École Nationale des Ingénieurs de Brest

Document de Conception

mdd-projet

Tower Defense

Nathan Calvarin et Maxime Delin

Table des matières

1 Rappel du cahier des charges

1.1 Contraintes techniques

- Le logiciel est associé à un cours, il doit fonctionner sur les machines de TP de l'ENIB pour que les élèves puissent les tester.
- Le langage utilisé 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 : Nommer le joueur
- F2 : Choisir le niveau
- F3 : Jouer une partie
  - F3.1 : Joueur un niveau
    - F3.1.1 Afficher le jeu
      - map
      - nom
      - niveau
      - score
      - case sélectionnée
      - nombre de monstres restants
      - différentes tours disponibles
      - argent
    - F3.1.2 Sélectionner une tour
    - F3.1.3 Se déplacer dans la map
    - F3.1.4 Placer une tour
    - F3.1.5 Améliorer une tour
    - F3.1.6 Finir manche
  - F3.2 Finir partie
    - F3.2.1 Afficher le résultat
    - F3.2.2 Quitter

1.3 P1 : Prototype P1

Ce prototype porte essentiellement sur la création de la map et sur l'affichage.
Mise en oeuvre des fonctionnalités : F1, F2, F3.1.1, F3.1.2, F3.1.3, F3.1.4, F3.1.5
Livré dans un 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.
Ajout à P1 des fonctionnalités F3.1.6, F3.2
Livré dans un archive au format *.zip* ou *.tgz*
Contient un manuel d'utilisation dans le fichier *readme.txt*

2 Principes des solutions techniques adoptées

2.1 Langage

Conformément aux contraintes énoncées dans le cahier des charges, le codage est réalisé avec langage python. Nous choisissons la version 2.7.5

2.2 Architecture du logiciel

Nous mettons en oeuvre 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 MDD en utilisant les modules :*termios*,*sys*, *select*.

2.3.1 Boucle de simulation

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

2.3.2 Affichage

L'affichage se fait en communicant 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 : *blalalalal.txt*, *bkkgjmg.txt* .....

2.4 Map, cases et tours...

Pour modéliser la *map* , une liste de liste (m*n) permet de stocker des caractères correspondant au tour posées sur la *map*. Cette liste de liste sera crée à partir d'un fichier .txt en utilisant les fonction split().

3 Analyse

3.1 Analyse noms/verbes

Verbes :

nommer,choisir,afficher,déplacer,placer,améliorer,finir,quitter

Noms :

jouer,niveau,nom,map,monstre,curseur,tour,argent,case selectionnée,nombre de monstres restant

3.2 Type de donnée

```
            type : Game    = struct
               niveau            : int
               score             : Score
               playerName        : string

type : Monster = struct
               nombre de vie     : int
               caractère affiché : string
               gain rapporté     : int
               vitesse           : float
               type              : string

type : Tower  = struct
              type               : string
              portée             : int
              dommage            : int
              amélioration       : boolean
              caractère affiché  : string


type : Score  = struct
              nombre de vie      : int
              argent             : int
              nb de monstres tués: int
              nb vagues restantes: int
              niveau actuel      : int


type : Map    = struct
              cases              : liste de liste caractères m*n
              selected           : tuple (int,int)
```

3.3 Dépendance entre modules

3.4 Analayse descendante

3.4.1 Arbre principal

```
            Main.main()
 +--Main.init()
 |   +--Main.askName()
 |   +--Main.askLevel()
 |   +--Game.create()
 |       +--Level.create()
 |           +--Map.create()
 |           +--Score.create()
 |
 +--Main.run()0
     +--Main.interact()
     +--Main.move()
     +--Main.anim()
     +--Main.show()
```

```
    +--Main.timesleep()
```

3.4.2 Arbre affichage

```
            Main.show()
+--Game.show()
    +--Map.show()
    +--Monster.show()
    +--Tower.show()
    +--Score.show()
```

3.4.3 Arbre interaction

```
            Main.interact()
+--Game.move()
|   +--Map.getselected()
|   +--Map.setselected()
|   +--Monster.move()
|
+--Game.play()
|   +--Map.play()
|   +--Tower.play()
|
+--Monster.play()
|   +--Monster.damage()
|   +--Monster.spawn()
|   +--Monster.die()
|   +--Monster.effect()
|
+--Main.finish()
```

4 Description des fonctions

4.1 Programme principal : Main.py

- Main.**main()**
- Main.**init()**
- Main.**run()**
- Main.**show()**
- Main.**interact()**
- Main.**move()**
- Main.**timesleep()**
- Main.**anim()**
- Main.**askName()**

- Main.**askLevel()**
- Main.**finish()**

4.2 Game.py

- Game.**create()**
- Game.**move()**
- game.**show()**

4.3 Map.py

- Map.**create()**
- Map.**getselected()**
- Map.**setselected()**
- Map.**play()**
- Map.**show()**

4.4 Level.py

- Level.**create()**

4.5 Monster.py

- Monster.**move()**
- Monster.**play()**
- Monster.**damage()**
- Monster.**spawn()**
- Monster.**die()**
- Monster.**effect()**

4.6 Score.py

- Score.**create()**
- Score.**show()**

4.7 Tower.py

- Tower.**play()**
- Tower.**show()**
- Tower.**play()**

5 Calendrier et suivi de développement

5.1 P1

5.1.1 Fonctions à développer

5.1.2 Autre

5.2 P2

5.2.1 Fonctions à développer