

CONCEPTION

ROGUE-LIGHT

Type	Conception
Nom du projet	Rogue
Commentaire	Cahier de Conception en cours
Auteur	Antonin Trévidy
Version	3
Date	10/06/2026

Table des matières

1 Rappel du cahier des charges.....	4
1.1 Contraintes techniques.....	4
1.2 Fonctionnalités.....	4
2 Principes des solutions techniques.....	5
2.1 Langage.....	5
2.2 Architecture du logiciel.....	5
2.3 Interface utilisateur.....	5
2.3.1 Boucle de simulation.....	5
2.3.2 Affichage.....	5
2.3.3 Gestion du clavier.....	5
2.3.4 Image ascii-art.....	5
2.4 Carte du monde.....	5
3 Analyse.....	6
3.1 Types de donnée.....	6
3.2 Dépendance entre modules.....	7
3.3 Analyse descendante :.....	8
3.3.1 Arbre principal :.....	8
3.3.2 Arbre affichage.....	8
3.3.3 Arbre boucle de simulation.....	8
3.3.4 Arbre interaction.....	9
4 Description des fonctions.....	10
4.1 Programme Principal : Main.py.....	10
4.2 Game.py.....	11
4.3 World.py.....	13
4.4 Boss.py.....	15
4.5 Room.py.....	16
4.6 Enemy.py.....	17

4.7 Object.py.....	18
5 Calendrier et suivi de développement.....	19
5.1 P1 :.....	19
5.1.1 fonctions à développer.....	19
5.1.2 autre.....	21

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.
- Le logiciel devra être réalisé en conformité avec les pratiques préconisées en cours de IPI : boucle de simulation, type abstrait de données etc...
- L'interface sera réalisée en mode texte dans un terminal

1.2 Fonctionnalités

- F0 : Démarrer une partie
- F1 : Déplacer le joueur
 - F1.1 : Ne pas traverser les murs
 - F1.2 : Afficher le monde en fonction des parties explorées
- F2 : Générer le monde
 - F2.1 : Générer les salles
 - Générer la structure et la position
 - Générer les objets/ennemis présents
- F3 : Interagir avec l'environnement
 - F3.1 : Affronter les ennemis
 - Rencontrer les petits ennemis
 - ▶ Lister les caractéristiques de chaque ennemi rencontré
 - Lancer le combat de boss
 - F3.2 : Récupérer les objets
 - Lister les objets récupérés et leurs statistiques
- F4 : Combat final
 - F4.1 : Changer de scène
 - F4.2 : Sélectionner ses attaques
 - F4.3 : Gagner le combat
 - Changer de monde
 - Changer de difficulté
- F5 : Perdre
- F6 : Afficher le jeu
 - F6.1 : Afficher le monde
 - F6.1 : Afficher l'interface utilisateur

2 Principes des solutions techniques

2.1 Langage

Conformément aux contraintes énoncées dans le cahier des charges, le codage est réalisé avec langage python 3.

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.

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.

2.4 Carte du monde

Pour modéliser le monde du jeu, une liste de liste (180X35) permet de stocker des caractères correspondant au joueur, aux murs, aux ennemis etc...

