

ÉCOLE NATIONALE DES INGÉNIEURS DE BREST

DOCUMENT DE CONCEPTION

MDD-PROJET

---

# Tower Defense

---

Nathan CALVARIN et Maxime DELIN

# Table des matières

<b>1</b>	<b>Rappel du cahier des charges</b>	<b>2</b>
1.1	Contraintes techniques . . . . .	2
1.2	Fonctionnalités . . . . .	2
1.3	P1 : Prototype P1 . . . . .	2
1.4	P2 : Prototype P2 . . . . .	2
<b>2</b>	<b>Principes des solutions techniques adoptées</b>	<b>3</b>
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
<b>3</b>	<b>Analyse</b>	<b>3</b>
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
<b>4</b>	<b>Description des fonctions</b>	<b>5</b>
4.1	Programme principal : Main.py . . . . .	5
4.2	Game.py . . . . .	6
4.3	Background.py . . . . .	7
4.4	Level.py . . . . .	7
4.5	Animat.py . . . . .	8
4.6	Tower.py . . . . .	8
<b>5</b>	<b>Calendrier et suivi de développement</b>	<b>9</b>
5.1	P1 : 14 Mai . . . . .	9
5.1.1	Fonctions à développer . . . . .	9
5.1.2	Autre . . . . .	9
5.2	P2 : 28 Mai . . . . .	9
5.2.1	Fonctions à développer . . . . .	9

# 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
      - case sélectionnée
      - nombre de monstres restants
      - tour posée
      - argent
    - \* F3.1.2 Placer une tour
    - \* F3.1.3 Tuer monstre
    - \* F3.1.4 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

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 F3.1.4, 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 utiliserons le module *ncurses*.

#### 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 *win.keypad* 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 texte : *image.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 *.xml* en utilisant les fonction *split()*.

## 3 Analyse

### 3.1 Analyse noms/verbes

Verbes :

nommer, choisir, afficher, déplacer, placer, 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

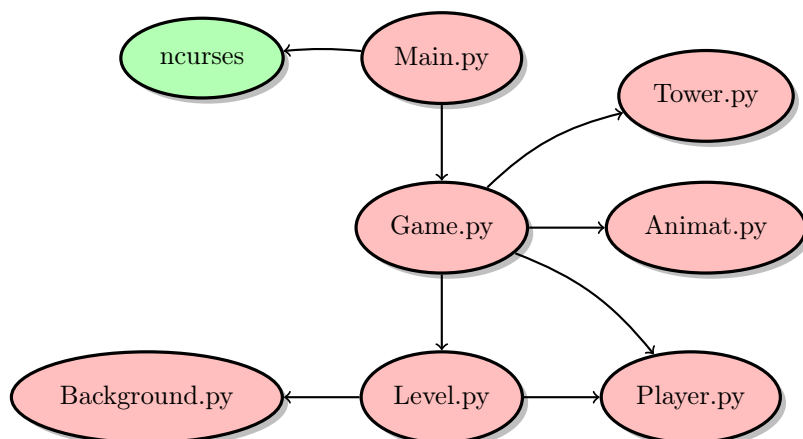
type : Monster = struct
    nombre de vie : int
    caractère affiché : string
    gain rapporté : int
    vitesse       : float
    type          : string

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()
  | +--Background.create()
  | +--Player.create()
  | +--Level.create()
  | +--Game.create()
  |
  +--Main.run()
  | +--Main.interact()
  |
  +--Main.interact()
  | +--Game.interact()
  |
  +--Main.show()
    +--Game.show()

```

### 3.4.2 Arbre affichage

```

Main.show()
  +--Game.show()
    +--Background.show()
    +--Game.showCursor()
    +--Tower.show()
    +--Animat.show()
    +--Player.show()

```

### 3.4.3 Arbre interaction

```

Main.interact()
  +--Game.interact()
    +--Game.cursorMove()
    +--Tower.shoot()
    +--Game.move()

```

## 4 Description des fonctions

### 4.1 Programme principal : Main.py

```

    Main.main()
Main.init()
Main.run()
Main.show()

```

```
Main.interact()
Main.refTime
```

```
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
```

## 4.2 Game.py

```
Game.create()

Game.move()
Game.show()
Game.interact()
Game.showCursor()
Game.cursorMove()

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

`Game.interact()`->?

`Game.showCursor()`->?

`Game.cursorMove()`->?

### 4.3 Background.py

`Background.create()`  
`Background.getSpawn()`  
`Background.setSpan()`  
`Background.getFinish()`  
`Background.setFinish()`  
`Background.show()`

`Background.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

`Background.getSpawn()`->rien  
 Description : définit la case sélectionnée  
 Paramètres : `Map`  
                   `index : tuple(entier,entier)`  
 Valeur de retour : rien

`Background.setSpawn`->?

`Background.getFinish`->?

`Background.setFinish`->?

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

### 4.4 Level.py



## 4.5 Animat.py

```
Animat.create()
Animat.show
Animat.colorMonster

Animat.create()-> ?

Animat.show()->?
    Description : ?
    Paramètres : ?
    Valeur de retour : ?

Animat.colorMonster()->?
```

## 4.6 Tower.py

```
Tower.set()
Tower.show()
Tower.showLife()
Tower.showMoney()

Tower.set()->?
    Description : ?
    Paramètres : ?
    Valeur de retour : ?

Tower.show()->rien
    Description : affiche les tours
    Paramètres : aucun
    Valeur de retour : aucune

Tower.showLife()->?

Tower.showMoney()->?
```

## 5 Calendrier et suivi de développement

### 5.1 P1 : 14 Mai

#### 5.1.1 Fonctions à développer

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

#### 5.1.2 Autre

Possibilité d'enregistrer son score dans un fichier *.xml*,

### 5.2 P2 : 28 Mai

#### 5.2.1 Fonctions à développer

fonctions	codées	testées	commentaires
Main. <b>finish()</b>			
Monster. <b>effect()</b>			
Tower. <b>play()</b>			