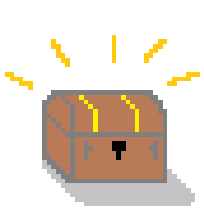
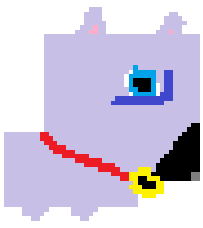
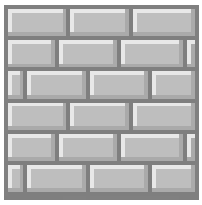
DUC Julien
DYER John
GUERBOIS Alban

M1 E.I.F.

Projet Python : Julius et Martin







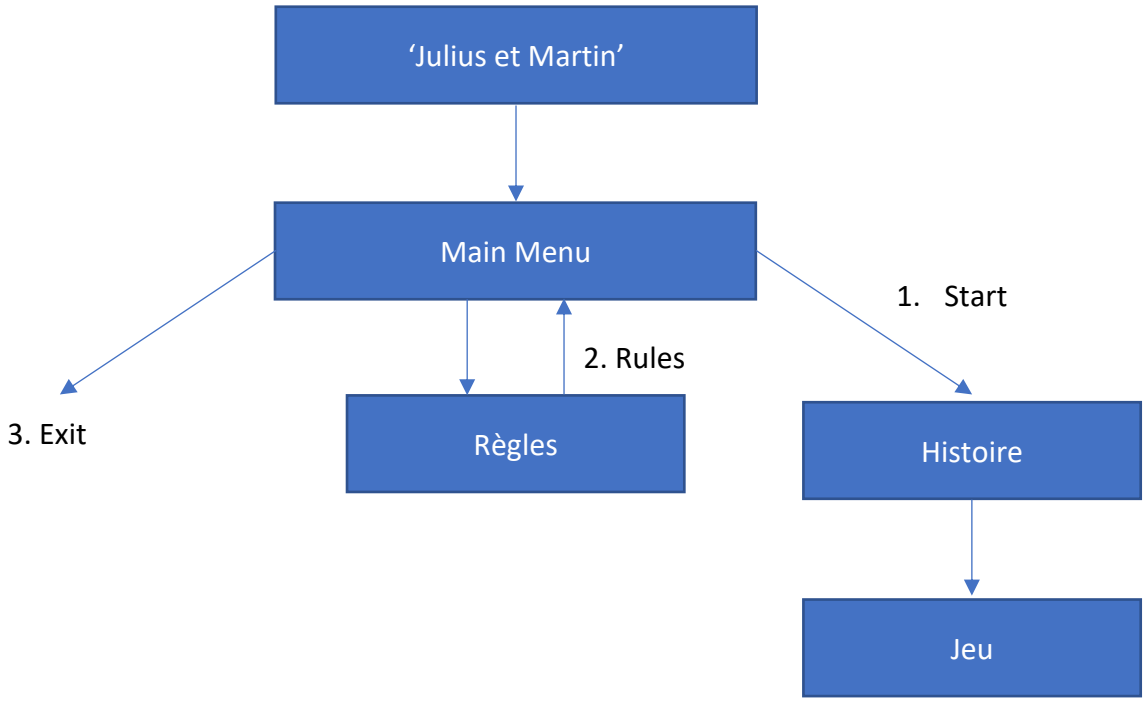
Année 2017-2018
Professeur : Jean-Christophe GAY

Introduction

Pour ce projet Python, nous avons décidé de créer un jeu de labyrinthe composé de 10 niveaux. Le joueur doit trouver la sortie en un temps limité en se déplaçant à l'aide des flèches du clavier. Il peut récupérer des vies qui lui permettront de recommencer le niveau auquel il échoue et récupérer des pièces dans les coffres avec lesquelles il peut s'acheter des chaussures qui augmentent sa vitesse.

L'interface graphique :

Le jeu comporte une interface graphique utilisant le module pygame. Il comporte un menu avec plusieurs affichages :



Nous avons effectué les graphismes nous-mêmes à l'aide de Paint en prenant en compte la pixelisation de notre 'display window' (fenêtre d'affichage). La fenêtre est donc de 1000 px de largeur et 800 px de hauteur.