3 Analyse

3.1 Types de donnée

```
type: Game = struct
    world          : World
    boss           : Boss
    level          : int
    statistics     : dict
    mode           : str
fstruct

type: Boss = struct
    statistics     : dict
    player attacks : tuple

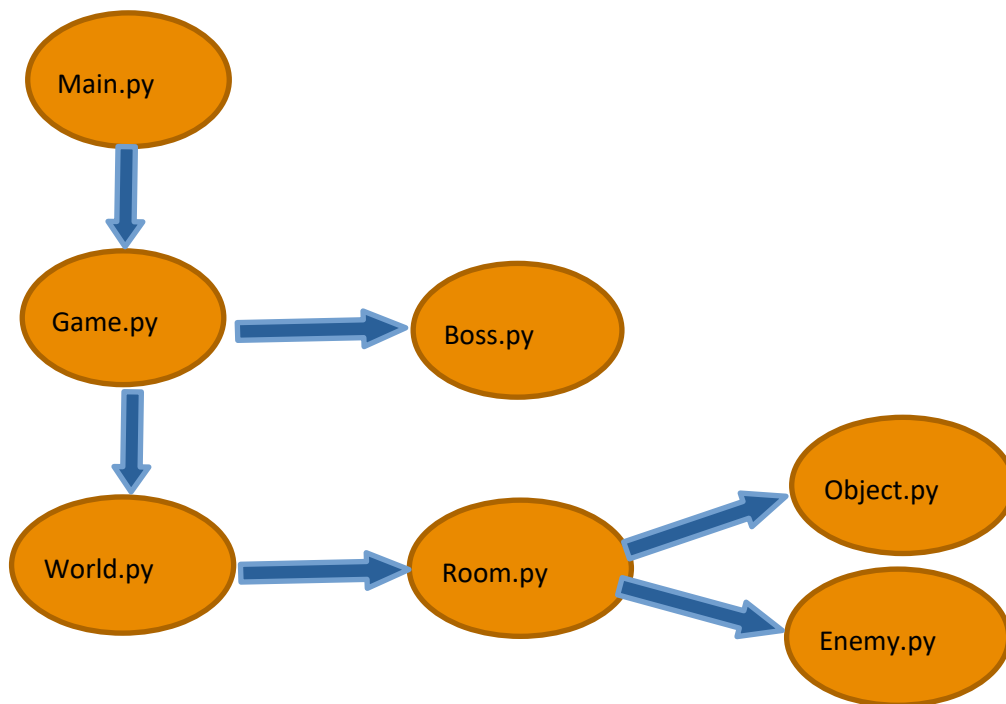
type: World = struct
    size           : (columns, lines)
    grid           : list of list of str/Enemy/Object
    rooms          : list of Room
    spawn point    : tuple (int,int)
    boss entry     : tuple (int,int)
fstruct

type: Rooms = struct
    level          : int
    coordinates    : tuple (int,int)
    gates          : tuple (int,int)
    size           : tuple (int,int)
    enemies        : list of Enemy
    objects        : list of Object
    type           : str
fstruct

type: Enemy = struct
    type           : str
    coordinates    : list [int,int]
    life           : int
    strength       : int
    speed          : int
fstruct

type: Object = struct
    type           : str
    coordinates    : tuple (int,int)
    strength       : int
    armor          : int
fstruct
```

3.2 Dépendance entre modules



3.3 Analyse descendante :

3.3.1 Arbre principal :

```
Main.main()
+-- Main.init()
|   +-- Game.create()
|       +-- World.create()
|           +-- Room.create()
|               +-- Enemy.create()
|               +-- Object.create()
+-- Main.run()
    +-- Main.interact()
    +-- Main.live()
    +-- Main.show()
```

3.3.2 Arbre affichage

```
Main.show()
+-- Game.show()
    +-- Boss.show()
    +-- World.show()
```

3.3.3 Arbre boucle de simulation

```
Main.live()
+-- Game.live()
    +-- World.live()
        +-- Game.movePlayer()
        +-- Enemy.move()
        +-- World.isTileEnemy()
            +-- World.getTileData()
        +-- Game.remeberEnemy()
        +-- Game.confrontEnemy()
            +-- Enemy.isWeaker()
        +-- Game.setStatistics()
        +-- Game.isPlayerDead()
            +-- Game.getStatistics()
        +-- Game.end()
```

3.3.4 Arbre interaction

```
Main.interact()  
  +-- Game.acceleratePlayer()  
  |   +-- World.isNextTileReachable()  
  |   |   +-- World.getTileData()  
  |   +-- World.discoverArea()  
  |  
  +-- Game.playerAction()  
  |   +-- World.isTileObject()  
  |   |   +-- World.getTileData()  
  |   |   +-- Game.addObject()  
  |   |  
  |   +-- Game.create()  
  |   |   +-- World.create()  
  |   |   |   +-- Room.create()  
  |   |   |   |   +--Enemy.create()  
  |   |   |   |   +--Object.create()  
  |   +-- Game.isPlayerDead()  
  |   +-- Game.end()  
  +-- Main.finish()
```

4 Description des fonctions

4.1 Programme Principal : Main.py

- `Main.main()`
- `Main.init()`
- `Main.run()`
- `Main.show()`
- `Main.interact()`
- `Main.finish()`

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

`Main.init()` -> rien
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 la partie
Paramètres : aucun
Valeur de retour : aucune

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

`Main.live(dt)` -> rien
Description : gère la boucle de simulation
Paramètres :
 `dt` : int
Valeur de retour : aucune

`Main.finish()` -> rien
Description : termine la partie
Paramètres : aucun
Valeur de retour : aucune

