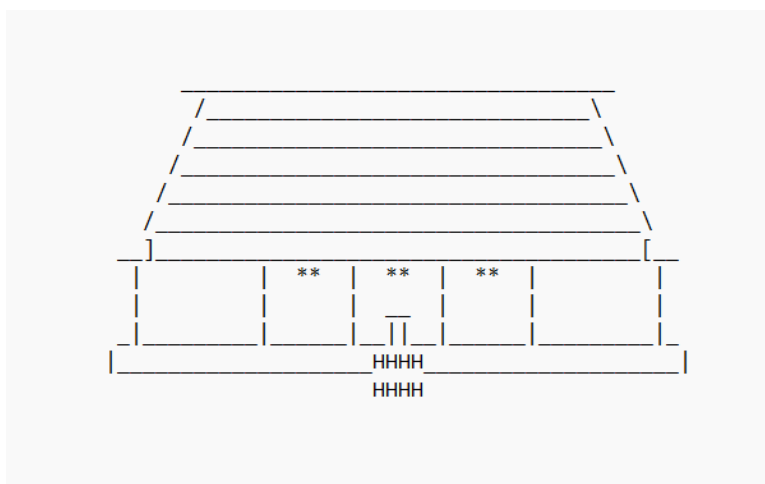


CONCEPTION



FIGHTERS CONCEPTION

Type	Conception
Nom du projet	Fighters
Commentaire	S2, ENIB
Auteur	Nicolas Quéré
Version	2.0
Date	29/05/2024

Table des matières

1 Rappel du cahier des charges.....	.4
-------------------------------------	----

1.1 Contraintes techniques.....	.4
1.2 Fonctionnalités.....	.4
1.3 P1 :Prototype P1.....	.5
1.4 P2 :Prototype P2.....	.5
2 Principes des solutions techniques adoptées.....	.6
2.1 Langage.....	.6
2.2 Architecture du logiciel6
2.3 Interface utilisateur.....	.6
2.3.1 Boucle de simulation.....	.6
2.3.2 Affichage.....	.6
2.3.3 Gestion du clavier.....	.6
2.3.4 Image ascii-art.....	.6
3 Analyse de conception.....	.7
3.1 Analyse noms/verbes :.....	.7
3.2 Types de donnée.....	.7
3.3 Dépendance entre modules.....	.7
3.4 Analyse descendante :.....	.8
3.4.1 Arbre principal :.....	.8
3.4.2 Arbre affichage.....	.8
3.4.3 Arbre interaction.....	.8
4 Description des fonctions.....	.9
4.1 Programme Principal : mainter.py.....	.9
4.2 aniter.py.....	.10
4.3 grilleter.py et Background.py.....	.11

1 Rappel du cahier des charges

1.1 Contraintes techniques

- Le logiciel est associé à un cours, il doit donc fonctionner sur les machines de TP de l'ENIB pour que les élèves puissent le tester.
- Le langage utilisé en cours est Python. Le développement devra donc se faire en python.
- Les notions de programmation orientée objet n'ayant pas encore été abordées, le programme devra essentiellement s'appuyer sur le paradigme de la programmation procédurale.
- Le logiciel devra être réalisé en conformité avec les pratiques préconisées en cours de MDD : barrière d'abstraction, modularité, unicode, etc...
- L'interface sera réalisée en mode texte dans un terminal

1.2 Fonctionnalités

- F1 : lancement du jeu
- F2 : jouer
 - F2.1 Jouer une manche
 - F2.1.1 Afficher le jeu :
 - ▶ dojo
 - ▶ les vies
 - ▶ joueur
 - ▶ adversaire
 - F2.1.2 Se déplacer dans le dojo
 - F2.1.3 frapper
 - F3.1.4 frappe et déplacement ordinateur
 - F3.1.5 Quitter le jeu avant la fin
 - F2.2 :Finir partie
 - F2.2.1 : Afficher résultat (victoire/défaite)
- F3: Quitter

1.3 P1 :Prototype P1 #####

Ce prototype porte essentiellement sur la création de l'espace de jeu et son utilisation.

Mise en œuvre des fonctionnalités : F1 et F2.1.1

Livré dans une archive au format .zip ou .tgz

Contient un manuel d'utilisation dans le fichier readme.txt

1.4 P2 :Prototype P2 #####

Ce prototype réalise toutes les fonctionnalités.

Livré dans une archive au format .zip ou .tgz

Contient un manuel d'utilisation dans le fichier README.txt

2 Principes des solutions techniques

2.1 Langage

Conformément aux contraintes énoncées dans le cahier des charges, le codage est réalisé avec langage python.