La première partie du programme est constituée du chargement de toutes les images utilisées et du paramétrage de la fenêtre.

Spécificités de Pygame :

La classe la plus importante de pygame est la catégorie d'objets de type Surface. On peut voir cela comme un bout de papier. Une surface peut être « collée » sur une autre créant un effet de premier et second plan, on peut écrire dessus, changer la couleur de son fond etc…

Les méthodes spécifiques à une Surface principalement utilisées sont :

.blit() (coller)

.fill((r,g,b)) (remplir une surface avec un code couleur rgb)

.set_colorkey((r,g,b))
(la couleur spécifiée de la surface est rendue transparente)

Les Surfaces évoluent sur notre fenêtre de jeu (variable window qui est une surface d'affichage) qui sera raffraichie grâce à la méthode pygame.display.update().
Ainsi un déplacement d'une surface x est simplement une alternance d'affichage avec x.blit()et window.fill().

Pygame détecte un certain nombre d'évènements, tels que la position et le déplacement du curseur de la souris, les clics, le fait d'appuyer ou de relâcher une certaine touche du clavier etc…
On accède à ces évenements par pygame.event.get().
Chaque event un certain type, accessible grâce à pygame.event.type.
On note par exemple,
pygame.KEYUP (On appuie sur une touche du clavier)
pygame.KEYDOWN (On relache une touche du clavier)
Pygame.QUIT (On quitte la fenêtre)

On peut aussi accéder à la touche du clavier en question avec event.key.

Ainsi l'instruction suivante, qui sera rencontrée de nombreuses fois, va inspecter indéfiniment les évènements qui ont lieu, si l'utilisateur appuie sur la touche 1, le message ("touche 1 appuyée") sera affiché sur la console, si appuie sur le bouton de fermeture de pygame, alors la fenêtre se fermera.

```
while True:

for event in pygame.event.get():

                if event.type == pygame.QUIT:
```

```
            pygame.quit()

        elif event.type == pygame.KEYDOWN:

            if event.key == pygame.K_1
                print("touche 1 appuyée")
```

Les menus :

Les fenêtres de menu :

- **Fenêtre de titre :**
  - Affichage du titre du jeu 'Julius et Martin'
  - Évènements :
    - Quitter le jeu : pygame.quit()
    - Appuyer sur n'importe quelle touche pour aller au Main Menu.
- **Main Menu :**
  - Le joueur peut appuyer sur :
    - 1 : Jouer
      - **Histoire :** nous faisons défiler l'histoire du jeu sur l'interface graphique en lisant les lignes d'un fichier .txt.
      - **Jeu :** Le jeu se lance après l'histoire, nous détaillerons son fonctionnement ultérieurement.
    - 2 : Lire les règles
      - **Fenêtre « Rules » :** permet de lire les règles et de revenir au **Main Menu** en appuyant sur ENTREE.
    - 3 : Quitter le jeu

Les boucles de menu :

- **While Menu :**
  - Affichage du titre et des options en collant du textes et des images sur la fenêtre.
  - Gestion des évènements : Le joueur peut quitter le jeu (if event.type==pygame.QUIT) ou appuyer sur 1,2,3 comme décrit précédemment Si celui-ci appuie sur 2, la boucle **while Subenu_Rules :** s'enclenche. S'il appuie sur 1, l'histoire se lance, puis le jeu.

- **While Submenu_Rules :**
  - Affichage des messages et images
  - Evenements :
    - Quitter en appuyant sur la croix
    - Retour au Main_Menu en appuyant sur ENTREE

Board de jeu :

Les niveaux sont des fichiers .txt composés de :

- 0 : fond noir
- 1 : mur
- 2 : départ
- 3 : coeur
- 4 : sortie
- 5 : coffre

Par exemple, le 1er niveau est le suivant :

```
00000000000000000000
11111111111111111111
10000000121000000011
10000000101000000011
10000000101000000011
10000000101000000011
10000000101000000011
10000000101000000011
11111111101111111111
10000000000000000111
10100000011010000011
10110000111010010011
10110111130010000011
10110100111101000011
15110000041100010511
11111111111111111111
```

Nous voulions découper notre board de jeu en un tableau de 20 colonnes et 16 lignes, ainsi notre board ressemble à un tableau de 320 cases de 50*50 pixels.

