L2 MIASHS - Informatique S3
Mini Game

Garance Gourdel

2020-2021

1 Introduction

L'objectif de ce TD (sur plusieurs séances) est de vous faire réaliser un petit jeu vidéo, jouable dans un terminal.

Le TD comprend de nombreuses parties, il n'est pas nécessaire de toutes les faire. Les questions ont pour la plupart plusieurs solutions.

Le but est que vous mettiez en application les connaissances acquises dans les précédents TDs.

2 Affichage de base

2.1 Carte

On vous donne une carte sous la forme d'une matrice et un dictionnaire qui associe à ses nombre un caractère :

```
# Exemple de carte
map = [[0,0,0,1,1],
       [0,0,0,0,1],
       [1,1,0,0,0],
       [0,0,0,0,0]]

dico = {0:' ',1:'#'}
```

1. Écrivez une fonction `display_map` qui prend en argument une carte (une matrice donc) `m` et un dictionnaire `d` qui associe chacun des nombres dans `m` à un caractère, et qui affiche la carte.
   Pour cet exercice, vous aurez besoin de connaître l'option `end` de la fonction `print`. Cette option permet de préciser ce qu'on affiche après avoir afficher les arguments donnés à print. Par défaut print affiche un retour à la ligne, `print("blabla", end="")` permet d'afficher blabla sans revenir à la ligne après l'avoir affiché.

2. Testez votre programme en affichant la carte et le dictionnaire donnés précédemment. Le résultat devrait être :

```
   ##
    #
##
```

2.2 Personnage

Le but est désormais d'ajouter un petit personnage. On va le représenter dans le code par un dictionnaire contenant 3 entrées `char` , `x` , et `y` : l'entrée "char" contient le caractère représentant le personnage (ici "o"), "x" sa position en abscisse et "y" sa position en ordonné.

1. Écrivez une fonction `create_perso` qui prend pour argument `départ` un couple d'entiers représentant la position de départ, et renvoie un dictionnaire représentant le personnage.

2. Testez votre fonction en affichant `create_perso((0,0))`, le résultat devrait être {`"char":"o", "x":0, "y":0` }.

3. En vous inspirant de votre fonction `display_map` créer une nouvelle fonction `display_map_and_char` qui prend une carte `m`, un dictionnaire de caractères `d` et un personnage `p`, et affiche la carte et le personnage sur cette carte. Attention, elle ne doit pas modifier `m`, `p`, et `d`.

4. Testez votre fonction avec la même carte et le même dictionnaire donnés précédemment et le personnage crée précédemment, le résultat devrait être :

```
o  ##
    #
##
```

2.3 Déplacement

Le but est maintenant d'ajouter la possibilité de déplacer son personnage, on va utiliser la fonction `input` que vous avez déjà utilisé et qui permet de récupérer ce que l'utilisateur écrit dans le terminal.

Rappel de d'utilisation de `input` :

```
d = input("Quel déplacement ?")
```

Ici on affiche "Quel déplacement ?" puis on attend que l'utilisateur écrive quelque-chose et appuie sur "entrée" et on stocke dans `d` ce que l'utilisateur a écrit.

À noter que ce n'est pas ce qu'il y a de plus naturel comme contrôles et on vous proposera de l'améliorer dans la suite du TP.

Avec la fonction `input`, on veut récupérer une lettre entre "z" (haut), "q" (gauche), "s" (bas), et "d" (droite), si on a autre chose on dit que c'est une erreur et qu'il faut recommencer.

1. Écrivez une fonction `update_p` qui prend en argument une lettre `letter` et un joueur `p` (représenté comme expliqué ci dessus par un dictionnaire) et qui met à jour la position du joueur et ne renvoi rien (il n'y a pas encore besoin de la fonction `input`).

2. Récupérez l'entrée de l'utilisateur avec `input`, et mettez à jour la positon du joueur avec `update_p`, Affichez de nouveau la carte et le personnage (à sa nouvelle position).

3. Testez votre programme en rentrant une des directions à l'exécution.

2.4 Jouer à l'infini

Vous avez maintenant pu déplacer notre personnage une fois, il est naturel que vous souhaitiez pouvoir le faire autant de fois que vous le vouliez. Pour cela vous aller avoir besoin d'une boucle `while`.

1. Mettez le code nécessaire pour déplacer le personnage une fois (input puis mise à jour de la position, puis affichage de la carte) dans une boucle `while True :`. Cela répétera indéfiniment l'exécution du code contenu dans la boucle.

2. Testez en essayant de déplacer votre personnage plusieurs fois. Vous devrez alors stopper l'exécution en appuyant sur le bouton Stop (le carré proche du terminal).

3 Améliorations

3.1 Ne pas sortir de la carte

En testant votre programme, vous avez peut être pu voir que rien ne vous empêche de sortir de la carte (en fonction de votre programme ça provoque peut être même une erreur).

1. Modifiez la fonction `update_p` pour vérifier qu'on ne sort pas de la carte. Vous aurez besoin d'ajouter la carte `m` aux paramètres.

2. Testez en essayant de sortir de la carte.

3.2 Murs infranchissables

Maintenant le but est de rendre les murs (représenté sur la carte par des 1 et avec comme caractères "#") infranchissable. C'est a dire que si le joueur veut se déplacer sur une case avec un mur, il ne peux pas.