2.2 Architecture du logiciel

Nous mettons en œuvre le principe de la barrière d'abstraction. Chaque module correspond à un type de donnée et fournit toutes les opérations permettant de le manipuler de manière abstraite.

2.3 Interface utilisateur

L'interface utilisateur se fera via un terminal de type linux.

Nous reprenons la solution donnée en cours de IPI en utilisant les modules :
`termios`, `sys`, `select`.

2.3.1 Boucle de simulation

Le programme mettra en œuvre une boucle de simulation qui gérera l'affichage et les événements clavier.

2.3.2 Affichage

L'affichage se fait en communiquant directement avec le terminal en envoyant des chaînes de caractères sur la sortie standard de l'application.

2.3.3 Gestion du clavier

L'entrée standard est utilisé pour détecter les actions de l'utilisateur.

Le module `tty` permet de rediriger les événements clavier sur l'entrée standard.

Pour connaître les actions de l'utilisateur il suffit de lire l'entrée standard.

2.3.4 Image ascii-art

Pour dessiner certaines parties de l'interface nous utilisons des « images ascii ».

Dans l'idée de séparer le code et les données, les différentes images ASCII seront stockées dans des fichiers textes : `dojo.txt`, `joueur.txt`, `joueur1.txt`,
`joueur_coup_droit.txt`, `ennemie.txt`, `ennemi1.txt`,
`ennemie_coup_droit.txt`

3 Analyse

3.1 Analyse noms/verbes :

- Verbes :

nommer, choisir, jouer, afficher, déplacer, poser, finir, quitter

- Nom :

joueur, dojo, nom, vie, case, combattant, manche, utilisateur, ordinateur, partie, résultat

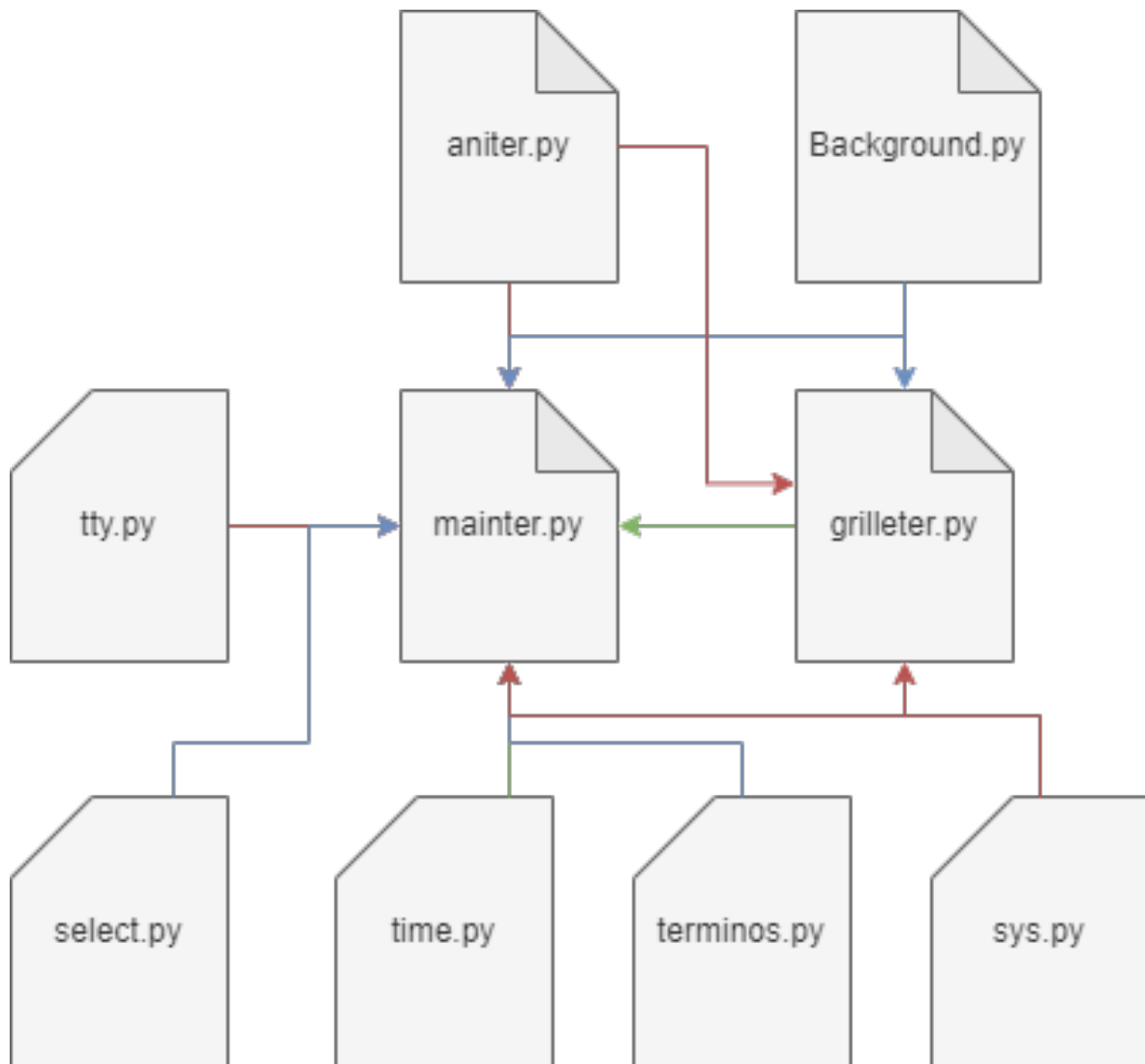
3.2 Types de donnée

```
type: Joueur = struct
    color      : entier
    x          : entier
    y          : entier
    vie        : entier
    i          : entier
    etat1      : liste
    etat2      : liste
    coup       : liste
fstruct

type: Background = struct
    map        : liste de liste
fstruct

type: Grille = struct
    identification : liste
    affichage      : liste
```