4.2 Game.py

- `Game.create(statistics, level)`
- `Game.getStatistics(g)`
- `Game.getLevel(g)`
- `Game.getMode(g)`
- `Game.getWorld(g)`
- `Game.getBoss(g)`
- `Game.setStatistics(g)`
- `Game.setLevel(g)`
- `Game.setMode(g)`
- `Game.setWorld(g)`
- `Game.setBoss(g)`
- `Game.isPlayerDead(g)`
- `Game.rememberObject(g, o)`
- `Game.remeberEnemy(g, e)`
- `Game.movePlayer(g, k)`
- `Game.moveEnemy(g, e)`
- `Game.confrontEnemy(g)`
- `Game.giveObjectEffect(g, o)`
- `Game.isNextTileReachable(g, coordinates, direction, acceptObject=True)`
- `Game.isTileObject(g)`
- `Game.isTileEnemy(g)`
- `Game.show(g)`

`Game.create(statistics, level)` -> Game
 Description : crée une nouvelle partie
 Paramètres :
 statistics : dict
 level : int
 Valeur de retour : nouvelle partie

`Game.isPlayerDead(g)` -> bool
 Description : regarde si le joueur a perdu toutes ses vies
 Paramètres :
 g : Game
 Valeur de retour: True ou False

`Game.rememberObject(g, o)` -> rien
 Description : rajoute l'objet à l'inventaire
 Paramètres :
 g : Game
 o : Object
 Valeur de retour: rien

Game.rememberEnemy(g, e) ->rien
 Description : rajoute l'objet au bestiaire
 Paramètres :
 g : Game
 e : Enemy
 Valeur de retour: rien

Game.movePlayer(g,k) ->rien
 Description : gère le déplacement du personnage sur la carte
 Paramètres :
 g : Game
 k : str
 Valeur de retour: rien

Game.moveEnemy(g, e) ->rien
 Description : gère le déplacement de l'ennemi dans sa salle
 Paramètres :
 g : Game
 e : Enemy
 Valeur de retour: rien

Game.confrontEnemy(g, e) ->rien
 Description : fait s'affronter le personnage contre un ennemi
 Paramètres :
 g : Game
 e : Enemy
 Valeur de retour: rien

Game.giveObjectEffect(g, o) ->rien
 Description : applique l'effet de l'objet au joueur
 Paramètres :
 g : Game
 o : Object
 Valeur de retour: rien

Game.isNextTileReachable(g, coordinates, direction, acceptObject=True) ->bool
 Description: regarde si la prochaine case est accessible avec la possibilité de choisir si le fait qu'il y ait un objet sur la case soit bloquant ou non
 Paramètres :
 g : Game
 coordinates : list [int, int]
 direction : list [int, int]
 acceptObject : bool → default : True
 Valeur de retour: accessibilité de la case d'arrivée

Game.isTileObject(g) ->bool
 Description: regarde si la case sur laquelle est le joueur est un objet
 Paramètres :
 g : Game
 Valeur de retour: True ou False

Game.**isTileEnemy(g)**->bool

Description: regarde si la case sur laquelle est le joueur
est un ennemi

Paramètres :

g : Game

Valeur de retour: True ou False

Game.**show(g)**->str

Description: renvoie le jeu

Paramètres :

g : Game

Valeur de retour: affichage de la zone de jeu

4.3 World.py

- `World.create()`
- `World.getRooms(w)`
- `World.getCorridors(w)`
- `World.getSpawnPoint(w)`
- `World.setRooms(w)`
- `World.setCorridors(w)`
- `World.setSpawnPoint(w)`
- `World.getTileData(w, coordinates)`
`World.int(w)`
- `World.show(w)`
- ~~`World.isTileBossEntry(w, g)`~~
- ~~`World.discoverArea(w, g)`~~

`World.create()` -> `World`

Description: crée un nouveau monde, avec des salles aléatoires

Paramètres: rien

Valeur de retour: `world`

`World.getTileData(w, coordinates)` -> `str`

Description: donne les données de la case sélectionnée

Paramètres:

`w` : `World`

`coordinates` : `tuple(int,int)`

Valeur de retour: rien

`World.connectHorizontalDoors(w, d1, d2)` -> rien

Description: relie deux portes parallèles horizontales

Paramètres:

`w` : `World`

`d1` : `tuple (int, int)`

`d2` : `tuple (int, int)`

Valeur de retour: rien

`World.connectVerticalDoors(w, d1, d2)` -> rien

Description: relie deux portes parallèles verticales

Paramètres:

`w` : `World`

`d1` : `tuple (int, int)`

`d2` : `tuple (int, int)`

Valeur de retour: rien

`World.connectVerticalToHorizontalDoors(w, d1, d2)` -> rien

Description: relie deux portes perpendiculaires

Paramètres:

