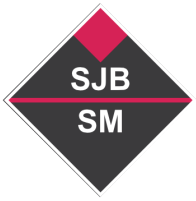
Institut Saint Jean Berchmans - Sainte Marie
Section informatique



Programmation
The Shell Adventure

Travail de fin d'étude
réalisé par
Jordan Dalcq

Année académique 2018 - 2019

Table des matières

Remerciments

Je tiens tout d'abord à remercier mon grand père Yvan, qui m'a introduit au monde de l'informatique dés mon plus jeune âge, et de m'avoir montré aussi les joies de Linux dés mes 7 ans.

Je remercie mes parents Fabienne et Marc pour avoir investis en moi afin de me permettre de continuer sur ma voie vers un métier qui me pasionne.

Je remercie Alex Roşca, d'avoir été mon premier amis, dans cette grande aventure, merci de m'avoir permis de découvrire la programmation

Je remercie Monsieur David Carrera pour avoir été le seule professeur qui a été capable à me poussé au meillieur dans ma passion.

Je tiens aussi à remercier toutes les personne que j'ai rencontré à l'Institut Saint Jean Berchmans, pour m'avoir influencé dans ce projet d'une manière ou d'une autre.

Première partie

Introduction et Analyse

Contexte

Durant quelques années j'ai été mentor au Coderdojo de Liège durant 2 ans. J'animais un atelier Python / Linux auprès de jeunes âgés entre 14 et 18 ans. Le problème lors de l'apprentissage était qu'ils ne savaient pas quelle commande / fonction utiliser dans un cas précis. Dans ce projet, j'ai décidé de me focaliser sur le terminal bash et de présenter un outil d' apprentissage basé sur le visuel.

Chapitre 1

Présentation

1.1 Fonctionnement

L'idée de base est assez simple, il faut taper des commandes pour interagir avec le monde qui nous entoure.

Comme par exemple :

- cd : Pour se déplacer d'une pièce à une autre touch : Pour créer un objet ou bien faire apparaître une personne
- cp : Pour cloner un objet ou personnage
- mv : Pour déplacer un objet ou personnage
- cat : Pour connaître le contenu d'un objet ou alors l'identité d'un personnage
- rm : Pour jeter un objet ou "éliminer" un personnage
- tree : Scanner à rayon X pour voir à travers les murs

A cause du temps assez limité, j'ai implimenté un langage de programmation afin de laisser à l'utilisateur une liberté de créer des niveaux à sa guise.

FIGURE 1.1 – Capture d'écran de la disposition par défault

1.2 Interface utilisateur

Par défaut, l'interface du jeu se présente comme représenté sur la figure. 1.1

Élément visuel / RPG : Là où seront dessinés les personnages et objet (armoir, chaise, garage, maison ...).

Ligne de commandes : Permet d'introduire des lignes de commande SH, qui seront interprétés par la suite.

Objectifs : Permet d'afficher les tâches que le joueur doit effectuer.

Il est aussi important de noter que chaque scripts est libre de changer cette disposition comme bon l'entend.

Chapitre 2

Analyse

2.1 Classes

2.1.1 game.mechanics.term.Term

La classe *Term* permet de créer des surfaces d'interaction tel que ligne de commande POSIX et entrée utilisateur (= input), une fois initialisé elle charge la police de caractères monospace (en 22 pixels car c'est beaucoup plus lisible et agréable à utiliser), ensuite elle crée un dossier *.shelladv* dans le dossier personnel de l'utilisateur. (Exemple : */home/dalcjor/.shelladv*)