3.3 Dépendance entre modules



3.4 Analyse descendante :

3.4.1 Arbre principal :

```
mainter()  
+-- mainter.init()  
|   +-- aniter.create()  
|   +-- Background.create()  
|   +-- grilleter.creer_grille()  
|  
+-- mainter.run()  
    +-- mainter.ia_action()  
    +-- mainter.interact()  
    +-- mainter.show()
```

3.4.2 Arbre affichage

```
grilleter.show()
```

3.4.3 Arbre interaction

```
mainter.interact ()  
|  
+-- mainter.isData()  
|  
+-- mainter.dammage()
```


4 Description des fonctions

4.1 Programme Principal : mainter.py

- `mainter.init()`
- `mainter.collission()`
- `mainter.dammage()`
- `mainter.interact()`
- `mainter.ia_action()`
- `mainter.isData()`
- `mainter.show()`
- `mainter.run()`
- `mainter.quitGame()`

`mainter.init()` ->rien

Description : initialisation du jeu

Paramètres : aucun

Valeur de retour :aucune

`mainter.collission()` ->rien

Description : vérification si le joueur ne sort pas de la carte

Paramètres : dict,str

Valeur de retour : bool

`mainter.dammage()` ->rien

Description : vérification s'il y a une frappe réussite

Paramètres : dict,str

Valeur de retour :bool

`mainter.interact()` ->rien

Description : gère les événements clavier

Paramètres : dict,int

Valeur de retour :aucune

`mainter.ia_action()` ->rien

Description : gère la réponse de l'ia

Paramètres : dict

Valeur de retour :aucune

`mainter.isData()` ->rien

Description : récupération des événements claviers

Paramètres : aucun

Valeur de retour : srt

`mainter.show()` ->rien

Description : affiche le jeu

Paramètres : dict

Valeur de retour :aucune

`mainter.run()` ->rien

Description : boucle de simulation

Paramètres : dict

Valeur de retour :aucune

`mainter.quitGame()` ->rien

Description : quitte le jeu en affichant le résultat

Paramètres : str

Valeur de retour : aucune

4.2 aniter.py

- `aniter.apparence()`
- `aniter.create()`

`aniter.apparence()` -> rien

Description : crée une liste de liste avec un txt

Paramètres : str

Valeur de retour : liste

`aniter.create()` -> rien

Description : crée un nouveau combattant

Paramètres : int,int,int,str,int

Valeur de retour : struc

4.3 grilleter.py et Background.py

- grilleter.**creer_grille()**
- grilleter.**show()**
- Background.**create_bg()**

grilleter.**creer_grille()** -> rien
Description: Crée une grille
Paramètres: liste, struc, struc
Valeur de retour: struc

grilleter.**show()** -> rien
Description: montre la grille
Paramètres: dict
Valeur de retour: rien

Background.**create_bg()** -> rien
Description: Crée un background
Paramètres: str
Valeur de retour: struc