`w` : `World`

`d1` : `tuple (int, int)`

`d2` : `tuple (int, int)`

Valeur de retour: rien

World.**init**(w) -> rien
Description: charge le visuel du monde
Paramètres:
 w: World
Valeur de retour: rien

World.**show**(w) -> rien
Description: affiche le monde
Paramètres:
 w: World
Valeur de retour: rien

~~World.**isTileBossEntry**(w, g) -> bool~~
~~—— Description: regarde si la case du joueur est une entrée~~
~~—— de Boss~~
~~—— Paramètres :~~
~~—— w: World~~
~~—— g: Game~~
~~—— Valeur de retour: True ou False~~ Abandonné

~~World.**discoverArea**(w, g) -> rien~~
~~—— Description: active l'affichage des salles où le joueur~~
~~—— est passé~~
~~—— Paramètres :~~
~~—— w: World~~
~~—— g: Game~~
~~—— Valeur de retour: rien~~ Abandonné

4.4 Boss.py

- ~~Boss.create(level)~~
- ~~Boss.getStatistics(b)~~
- ~~Boss.getPlayerAttacks(b)~~
- ~~Boss.setStatistics(b)~~
- ~~Boss.setPlayerAttacks(b)~~
- ~~Boss.generatePlayerAttacks(b, g)~~
- ~~Boss.letPlayerAttack(b, g)~~
- ~~Boss.letBossAttack(b, g)~~
- ~~Boss.isDead(b)~~

~~Boss.create(level) -> Boss~~
 — Description: génère un boss en fonction du niveau de la partie
 — Paramètres:
 — level: int
 — Valeur de retour: un boss de fin de niveau

~~Boss.generatePlayerAttacks(b, g) -> tuple~~
 — Description: génère un tuple contenant quatre attaques
 — différentes en fonction des armes du joueur
 — Paramètres:
 — b: Boss
 — g: Game
 — Valeur de retour: quatre attaques pour le combat de boss

~~Boss.letPlayerAttack(b) -> rien~~
 — Description: le joueur attaque le Boss
 — Paramètres:
 — b: Boss
 — Valeur de retour: rien

~~Boss.letBossAttack(b) -> rien~~
 — Description: le Boss attaque le joueur
 — Paramètres:
 — b: Boss
 — Valeur de retour: rien

Abandonné

4.5 Room.py

- `Room.create(level)`
- `Room.getCoordinates(r)`
- `Room.getGates(r)`
- `Room.getSize(r)`
- `Room.getEnemies(r)`
- `Room.getObjects(r)`
- `Room.setCoordinates(r)`
- `Room.setGates(r)`
- `Room.setSize(r)`
- `Room.setEnemies(r)`
- `Room.setObjects(r)`
- `Room.show(r)`

`Room.create(level)` -> `Room`

Description: génère une salle en fonction du niveau de la partie, avec des ennemis, des objets et des accès

Paramètres:

level: int

Valeur de retour: une pièce du donjon

`Room.createDoors(r, gates)` -> rien

Description: génère les portes de la salle en fonction de sa position et de sa taille

Paramètres:

r : Room

gates: list

Valeur de retour: rien

`Room.createEnemies(r)` -> rien

Description: génère les ennemis de la salle

Paramètres:

r : Room

Valeur de retour: rien

`Room.createObject(r)` -> rien

Description: génère l'objet de la salle

Paramètres:

r : Room

Valeur de retour: rien

`Room.show(r)` -> rien

Description: affiche la salle

Paramètres:

r : Room

Valeur de retour: rien

4.6 Enemy.py

- `Enemy.create(level)`
- `Enemy.getType(e)`
- `Enemy.getCoordinates(e)`
- `Enemy.getLife(e)`
- `Enemy.getStrenght(e)`
- `Enemy.getSpeed(e)`
- `Enemy.setType(e)`
- `Enemy.setCoordinates(e)`
- `Enemy.setLife(e)`
- `Enemy.getStrenght(e)`
- `Enemy.getSpeed(e)`
- `Enemy.move(e, r)`
- `Enemy.show(e)`

`Enemy.create(level)` -> `Enemy`

Description: génère un ennemi en fonction du niveau de la partie

Paramètres:

level: int

Valeur de retour: un ennemi

`Enemy.move(e, r)` -> `Enemy`

Description: déplace l'ennemi dans la salle

Paramètres:

e: `Enemy`

r: `Room`

Valeur de retour: rien

`Enemy.move(e)` -> rien

Description: affiche l'ennemi

Paramètres:

e: `Enemy`

Valeur de retour: rien

4.7 Object.py

- `Object.create(level)`
- `Object.getType(o)`
- `Object.getCoordinates(o)`
- `Object.getStatistics(o)`
- `Object.setType(o)`
- `Object.setCoordinates(o)`
- `Object.setStatistics(o)`

