The Shell Adventure

Jordan Dalcq

2018 - 2019

Table des matières

I Introductions **4**

1 Présentation **6**
1.1 Fonctionnement 6
1.2 Élement 6
1.3 Interface utilisateur 7

2 Analyse **8**
2.1 Classes 8
2.1.1 game.mechanics.term.Term 8

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

Introductions

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 une liberté de choisir le pouvoir de chaque commande à l'utilisateur

1.2 Élement

Énumérez et décrivez, dans l'ordre et en français, les différentes parties de votre application. Il s'agit de présenter les différents éléments (ainsi que leurs caractéristiques) qui sont amenés à interagir au sein de votre programme.

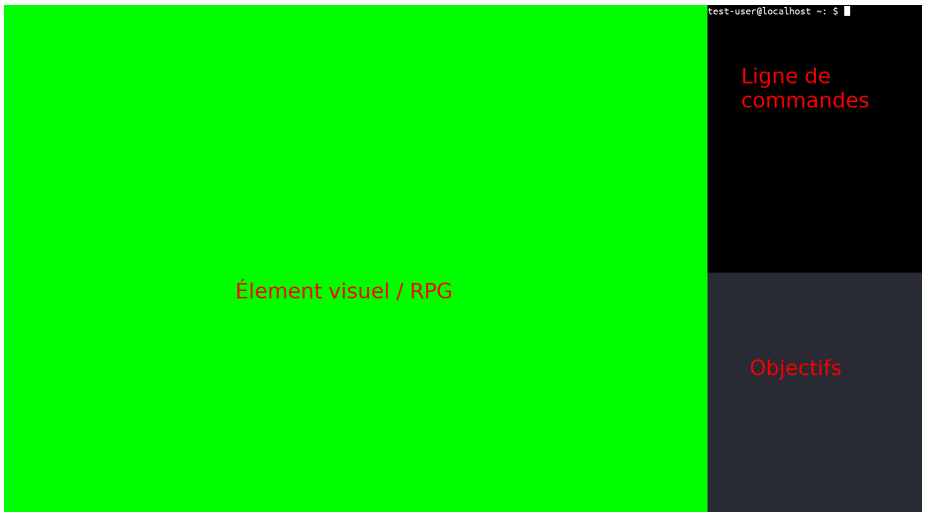

FIGURE 1.1 – Capture d'écran de la disposition par défault

1.3 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 et entrée standard pour l'utilisateur, 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
- `lineRect : pygame.Rect` → Rectangle de l'entrée utilisateur
- `blinkRect : pygame.Rect` → Rectangle avec le rectangle clignotant
- `fontSurface : pygame.Rect` → Rectangle 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 ou pas
- `currentTyping : str` → Stock l'entrée utilisateur
- `blinkX : int` → Position x du rectangle clignotant à côté du prompt
- `custom : str` → Stock les prompts customisés
- `history : mechanics.term.History` → Stock les prompts customisés
- `env : Dict[str, str]` → Variables d'environnements semblables à celles présentes dans les systèmes UNIX

— `prompt: str` → Permet de prendre connaissance du nom d'utilisateur, nom de la machine et le CWD (Current Work Directory)
— `tick: float` → Heure actuel pour permettre de réguler la vitesse du blink (le rectangle à côté du prompt)