faites à la fin
- Le programme stocke également les **lettres et mots déjà essayés** afin que l'utilisateur ne perde pas de points en retapant plusieurs fois la même chose, même s'il change les accents
- Pour finir, un gros défi a été de géré les **accents**, j'ai réussi à faire comme dans le vrai jeu, exemple : un 'e' valide à la fois 'e' 'è' et 'è' dans le mot et vice versa

II – Explication du code

Je vais ici détailler mon code, sans forcément recopier étant donné que je suis repassé sur à peu près tout donc ça serait trop lourd, (le mieux est de lire le code en simultané). Je nommerai et expliquerai au moins les différentes fonctions ainsi que les difficultés et choses intéressantes auquel j'ai fait face.

setup():

- paramètres qui ne changent pas entre chaque partie (fenêtre)
- je définis également les caractères valides selon la difficulté avec des tableaux globaux. J'ai rencontré des difficultés avec les caractères spéciaux dans les tableaux mais finalement résolus (u)
- ensuite j'appelle une fonction newStart() : je n'y avais pas pensé dès le début mais elle est indispensable pour pouvoir rejouer sans relancer le programme, elle permet de réinitialiser toutes les variables à chaque partie et de rappeler les fonctions d'initialisation

newStart():

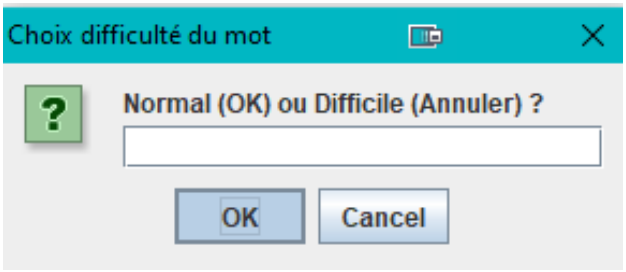
à chaque démarrage, on a besoin d'initialiser plusieurs variables globales :

- trouve, primordiale : 0 = mot pas trouvé, jeu en cours
  1 = mot trouvé, jeu gagné
  2 = mot pas trouvé, jeu perdu
  cette variable (sorte de booléen à trois issues) permet de vérifier l'état du jeu à tout moment
- 1 string lettres_fausses afin d'afficher les lettre fausses déjà entrées par l'utilisateur

- 2 tableaux lettre_testees et mots_testes afin de stocker les erreurs déjà faites

newStart appelle ensuite les fonctions :

- **chances_initiales()** : l'utilisateur choisit le nombre de chances qu'il veut avoir (11 = il commence avec aucune barre), j'ai eu du mal au départ car certaines entrées faisaient crash le logiciel, mais le while corrige : tant que l'utilisateur n'entre pas un nombre entier, la boite s'affiche (while), et s'il entre un mauvais nombre, elles se mettent à 11
- **difficulty()** : ici j'ai trouvé une idée plutôt originale pour utiliser la boite de dialogue, le bouton cancel renvoie None donc j'ai pu prendre en compte les boutons OK et CANCEL pour définir le booléen « hard » (à true ou false)



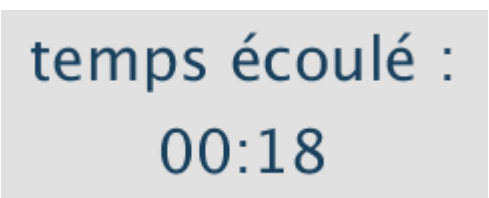
- **tirage()** : à la base Oriane avait mis un « randint » pour choisir un entier (entre 0 et le nb de mots) puis choisir mot = tableau[nombre] mais j'ai changé avec random.choice(tableau), plus pratique pour agrandir le tableau. On a choisi plusieurs mots puis on les a répartis en deux listes, la liste difficile comporte davantage de caractères. Enfin, la chose intéressante est l'apparition de la variable globale **avancee_mot** qui commence à « _ _ _ _ » : (len mot) * « _ » et évoluera au fil du jeu, les - s'affichent dès le début si le mot en contient (« pop-up »)

- **tic()** : faite par Oriane, définition variable de début du temps global start time grâce à l'import time

draw():

appel de toutes les fonctions de dessin, d'affichage, d'entrée utilisateur aussi ainsi que de chronomètre :

- **tac() :** cette fonction a été réalisée par Oriane, elle permet au compteur, variable comp, d'augmenter. J'ai également corrigé le compteur qui affichait 2:6 par exemple au lieu de 02:06 + variable comp_final pour stop si le jeu s'arrête

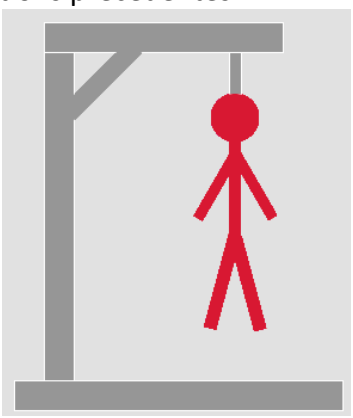


- **compteur():** affichage du compteur, j'ai aidé Oriane pour que les chiffres ne s'affichent pas en superposé, problème rencontré à plusieurs reprises et solvé par la superposition de rectangles de la couleur du fond + affichage de comp_final quand la partie est finie
- **printlettres() :** affichage continu des lettres fausses déjà testées (sous forme de string)
- **bouton () :** affichage du bouton (deviner/rejouer) selon l'état de la partie vérifiable avec la variable « trouve » (bouton vert/rouge selon le nb de chances restantes)