Attributs

- `surface : pygame.Surface` → Surface principale du terminal.
- `mono : pygame.font.Font` → Police de caractère monospace.
- `visualLine : List[str]` → Lignes visibles sur la surface du terminal (Résultats de commandes ou entrées de l'utilisateur).
- `lineRect : pygame.Rect` → Rectangle de collision de l'entrée utilisateur.
- `blinkRect : pygame.Rect` → Rectangle de collision avec le rectangle clignotant (= Blink).
- `fontSurface : pygame.Rect` → Rectangle de collision avec les lignes précédentes.
- `inInput : bool` → Permet de savoir si l'utilisateur est en train d'entrer du texte.
- `promptVisual: bool` → Permet de savoir si le terminal doit afficher le prompt.
- `bash: bool` → Permet de savoir si le terminal doit exécuter les commandes entrées par l'utilisateur ou pas.
- `currentTyping : str` → Stock la dernière entrée utilisateur
- `blinkX : int` → Position x du rectangle clignotant à côté du prompt (=Blink).

- `custom : str` → Stock les décorateur terminal (=prompt) customisés.
- `history : mechanics.term.History` → objet qui gère l'historique du terminal.
- `env : Dict[str, str]` → Variables d'environnements semblables à celles présentes dans les systèmes UNIX (Linux, MacOS) et NT (Windows).
- `prompt: str` → Permet de prendre connaissance du nom d'utilisateur, nom de la machine et le CWD (Current Work Directory, = Dossier de travail courrant).
- `tick: float` → Heure actuel pour permettre de réguler la vitesse du blink (le rectangle à côté du prompt).

Méthodes

- `resize(size: Tuple[int, int])` → Change la taille de la surface du terminal.
- `disable_prompt()` → cache le le decorateur (=prompt).
- `enable_prompt()` → Ré-affiche le décorateur (=prompt).
- `isprompt_enabled() : bool` → Retourne `True` si le décorateur (=prompt) est affiché.
- `disable_bash()` → Désactive l'exécution des entrées de l'utilisateur.
- `enable_bash()` → Réactive l'exécution des entrées de l'utilisateur.
- `getInput()` → Active l'entrée utilisateur afin d'en recupérer le contenu par la suite.
- `removeLine()` → Supprime la dernière ligne affichée dans le terminal.
- `set_custom_prompt(string: str)` → Permet de mettre un décorateur (=prompt) customisé.
- `add_to_display(output: str)` → Permet d'ajouter une ligne à l'attributs. `visualLine`, qui servira ensuite pour l'affichage
- `clear()` → Permet d'effacer les lignes affichées à l'écran.
- `get_env(): Dict[str, str]` → Retourne les variables d'environnements.
- `draw()` → Permet de dessiner les lignes de commandes entrées par l'utilisateur.
- `drawBlink()` → Dessine le rectangle qui clignote à côté du décorateur (=prompt) et de l'entrée utilisateur.
- `keydown()` → Méthode de traitement des touches pressées, pour ensuite les affichéer dans le terminal.
- `update()` → Méthode qui gère le curseur et appelle la méthode `draw`
- `get_surface(): pygame.Surface()` → Actualise et retourne l'écran (=surface) du terminal.

2.1.2 game.mechanics.term.History

Comme dans tout bon terminal POSIX, il y a un gestionnaire d'historique, cela permet de rechercher des commandes déjà tapées. Les historiques sont sauvegardé de façon à ce que l'utilisateur puisse retrouver ces commandes de jeu qu'il a précedemment entré.

Attributs

- `home: str` → Chemin vers le dossier utilisateur.
- `hist: file` → Lien en lecture vers le fichier d'historique (`\$HOME/.bash_history`).

Méthodes

- `__getitem__(index: int) : str` → retourne une ligne spécifique de l'historique.
- `append(line: str)` → ajoute une ligne à l'historique.
- `openFile()` → Ouvre le fichier d'historique.
- `get_size() : int` → Retourne le nombre de commande présent dans l'historique.
- `get_previous() : str` → Retourne la commande qui a été précedemment introduite.
- `get_next()` → Retourne la commande qui suis la dernière retournée.

2.1.3 game.mechanics.rpg.Rpg.Rpg

La classe `Rpg` permet d'afficher les éléments visuels, que se soit personnages ou objets.