Pour créer la matrice correspondante,
on ouvre d'abord le fichier texte avec la méthode
open('x.txt').

L'instruction suivante permet de créer une liste de listes, le 1er élement étant la 1ère ligne du tableau sous forme de chaîne de caractère.

```
niveau = [ligne.split() for ligne in lvl]
```

Ainsi niveau[1][0] renvoie '11111111111111111111'

Donc pour obtenir sous forme d'entier l'élément de la colonne c, ligne l du niveau on écrit : int(niveau[l][0][c])

La création d'images pour chaque type d'élément du tableau a été laborieuse.

Le héro :

Julius n'est finalement qu'une surface disposant d'une variable position_hero qui est une liste de 2 éléments, sa position en ligne et sa position en colonne.

La position de départ de Julius est determinée par :

```
starting_position = hero.get_rect(topleft=(j*50, i*50))
```

j*50 et i*50 permet la conversion d'une position de type ligne/colonne dans une matrice à une position de type coordonnées de pixels dans la fenêtre window.
La colonne 0 ligne 0 de la matrice (base 0) correspondant à la coordonnée (0,0) en terme de pixels.

```
position_hero = starting_position
```

Nous allons pouvoir "tracker" un rectangle englobant l'image de notre héros. Ainsi, nous pouvons déplacer ce dernier en utilisant la méthode Move((pixels en longueur, pixels en hauteur) et en collant l'image de Julius avec window.blit(hero,position_hero).

Methodes Check :

Les méthodes de type checkmove ou check_chest/check_lives/check_exit que nous avons créés prennent en input position_hero et renvoient un booléen. Elles fonctionnent toutes de la même manière.

```
x = int(position_hero[0]/50)
y = int(position_hero[1]/50)
```

On récupère la coordonnée de l'élement de la matrice dans lequel se trouve le personnage.
Si position_hero[0] = 50 et position_hero[y] = 50
X = 1 et y = 1 (50,50 est en fait la position topleft c'est-à-dire du coin haut gauche)

```
alignx = position_hero[0] % 50
aligny = position_hero[1] % 50
```

Ainsi si alignx = 0 ou aligny = 0 alors le héro est parfaitement aligné dans les lignes ou les colonnes

Pour se déplacer vers le haut, si il est parfaitement aligné alors il suffit de vérifier si int(niveau[y-1][0][x]) !== 1:

S'il est aligné en ligne mais pas en colonne alors on vérifie que :

```
int(niveau[y-1][0][x]) !== 1 and int(niveau[y-1][0][x+1]) !== 1
```

C'est-à-dire que les cases au dessus de lui et au dessus à droite ne sont pas des murs.

On execute ce genre de tests pour la direction concernée recupérée par le gestionnaire d'évènements.
Si la fonction checkmove renvoie True, alors on effectue le déplacement correspondant.

Ces conditions sont nécessaires car dans un souci de réalisme nous avons voulu que le personnage se déplace par « saut » de 5 pixels et non de case en case.

Affichage du board :

Pour afficher la board on va parcourir les élements du niveau sous forme d'entier, si cet élément est égal à tel ou tel entier, 1 par exemple on affichera l'image correspondante soit un mur dans cet exemple a la position (j*50,i*50) avec j = colonne dans la matrice et i = ligne.

Lorsque le personnage récupère un item, celui-ci est détruit (remplacé par un 0 dans niveau). Il ne réapparaîtra pas après la mort, mais ses effets (ajout d'argent ou de cœur) subsistent. Ainsi le joueur ne peut récuperer en premier en cœur pour rejouer indéfiniment le même niveau jusqu'à trouver le meilleur chemin possible.

La boucle while game :

Nous avons réalisé une boucle sur l'ensemble des 10 niveaux. La matrice du board est donc rechargé à chaque passage de niveau.

S'en suit une boucle classique while game :
On initialise les golds du joueur à 0, son nombre de cœur à 0 et sa vitesse à 20 ticks par seconde. Cela sera discuté dans la partie gestion du temps.

