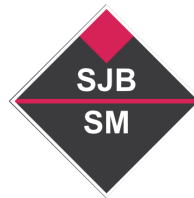


INSTITUT SAINT JEAN BERCHMANS - SAINTE MARIE
SECTION INFORMATIQUE



PROGRAMMATION
THE SHELL ADVENTURE

*Travail de fin d'études
réalisé par
Jordan Dalcq*

ANNÉE ACADÉMIQUE 2018 - 2019

Table des matières

I	Introduction et Analyse	4
1	Présentation	6
1.1	Fonctionnement	6
1.2	Interface utilisateur	7
2	Analyse	8
2.1	game.mechanics.term.Term	8
2.2	game.mechanics.term.History	10
2.3	game.mechanics.rpg.Rpg.Rpg	11
2.4	game.mechanics.rpg.dialog.Dialog	12
2.5	game.mechanics.rpg.sprite.Sprite	13
2.6	game.mechanics.quest.Quest.Quest	15
2.7	game.lang.Interpreter	16
2.8	Programme principale	18
2.9	Quelques fonctions importantes	18
II	Conclusions et Sources	19
2.10	Problèmes rencontré	20
2.11	Pour conclure...	20
3	Sources	21

Remerciements

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

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

Je remercie Alex Roşca, d'avoir été mon premier ami, dans cette grande aventure. Merci de m'avoir permis de découvrir la programmation

Je remercie Monsieur David Carrera pour avoir été le seul professeur qui a été capable de me pousser au meilleur dans ma passion.

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

J'aimerais aussi remercier Madame Challon pour m'avoir aidé à la rédaction de ce rapport.

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 implémenté un langage de programmation afin de laisser à l'utilisateur une liberté de créer des niveaux à sa guise.

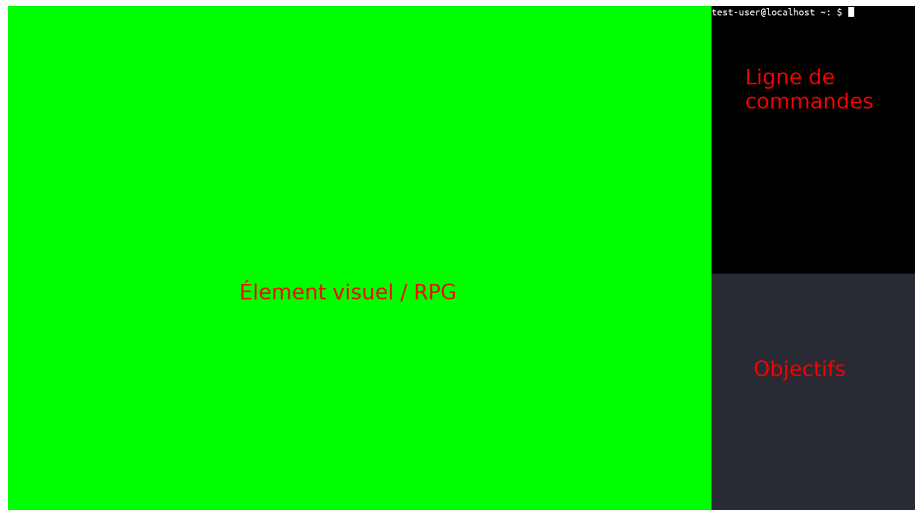


FIGURE 1.1 – Capture d’écran de la disposition par défaut

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 objets (armoire, chaise, garage, maison ...).

Ligne de commandes : Permet d’introduire des lignes de commandes BASH, 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 l’utilisateur l’entend.

Chapitre 2

Analyse

2.1 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 initialisée 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 caractères 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` → Stocke la dernière entrée utilisateur
- `blinkX` : `int` → Position x du rectangle clignotant à côté du prompt (=Blink).
- `custom` : `str` → Stocke les décorateurs 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, du nom de la machine et du CWD (Current Work Directory, = Dossier de travail courant).
- `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 décorateur (=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 récupérer 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, puis ensuite les affiche 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.

2.2 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 sauvegardés de façon à ce que l'utilisateur puisse retrouver ses commandes de jeu qu'il a précédemment entrées.

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 précédemment introduite.
- `get_next()` → Retourne la commande qui suit la dernière retournée.

2.3 game.mechanics.rpg.Rpg.Rpg

La classe `Rpg` permet d'afficher les éléments visuels, qu'ils soient 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 complètement 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 sprites (= personnage, objet).
- `update()` → Actualise la surface et applique les sprites à la surface.
- `resize(size: Tuple[int, int])` → Change la taille de la surface.

2.4 game.mechanics.rpg.dialog.Dialog

Cette classe permet de gérer les boîtes de dialogue des personnages / objets

Attributs

- `font: pygame.font.Font` → Police de caractères qui gère les dialogues, par défaut Monospace, mais il y a moyen de transmettre une autre police de caractères par le constructeur (argument `font`)
- `surface: pygame.Surface` → Surface du dialogue
- `haveToContinue: bool` → Permet d'afficher une petite flèche qui signale qu'une boîte de dialogue va suivre.
- `txt: str` → Contient le texte à afficher
- `screenHeight: int` → Longueur de l'écran
- `img` → Image qui montre le personnage qui parle.

Méthodes

- `update()` → Redessine la boîte de dialogue
- `get_surface() : pygame.Surface` → retourne la surface de la boîte de dialogue
- `get_pos(): Tuple[int, int]` → Permet de retourner la position de la boîte de dialogue
- `keydown(keydown: int)` → Vérifie si le joueur a bien appuyé sur la touche Enter pour enlever le dialogue en cours

2.5 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 dédiée 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.