1. Modifiez la fonction `update_p` pour vérifier qu'on ne rentre pas dans un mur.

2. Testez en essayant de passez au travers d'un mur.

3.3 Objets à récolter

Pour donner un objectif au jeu, on va maintenant rajouter des objets à récolter symbolisés par des ".". On choisi leurs positions au hasard avec la fonction `randint` de la bibliothèque `random`.

Pour obtenir un nombre `n` entre 0 et 10 (inclus) il suffit d'écrire au début du fichier `import random` et puis `n = random.randint(0, 10)`.

- Écrivez une fonction `create_objects` qui prend comme argument `nb_objects` le nombre d'objets qu'on souhaite ajouter et `m` la carte et qui renvoie un set de couple représentant la position des objets. Pour chaque objet, on tire au hasard une position sur la carte et si elle est vide (on a un 0 dans `m`) on ajoute un objet, si elle est déjà prise on ne fait rien.

- En vous inspirant de votre fonction `display_map_and_char` créez une nouvelle fonction `display_map_char_and_objects` qui prend les mêmes arguments que `display_map_and_char` mais aussi `objects` un ensemble d'objets. Attention, elle ne doit pas modifier ses arguments.

- Testez votre jeu en l'exécutant.

3.4 Ramasser automatiquement les objets

Maintenant il faut implémenter le fait qu'on ramasse automatiquement un objet si on marche dessus.

- Écrivez une fonction `update_objects` qui a pour argument `p` le joueur et `objects` l'ensemble des objets et qui retire l'objet (s'il y en a un) à la position du joueur.

- Ajoutez un appel à `update_objects` dans votre boucle while de jeu, entre `update_p` et `display_map_char_and_objects`.

- Testez votre jeu en l'exécutant et en marchant sur des objets.

3.5 Afficher le score

On peut ramasser des objets, mais pour l'instant ça ne sert à rien... Il faut ajouter et afficher un score qui correspond au nombre d'objets ramassé. On enregistre ce score dans le dictionnaire du joueur avec une nouvelle clé : "score".

- Modifiez votre fonction `create_perso` pour y ajouter la clé "score" et initialiser le score à 0.

- Modifiez votre fonction `update_objects` pour qu'elle augmente le score du joueur si on ramasse un objet.
- Modifiez votre fonction `display_map_char_and_objects` pour qu'elle affiche le score du joueur en dessous de la carte.
- Testez votre jeu en l'exécutant et en marchant sur des objets.

4 Fonctionnalités supplémentaires

À partir de là, les améliorations sont facultatives, il est possible d'avoir une très bonne note sans. Elles sont un peu plus difficiles, mais elle permettent de se rapprocher d'un vrai jeu.

4.1 Génération de carte aléatoire

La carte donnée en début de sujet est toute petite et pas très intéressante. Le but est donc d'en générer automatiquement de nouvelles, plus grandes.

Notre but pour cette partie est d'écrire une fonction `generate_random_map` qui prenne comme argument `size_map`, une paire qui représente la taille de la carte, et `proportion_wall` un nombre a virgule : la proportion de cases qui sont des murs sur la carte et renvoi la matrice représentant la nouvelle carte.

Dans la fonction `generate_random_map` :

- Commencez par créer une matrice remplie de 0 de dimension `size_map`.
- À l'aide de la fonction `random` précédemment utilisé pour rajouter les objets, ajoutez les murs (marqués par des 1 dans la matrice) en respectant la proportion `proportion_wall`.
- Renvoyez la matrice.
- Testez votre code avec la taille de carte et la proportion de mur de votre choix.

4.2 Entrée et sortie du niveau

Si on ramasse tous les objets d'un niveau, on a plus rien a faire, c'est pas top... Pour réparer ça on va rajouter une entrée et sorties à chacune de nos cartes. Ainsi quand on atteint la sortie, on change de niveau, la carte change et on a de nouveaux objets à ramasser. Dans notre matrice on représente l'entrée par un 2 et la sortie par un 3. À l'affichage l'entrée n'est pas affichée (elle et représenté comme une case vide, `" "`) et la sortie est marquée par un `X`.

- Modifiez votre fonction `generate_random_map` pour ajouter dans la carte une entrée et une sortie, elles viennent remplacer des cases vides.

- Créez une fonction `create_new_level`, qui prends comme argument le personnage `p`, la carte `m`, les objets `obj`, `size_map` et `proportion_wall`, et qui remplace `m` par une nouvelle carte, remplace les objets, et place le personnage a l'entrée du niveau m.
- Modifiez votre boucle `while` de jeu pour appeler `create_new_level` si le personnage atteint la sortie.
- Testez en rejoignant la sortie du niveau !

4.3 Ajout de bombes

Avec notre nouvelle création de carte, on ne vérifie pas que la sortie est accessible depuis l'entrée, le personnage peut donc se retrouver coincé. Pour régler ce problème, on va rajouter une action possible au personnage : avec la touche E, le joueur pourra faire exploser tous les murs présents sur les 8 cases entourant son personnage.

- Créez une fonction `delete_all_walls` ayant pour argument la carte `m` et une position `pos`, qui enlève les murs de la carte autour de la position `pos`.
- Modifiez la fonction `update_p` pour autoriser la touche E et l'associer à la suppression des murs autour du personnage.
- Testez !

4.4 Ce que vous voulez

Si ce projet vous a plu et que vous souhaitez le continuer, libre à vous d'ajouter des choses en plus (Un monde multi-joueur peut être ?) ou de modifier légèrement l'apparence du jeu, Il suffira de rendre en même temps que votre code de projet des explications sur ces ajouts et modifications !