Attributs

- `sprites: Dict[str, game.mechanics.rpg.sprite.Sprite]` → Dictionnaire qui lie un sprite à un nom (Le nom étant une variable du Adventure Script), ce dictionnaire contient tous les sprites à afficher à l'écran.
- `surface: pygame.Surface` → Surface du Rpg.
- `mouseCollide: bool` → Informe si la souris passe au-dessus de la surface du RPG.

Méthodes

- `add_to_surface(name: str, sprite: game.mechanics.rpg.sprite.Sprite)` → Permet d'ajouter un sprite à l'écran.
- `clear()` → Efface complétement la surface RPG.
- `set_mouse_collide(collide: bool)` → Informe la classe RPG que la souris passe dessus.
- `keydown(key: int)` → Transmet les entrées clavier de l'utilisateur au différent Sprite (= personnage, objet).
- `update()` → Actualise la surface et applique les sprites à la surface.
- `resize(size: Tuple[int, int])` → Change la taille de la surface.

2.1.4 game.mechanics.rpg.sprite.Sprite

Cette classe permet de gérer les spritesheets (=Image qui compose les mouvements des personnages, objet) automatiquement, grâce au *Adventure Script*, il est possible de gérer des spritesheets avec 4 actions dans cette ordre précis : Pars vers le haut, pars à Droite, pars à Gauche, pars vers le bas. Le premier sprite doit être celui par défaut (au repos). Par contre vous pouvez mettre autant de frames à une action (= étape par action) que vous voulez. Toutes les configurations nécessaires sont faites via *Adventure Script*.

FIGURE 2.1 – Exemple de spritesheet compatible

Attributs

- `size: Tuple[int, int]` → Taille d'une frame du sprite.
- `finalSize: Tuple[int, int]` → Taille d'affichage d'une frame (car elles sont souvent trop petites).
- `index: Tuple[int, int]` → Position de la frame actuelle.
- `limitSprite: int` → Nombre de frames par action.
- `spriteSurface: pygame.Surface` → Surface dédiée à une frame.
- `spriteSheet: pygame.Surface` → Surface qui contient toute la spritesheet au complet.

Méthodes

- `get_pos() : Tuple[int, int]` → Retourne la position du sprite (=personnage, objet) sur la Surface dédié au RPG.
- `get_surface(): pygame.Surface` → redessine le sprite (=personnage, objet) et retourne sa surface.
- `reset(index: int)` → Met le sprite (=personnage, objet) à une position repos en fonction de sa direction, s'il est à sa dernière frame ou s'il change de direction.

— `move(self, way: int)` → Fait passer le sprite à sa frame suivante.
— `stop(way: int)` → Met le sprite en repos, car il a fini de se déplacer.

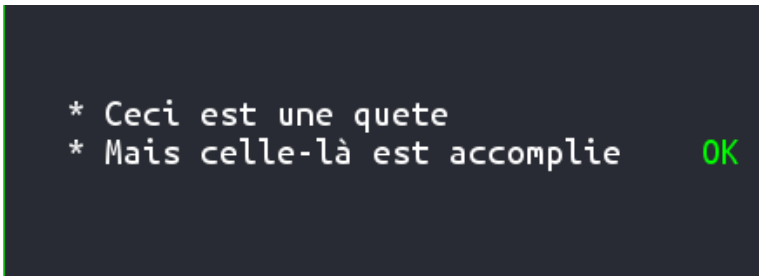


FIGURE 2.2 – Exemple de quêtes

2.1.5 game.mechanics.quest.Quest.Quest

Cette classe permet de gérer les quêtes grâce) à l'Adventure Script Il est important de noter que le jeu est limité à 18 quêtes à la fois.

Attributs

- `surface: pygame.Surface` → Surface qui permet d'afficher les quêtes.
- `mono: pygame.font.Font` → Oui, encore la police monospace, elle permet d'écrire les quêtes.
- `quests: Dict[str, str]` → Attributs qui stock les quêtes en associant un nom de quêtes (définit dans un script coder en Adventure Script).

