

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	2
1.1	Contraintes techniques	2
1.2	Fonctionnalités	2
1.3	P1 : Prototype P1	2
1.4	P2 : Prototype P2	2
2	Principes des solutions techniques adoptées	3
2.1	Langage	3
2.2	Architecture du logiciel	3
2.3	Interface utilisateur	3
2.3.1	Boucle de simulation	3
2.3.2	Affichage	3
2.3.3	Gestion du clavier	3
2.3.4	Image ascii-art	3
2.4	Map, cases et tours...	3
3	Analyse	3
3.1	Analyse noms/verbes	3
3.2	Type de donnée	4
3.3	Dépendance entre modules	4
3.4	Analayse descendante	5
3.4.1	Arbre principal	5
3.4.2	Arbre affichage	5
3.4.3	Arbre interaction	5
4	Description des fonctions	6
4.1	Programme principal : Main.py	6
4.2	Game.py	7
4.3	Map.py	7
4.4	Level.py	8
4.5	Monster.py	8
4.6	Score.py	9
4.7	Tower.py	9
5	Calendrier et suivi de développement	10
5.1	P1	10
5.1.1	Fonctions à développer	10
5.1.2	Autre	10
5.2	P2	10
5.2.1	Fonctions à développer	10

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 : Jouer 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 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 : *blalalalal.txt*, *bkkgjmg.txt*

2.4 Map, cases et tours...

Pour modéliser la *map* , une liste de liste ($m \times n$) permet de stocker des caractères correspondant aux tours posées sur la *map* ou à une case vide. Cette liste de liste sera créée 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 sélectionnée,nombre de monstres restant

3.2 Type de donnée

```

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

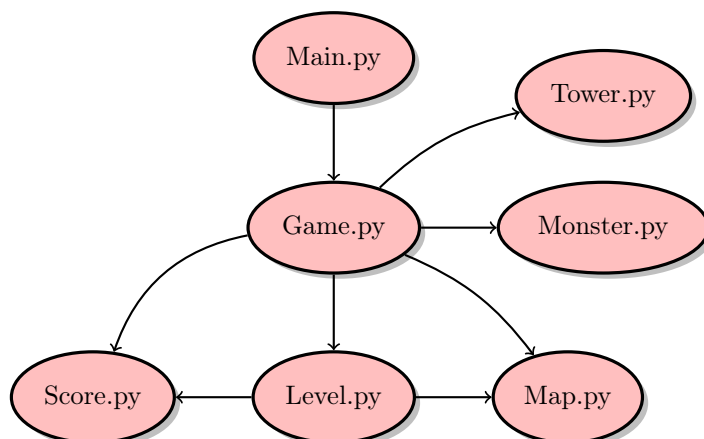
type : Monster = struct
    nombre de vie : int
    caractère affiché : string
    gain rapporté : int
    vitesse       : float
    type          : string

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 Analyse descendante

3.4.1 Arbre principal

```
Main.main()
+--Main.init()
|   +--Main.askName()
|   +--Main.askLevel()
|   +--Game.create()
|       +--Level.create()
|           +--Map.create()
|           +--Score.create()
|
+--Main.run()
    +--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()
```

```
Mainmain()->rien  
Description : fonction principale du jeu  
Paramètres : aucun  
Valeur de retour : aucune
```

```
Maininit()->return  
Description : initialisation du jeu  
Paramètres : aucun  
Valeur de retour : aucune
```

```
Main.run()->rien  
Description : boucle de simulation  
Paramètres : aucun  
Valeur de retour : aucune
```

```
Main.show()->rien  
Description : affiche le jeu dans le terminal  
Paramètres : aucun  
Valeur de retour : aucune
```

```
Main.interact()->rien  
Description : gère les événements clavier  
Paramètres : aucun  
Valeur de retour : aucune
```

```
Main.anim()->rien  
Description : animation des tirs des tours  
Paramètres : aucun  
Valeur de retour : aucune
```

```
Main.askName()->chaîne  
Description : demande le nom de l'utilisateur  
Paramètres : aucun  
Valeur de retour : le nom du joueur
```