Ensuite, nous rentrons dans une boucle qui rend l'écran tout noir, on ne voit plus que le personnage. Il faut appuyer sur

espace pour allumer la lumière, cela fait partie du jeu. En effet, de cette manière le joueur n'a pas accès au labyrinthe avant que le compteur ne se déclenche, il ne peut donc pas prendre son temps pour prévoir le chemin qu'il va emprunter. Cette boucle sera réappelée si le joueur meurt et qu'il lui reste une vie.

La boucle a cette forme :
While game :

Boucle Shop tous les 3 niveaux, renvoie un nombre de golds et une vitesse.
Boucle light (décrite plus haut)

- Calcul du timer
- Récupération de la touche préssée
  - On vérifie que le déplacement est possible
- Changement de l'état de marche
- Gestion des évènements
  - Changement d'image du hero lorsqu'on relache les touches de direction.
- Affichage du background
- Affichage des murs, des coffres, des cœurs et de leurs animations.
- Affichage du héro
- Affichage du level
- Affichage du timer
- Affichage des golds
- Affichage du nombre de vies
- Raffraichissement de la fenêtre
- Appel de check_chest, si le perso est sur un coffre, cela lui donne un montant d'or aléatoire entre 1 et 100.
- Appel de check_lives, seulement si le personnage a moins de 3 cœurs, ajout au compteur de cœurs.
- Appel de check_exit, si le personnage se trouve sur la sortie affichage d'un message de réussite puis sortie de la boucle -> niveau suivant
  - Si le joueur est au dernier niveau message de fin
- Si le timer arrive à 0, affichage d'un message de défaite et retrait d'une vie.
  - Le jeu continue si le joueur a encore une vie, retour à la boucle light.
  - Sinon le jeu s'arrête GAME OVER

Difficultés rencontrées :

Gestion du temps

L'utilisation du module pygame.time et notamment de pygame.time.Clock().tick() nous a permis de gérer le nombre de boucles par seconde et ainsi d'influencer la fréquence à laquelle les images sont affichées. Cela nous a permis notamment de modifier la vitesse, lorsque le joueur achète des nouvelles chaussures dans la boutique.
Nous avons aussi utilise ce module pour confectionner un timer.

```
countdown = (21000-nv*1000 - pygame.time.get_ticks()+start+pause)/1000
```

pygame.time.get_ticks() permet de récuperer le temps en millisecondes depuis pygame.init(), la 1ère instruction de notre programme.

Comme notre timer doit être réinitialisé, start est la valeur renvoyée par un appel de pygame.time.get_ticks()juste avant de commencer le jeu.
Pause est une variable qui augmente de 1500 à chaque fois qu'on ouvre un coffre ou qu'on trouve un cœur car on freeze le code pendant 1500 millisecondes pygame.time.delay() à ce moment-là alors que le timer tourne toujours.
A chaque fois qu'on se trouve dans la boucle light pause et start sont reinitialisés, le 1er à 0 et le second avec pygame.time.get_ticks().

Nous avons trouvé cette mode de gestion du temps peu pratique et imprécise, toutefois cela est difficile à gérer puisque il existe un facteur « aléatoire », l'éxecution d'une instruction ne prend pas exactement le même temps à chaque fois.

Animations :

Nous avons implémenté des animations en utilisant des compteurs, en spécifiant l'affichage d'une certaine image lorsque compteur modulo x == y.
Par exemple si x = 2, on pourra avoir une animation binaire en deux temps.

La plupart des animations présentes ont une fréquence de 3 images. Toutefois nous n'avons pas trouvé de solutions pour permettre de déplacer notre personnage avec une vitesse élevée tout en ayant des animations lentes. Il aurait surement fallu utiliser des threads (vu un petit peu en c#).

Conclusion :

La documentation de pygame était bien fournie, ce qui nous a permis d'utiliser les outils standards du module, toutefois nous n'avions pas remarqué que nous pouvions update seulement un rectangle ou une liste de rectangles, ce qui aurait pu nous être bien utile.
Nous aurions aimé ajouter des monstres qui « pop » dans le labyrinthe et que le joueur doit éviter ou tuer. Notre code aurait pu être plus affiné en utilisant des classes objets et en le divisant en plusieurs modules, même s'il n'est pas très lourd.