Méthodes

- `resize(size: Tuple[int, int])` → permet de changer la taille de la surface des quêtes.
- `add(var: str, event: str)` → Ajoute une quêtes et la place dans la variable `quests` en plaçant `var` comme clé et `event` comme valeur.
- `done(questVar: str)` → Notifie qu'une quêtes est bel et bien terminé.
- `clear()` → efface les quêtes
- `update()` → met à jour la surface des quêtes.
- `get_surface()` → retourne la surface de quête.

2.1.6 Programme principale

Ici selons moi, le programme principale ne se trouve pas dans le fichier tfe.py (qui est juste un simple lanceur), mais plutôt dans le fichier game/game.py. Voici en quelques mots toutes les choses mise en place au démarrage de l'application.

- Initialisation du moteur pygame, création de la fenêtre principale, initialisation de l'horloge interne à l'application.
- Initialisation des différentes parties du moteur (terminal, quête et rpg).

— Appel et transfer des parties du moteur (terminal, quête, rpg) à l'interpréteur Adventure Shell et execute un script (peux varier selons la modification de l'utilisateur).
— répartis les évennement (clavier et souris) entre chaque parties du jeu.

Deuxième partie

Conclusions et Sources

2.2 Problèmes rencontré

Heureusement pour moi, j'ai pas réellement rencontré de problème majeur à l'exception d'un problème mineur avec le curseur du terminal (=blink), régler grâce aux bon conseils de Monsieur Carrera (Merci :)).

2.3 Pour conclure...

Mon application répond à mon idée de base, et même va légerement plus loin que mes espérences. J'ai réussi mon défi de créer un langage de Script. Par contre j'ai pas eu le temps de faire un petit niveau, par contre une petite démo des possibilitées du moteur serait peut-être disponible à la présentation (croisons les doigts).

Chapitre 3

Sources

Documentation de Python3

https ://docs.python.org/3/tutorial/datastructures.html

https ://docs.python.org/3/howto/functional.html

https ://docs.python.org/3.5/library/functions.html#globals

StackOverflow / Ou semblables

https ://stackoverflow.com/questions/22992814/pygame-is-there-any-easy-way-to-find-the-letter-number-of-any-alphanumeric-pres

https ://stackoverflow.com/questions/24923078/python-keydown-combinations-ctrl-key-or-shift-key

https ://stackoverflow.com/questions/4408377/how-can-i-get-terminal-output-in-python

https ://stackoverflow.com/questions/3845423/remove-empty-strings-from-a-list-of-strings

https ://stackoverflow.com/questions/6825994/check-if-a-file-is-open-in-python

https ://stackoverflow.com/questions/500864/case-insensitive-python-regular-expression-without-re-compile

https ://stackoverflow.com/questions/30687783/create-custom-language-in-visual-studio-code

https ://stackoverflow.com/questions/8718885/import-module-from-string-variable

https ://stackoverflow.com/questions/12643009/regular-expression-for-floating-point-numbers

https ://gamedev.stackexchange.com/questions/47901/pygame-set-colorkey-transparency-issues

https ://stackoverflow.com/questions/6239769/how-can-i-crop-an-image-with-pygame#6240095

https ://stackoverflow.com/questions/5844672/delete-an-element-from-a-dictionary

ProgramCreek.com

https ://www.programcreek.com/python/example/50/subprocess.Popen

Documentation de Pygame

https ://www.pygame.org/docs/ref/rect.htm

https ://www.pygame.org/docs/ref/draw.html

https ://www.pygame.org/docs/ref/surface.html

https ://www.pygame.org/docs/ref/key.html

https ://www.pygame.org/docs/ref/sprite.html

https ://www.pygame.org/docs/ref/image.html

https ://www.pygame.org/docs/ref/mouse.html

https ://www.pygame.org/docs/ref/transform.html

Super Mario Bros par Alex Roşca

https ://github.com/roscale/SuperMarioBros/blob/master/tfe.py

Github de Sly

https ://github.com/dabeaz/sly