`Object.create(level)` -> `Object`

Description: génère un objet (arme/armure) en fonction du niveau de la partie

Paramètres:

level: int

Valeur de retour: une pièce du donjon

`Object.show(o)` -> rien

Description: affiche l'objet

Paramètres:

o: `Object`

Valeur de retour: rien

5 Calendrier et suivi de développement

5.1 P1 :

5.1.1 fonctions à développer

fonctions	codées	testées
<code>Main.main()</code>	Oui	Oui
<code>Main.init()</code>	Oui	Oui
<code>Main.run()</code>	Oui	Oui
<code>Main.show()</code>	Oui	Oui
<code>Main.interact()</code>	Oui	Oui
<code>Main.live()</code>	Oui	Oui
<code>Main.finish()</code>		
<code>Game.create(statistics, level)</code>	Oui	Oui
<code>Game.get... (g)</code>	Oui	Oui
<code>Game.set... (g)</code>	Oui	Oui
<code>Game.isPlayerDead (g)</code>	Oui	Oui
<code>Game.addObject (g)</code>	Oui	Oui
<code>Game.rememberEnemy (g)</code>	Oui	Oui
<code>Game.rememberObject (g)</code>	Oui	Oui
<code>Game.movePlayer (g,k)</code>	Oui	Oui
<code>Game.moveEnemy (g,e)</code>	Oui	Oui
<code>Game.giveObjectEffect (g)</code>	Oui	Oui
<code>Game.confrontEnemy (g)</code>	Oui	Oui
<code>Game.isNextTileReachable (...)</code>	Oui	Oui
<code>Game.isTileObject (g)</code>	Oui	Oui
<code>Game.isTileEnemy (g)</code>	Oui	Oui
<code>Game.show (g)</code>	Oui	Oui

World.create()	Oui	Oui
World.get...()	Oui	Oui
World.set...()	Oui	Oui
World.connectHorizontalDoors(w, d1, d2)	Oui	Oui
World.connectVerticalDoors(w, d1, d2)	Oui	Oui
World.connectVerticalToHorizontalDoors(w,d1,d2)	Oui	Oui
World.getTileData(w,coordinates)	Oui	Oui
World.init(w)	Oui	Oui
World.discoverArea(w,g)	Abandonné	
World.isTileBossEntry(w,g)		

Boss.create(level)	Abandonné	
Boss.get...()		
Boss.set...()		
Boss.generatePlayerAttacks(b, g)		
Boss.letPlayerAttack(b, g)		
Boss.letBossAttack(b, g)		
Boss.isDead(b)		
Room.create(level)	Oui	Oui
Room.get...()	Oui	Oui
Room.set...()	Oui	Oui
Room.createDoors(r, gates)	Oui	Oui
Room.createEnemies(r)	Oui	Oui
Room.createObject(r)	Oui	Oui
Room.show(r)	Oui	Oui
Enemy.create(level)	Oui	Oui
Enemy.get...()	Oui	Oui
Enemy.set...()	Oui	Oui
Enemy.show(e)	Oui	Oui
Object.create(level)	Oui	Oui
Object.get...()	Oui	Oui
Object.set...()	Oui	Oui
Object.show(o)	Oui	Oui

5.1.2 Autres

Fichier JSON RoomTypes.json permettant de lister toutes les formes de salles possibles.

De la forme :

```
{
  "Center" : {
    "gates": [1, 1, 1, 1],
    "maximum-size": [25, 8],
    "minimum-size": [15, 6]
  },
  ... ..
}
```

Fichier JSON EnnemiTypes.json permettant de lister tous les ennemis possibles.

De la forme :

```
{
  "Rat": {
    "character": "r",
    "life": 5,
    "strenght": 1
  },
  ... ..
}
```

Fichier JSON ObjectTypes.json permettant de lister tous les ennemis possibles.

De la forme :

```
{
  "Potion de vie": {
    "character": "+",
    "stat": "life",
    "statistiques": "Vie +20%",
    "value": 20
  },
  ... ..
}
```

Fichier JSON dataSaved.json permettant de lister tous les ennemis possibles.

De la forme : {"Best": 0, "Bestiary": [], "Inventory": [],...}