- **mousePressed() :** fonction qui permet de cliquer sur le bouton affiché, elle permet, contrairement à la variable mousePressed, de s'activer une seule fois par clic au lieu de rester activée tant que le clic est enfoncé, permet d'augmenter le frameRate sans problème ! (beaucoup de tps passé là-dessus)

- **keyPressed() :** prise en compte des touches clavier, si l'utilisateur tape une touche avec un caractère valide (selon la difficulté), la fonction correspondance_lettre(touche) s'exécute, s'active 1 seule fois par pression https://processing.org/tutorials/interactivity/#mouse-events
    - **deviner() :** fonction réalisée en partie par Julien, (boite de dialogue) mais j'ai apporté toutes les gestions d'erreur (« Cancel » faisait planter à l'origine). Avec while et l'appel de **prop_valid** notamment (que j'ai écrit), l'utilisateur doit entrer une chaîne valide et nouvelle (ou rien pour revenir en arrière), ensuite s'exécute la vérification de correspondance
    - **correspondance_lettre(prop_lettre) :** permet de vérifier si la lettre est dans le mot à trouver et de remplacer les « _ » dans avancee_mot par les lettres du mot. Pour cela, j'ai utilisé le slicing avec la string. Grâce à [lettre_testees], si la lettre a déjà été essayée, on ne perd pas de chance. Les accents s'affichent bien grâce à la fonction **noaccent**, par exemple un 'ë' tapé valide un 'e' 'é' 'è' etc..
    - **correspondance_mot(prop_mot) :** si le mot est bon, « trouve » se met à 1(win). La vérification se fait avec des mots désaccentués (fonction noaccent encore)

- **win () :** affiche une phrase selon le nombre d'erreurs + nb de chances restantes (tout en vert)
- **lose () :** affiche une phrase suivie de la révélation du mot, tout en rouge
- **printmot(avancee_mot) :** fonction d'affichage de la variable avancee_mot en vert ou rouge selon l'état de la partie, la variable globale avancee_mot évolue grâce aux fonctions précédentes
- **pendu(chances) :** même fonctionnement pour l'affichage du dessin du pendu selon le nombre de chances restantes (moins il y en a, plus il y a de barres avec chances<x : ajouter dessin), la couleur du corps est rouge et devient verte si l'utilisateur gagne



III – Difficultés rencontrées

- J'ai déjà pu citer quelques problèmes rencontrés, on peut aussi parler de la fonction rotate() qui faisait tourner tous les éléments qui la suivent. Pour remédier à ça, j'ai mis pendu() en dernier dans draw et à chaque objet penché je fais rotate(nb), dessin, puis rotate(-nb). Il fallait à chaque fois faire à tâtons pour les coordonnées car le plan (x, y) devient incliné avec rotate, c'est assez fastidieux avec Processing.
- Evidemment j'ai pu rencontrer de nombreuses erreurs de syntaxe, peu intéressantes à expliquer ou encore des problèmes de fonctions qui s'exécutent en boucle à cause de draw
- Mon plus gros problème a été de prendre en compte les accents, ma solution trouvée au bout d'énormément de recherches (importation unidecode impossible, boucles for caractere in mot trop fastidieuses) a été l'utilisation d'une fonction avec re.sub : **noaccent(),** j'ai repris celle-ci https://www.programcreek.com/python/?CodeExample=remove+accents, je l'ai adaptée aux besoins du jeu (ç = c, espace = -). C'est la solution la plus propre et complète possible que j'ai trouvée, elle permet un champ d'amélioration des listes de mot plus large. Elle enlève les accents pour les vérifications de correspondance. L'arument est pratique, elle permet d'appeler la fonction avec n'importe quel variable mot à tout moment ex : noaccent(proposition)
- PS : j'ai laissé quelques « print » (console) à mettre en commentaire pour jouer, cela permet de tester le programme plus efficacement

IV – Bilan

J'ai adoré faire ce projet, je n'ai pas vu les heures passer à essayer d'améliorer le fonctionnement du jeu.
Il m'a appris à mieux maîtriser certaines notions de Python et aussi à comprendre le fonctionnement du découpage en fonctions.

Pour finir, en termes d'améliorations, je pense qu'on pourrait rajouter un menu pour mieux choisir difficulté et chances initiales par exemple, on pourrait pourquoi pas envisager un historique des parties.
J'aurais aussi pu agrandir la liste de mots mais ce n'était pas forcément intéressant à faire, pour ma part je vais essayer d'intégrer le jeu à mon site mais ça s'annonce assez difficile…