```
Main.askLevel()->entier  
Description : demande à l'utilisateur le niveau auquel il souhaite jouer  
Paramètres : aucun  
Valeur de retour : niveau sélectionné
```

`Main.finish()`->rien

Description : termine ma partie, affiche la fin du niveau avec le score du joueur et sauvegarde

Paramètres : aucun

Valeur de retour : aucune

4.2 Game.py

`Game.create()`

`Game.move()`

`Game.show()`

`Game.create()`->`Game`

Description : crée une nouvelle partie

Paramètres :

g : `Game`

name : chaîne

Valeur de retour : nouvelle partie

`Game.move()`->return

Description : description

Paramètres : param

Valeur de retour : return

`Game.show()`->rien

Description : affiche le visuel de la partie en cours

Paramètres : g : `Game`

Valeur de retour : aucune

4.3 Map.py

`Map.create()`

`Map.getSelected()`

`Map.setSelected()`

`Map.play()`

`Map.show()`

`Map.create()`->liste de liste

Description : crée la *map* à partir d'un fichier .txt

stocker dans le module `Level` et à l'aide de la fonction `split()`

Paramètres : *fichier.txt*

Valeur de retour : liste de liste

`Map.setSelected(m,i)`->rien

Description : définit la case sélectionnée

Paramètres : `Map`

index : tuple(entier,entier)

Valeur de retour : rien


```
Map.getSelected(m)->tuple(entier,entier)
    Description : renvoie la case sélectionnée
    Paramètres : Map
    Valeur de retour : index de la case sélectionnée

Map.play(g,p)->int
    Description : pose une tour dans la case selectionnée
    Paramètres : Map
                  tower : caractère
    Valeur de retour : 1 si la case est vide, 0 si la case est occupée ou
                      2 si la case est occupée par une tour qu'il est possible d'améliorer

Map.show(m)->rien
    Description : affiche la map
    Paramètres : m: map
    Valeur de retour : rien
```

4.4 Level.py

```
Level.create()

Level.create()->return
Description : crée un niveau
Paramètres : param
Valeur de retour : return
```

4.5 Monster.py

```
Monster.move()
Monster.play()
Monster.damage()
Monster.spawn()
Monster.die()
Monster.effect()

Monster.move()->?
    Description : déplace les monstres le d'un chemin prédéfini dans le fichier .txt
    Paramètres : ?
    Valeur de retour : ?

Monster.play()->?
    Description : ?
    Paramètres : ?
    Valeur de retour : ?

Monster.damage()->dommage subit
    Description : calcule le nombre de dommages subit par le monstre
    Paramètres : tour qui l'a touché
    Valeur de retour : int : dommage subit
```

`Monster.spawn()-> tuple(int,int)`

Description : déduit en fonction du fichier `.txt` la position où apparaissent les montres

Paramètres :

Valeur de retour : position d'apparition du monstre la carte

`Monster.die()->booléen`

Description : calcule en fonction du nombre de vies du monstres et les dommages subit

si le monstre est toujours vivant ou non. Renvoie 0 vivant, renvoie 2 mort.

Paramètres : nb vie du monstre

Valeur de retour : mort ou vivant

`Monster.effect()->?`

Description : calcule en fonction du type de tour ayant atteint le monstre si celui-ci est ralenti ou autre (empoisonné)

Paramètres : param

Valeur de retour : return

4.6 Score.py

`Score.create()`

`Score.show()`

`Score.create()->argent, nb de vagues restantes, nb vies restantes`

Description : crée un score qui indique la quantité d'argent et de monstres restant ainsi que le nombre de vie

Paramètres : ?

Valeur de retour : ?

`Score.show()->return`

Description : affiche le score

Paramètres : aucun

Valeur de retour : aucune

4.7 Tower.py

`Tower.play()`

`Tower.show()`

`Tower.play()`

`Tower.play()->?`

Description : ?

Paramètres : ?

Valeur de retour : ?

`Tower.show()->rien`

Description : affiche les tours

Paramètres : aucun

Valeur de retour : aucune

5 Calendrier et suivi de développement

5.1 P1

5.1.1 Fonctions à développer

fonctions	codées	testées	commentaires
Main. main()			
Main. init()			
Main. run()			
Main. show()			
Main. interact()			
Main. move()			
Main. timesleep()			
Main. anim()			
Main. askName()			
Main. askLevel()			
Game. create()			
Game. move()			
Game. show()			
Map. create()			
Map. getSelected()			
Map. setSelected()			
Map. play()			
Map. show()			
Level. create()			
Monster. move()			
Monster. play()			
Monster. damage()			
Monster. spawn()			
Monster. die()			
Score. create()			
Score. show()			
Tower. show()			

5.1.2 Autre

5.2 P2

5.2.1 Fonctions à développer

fonctions	codées	testées	commentaires
Main. finish()			
Monster. effect()			
Tower. play()			