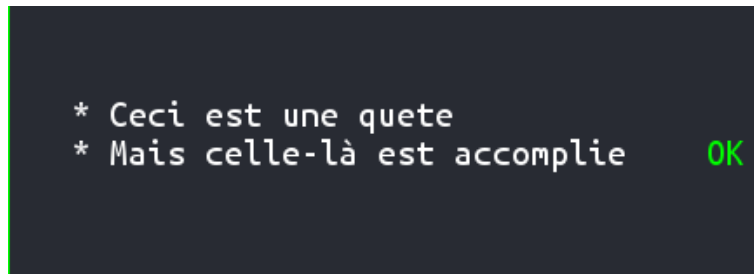


FIGURE 2.2 – Exemple de quêtes

2.6 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 simultanées.

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 stocke les quêtes en associant un nom de quête (défini 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 quête et la place dans la variable `quests` en plaçant `var` comme clé et `event` comme valeur.
- `done(questVar: str)` → Notifie qu'une quête est bel et bien terminée.
- `clear()` → efface les quêtes.
- `update()` → met à jour la surface des quêtes.
- `get_surface()` → retourne la surface de quête.

2.7 game.lang.Interpreter

Il s'agit du coeur même du langage Adventure Script. Avec l'aide du Lexer (partie du code qui crée les mots clé) et le Parser (partie du code qui définit la grammaire), il transforme le code défini par l'utilisateur en code compréhensible pour l'interpréteur Python.

Attributs

- `gameDir`: `str` → Contient le chemin vers le dossier du jeu
- `term`: `game.mechanics.term.Term` → Notre terminal (expliqué précédemment)
- `quest`: `game.mechanics.quest.Quest` → Nos quêtes (expliqué précédemment)
- `rpg`: `game.mechanics.rpg.Rpg` → Le RPG (expliqué précédemment)
- `lexer`: `game.lang.Lexer` Objet qui reprend les mots-clés du langage grâce aux regex. (= `sly.Lexer`)
- `parser`: `game.lang.Parser` Objet qui crée la grammaire du langage. (= `sly.Parser`)
- `codeTree`: `list` → Liste des instructions représentées en groupements de mots-clés simple à traiter
- `mainPath`: `str` → Reprend un dossier défini par l'utilisateur du langage là où il veut charger des sprites (=personnage, objet)
- `string`: `Tuple[int, str, float]` → Permet de faire une petite animation avec le texte (imitant quelqu'un qui tape sur un clavier) à vitesse et texte définis dans un script. Le premier composant est un nombre entier qui représente l'heure et la date du système, la chaîne de caractères est le texte à afficher et le nombre flottant représente le temps à attendre par caractères.
- `timing`: `int` variable qui permet de faire attendre le jeu (instruction WAIT)
- `typeBuffer` → Reprend les lettres à afficher dans l'animation de dactylographie.
- `buffer` : `dict` → Dictionnaire complexe qui reprend les fichiers et dossiers à attendre (supression ou création), lié à une fonction pour vérifier leur existence ou non. Lie le nom du fichier / dossier à des instructions Adventure Script.
- `dialog_buffer`: `List[game.mechanics.rpg.dialog.Dialog]` → Liste qui contient les dialogues pas encore affichés.
- `spriteMove`: `list` → Gère les déplacements des sprites (=personnage, objet)
- `sprites`: `Dict[str, game.mechanics.rpg.sprites.Sprite]` → Contient les sprites disponible sur l'écran et les lie à un nom afin de mieux les contrôler dans le langage.
- `pos`: `Dict[str, Tuple[int, int]]` → Contient les positions (à transmettre) des différentes parties du jeu (term, quest, rpg)
- `backPos`: `Dict[str, Tuple[int, int]]` → Contient les dispositions d'origine des différentes partie du jeu (term, quest, rpg)

Méthodes

- `parse(line: str)` → transforme une ligne de code en Adventure Shell en langage plus compréhensible pour Python
- `parse_string(string: str)` → Place les variables dans la chaîne de caractères (il faut mettre un \$ devant)
- `evaluate(code: list)` → Exécute les instructions données de l'utilisateur, traduites grâce au parser
- `execute(filename: str)` → Lit un fichier de script ligne par ligne et transmet les lignes au parser
- `mainloop()` → Vérifie les conditions du langage, vérifie si l'utilisateur passe d'un dialogue à l'autre, si un personnage, ou objet se déplace, gère les animations de dactylographie, répartit l'exécution et transmet les entrées utilisateurs au programme.

2.8 Programme principale

Ici selon 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 mises 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 transfert des parties du moteur (terminal, quête, rpg) à l'interpréteur Adventure Shell et exécute un script (peut varier selon la modification de l'utilisateur).
- répartit les événements (clavier et souris) entre chaque parties du jeu.

2.9 Quelques fonctions importantes

- `hex_convert(hexColor: str) : Tuple[int, int, int] →` Convertit une couleur hexadécimale en valeur RGB
- `execute(cmd: str, term: object) : str →` fonction qui exécute les commandes bash entrées par l'utilisateur

Deuxième partie

Conclusions et Sources

2.10 Problèmes rencontré

Heureusement pour moi, je n'ai pas réellement rencontré de problème majeur au niveau du code, juste d'un problème mineur avec le curseur du terminal (=blink), rapidement réglé grâce aux bons conseils de Monsieur Carrera (Merci :)).

Par contre au niveau du rapport, mon orthographe a été l'une de mes grandes difficultés.

2.11 Pour conclure...

Mon application correspond à mon idée de base, et va même légèrement plus loin que mes espérances. J'ai réussi mon défi de créer un langage de Script. Par contre je n'ai pas eu le temps de faire un petit niveau, ce pendant, une petite démo des possibilités du moteur sera 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>