
Centre de formation professionnelle technique
École d'informatique

Travail de diplôme
Documentation

2014-2015

---

# Hel

***The pixelated horror***

---

*Auteur :*
Yannick Brodard
yannick.r.brodard@gmail.com
+41 79 137 67 11

*Superviseur :*
Christophe Maréchal
christophe.marechal@edu.ge.ch

1er juin 2015

# Chapitre 1

# Avant-propos

## 1.1 Résumé

Le but de ce travail est de réaliser un jeu vidéo. Ce jeu doit inclure les aspects de bases pour la jouabilité. Il ne sera bien évidemment pas terminé, parce qu'il nécessite beaucoup de temps en plus pour équilibré et ajouté du contenu graphique. Le jeu est un hack n' slash classique avec une vue de dessus (personnage vu de haut). Le héros doit gagner des points de compétences pour devenir plus fort et parcourir trois mondes générer aléatoirement pour trouver son ennemi ultime. Avec l'aide de sorts, il essayera de vaincre ses ennemis. L'utilisation du Framework de jeu "Monogame" est indispensable pour ce travail, ce Framework fournit les fonctionnalités de base pour un jeu vidéo. La carte est générée en utilisant un algorithme cellulaire modifié pour la génération de cavernes naturelles. Les ennemis se déplacent grâce à des algorithmes de déplacement vectoriels avec les attaques et sorts inclus. Le projet s'est terminé avec du retard mais tout de même une bonne base sur laquelle il peut être continué facilement.

## 1.2 Abstract

The objective of this work is to achieve a video game. This game has to include basic aspects of gameplay. It will surely not be completely finished, because a game of this size needs more time to balance and add graphical content. The game is a classic hack n' slash with a point of view from the top of the character. The hero has to win skill points to become stronger et has to travel in three randomly generated worlds to find his ultimate enemy. With the aid of spells and skills, he will try to defeat his enemies. The use of the "Monogame" framework is essential for this work, this framework provides the basic functionalities and architecture for a game. The map is generated by using a cellular algorithm that has been tweaked to correspond to natural cavern generation. The enemies move thanks to vectorial movement algorithm which has also been tweaked to fit the need and add the attacks and spells. The project finished with some delay, but has a good base from which it can be easily continued.

# Table des matières

# Chapitre 2

# Cahier des charges

## 2.1 But

L'objectif est de créer un jeu divertissant pour le joueur moyen. Hel [1] doit être bien construit et doit avoir la possibilité d'ajouter du contenu facilement.

Le jeu-vidéo est en deux dimensions (2D) avec une vue de dessus [2] et inclue quelques aspects d'un jeu RPG [3]. L'utilisateur contrôle un personnage qui se retrouve coincé dans une pièce inconnue et doit s'aventurer dans cet environnement inconnu et découvrir ses mystères. Cette salle contient des ennemis que le joueur peut contourner ou vaincre. Grâce à ceci, le héros gagne des points de compétences qu'il peut dépenser pour augmenter ses capacités une fois que le joueur meurt. Quand le joueur meurt, celui-ci réapparaît au début de cette salle, mais sans perdre ses compétences, d'une part la pièce aura aussi changé. Des éléments de celui-ci auront disparu ou bougé et d'autres auront apparu. L'utilisateur devra s'aventurer dans cet endroit mystérieux pour essayer de trouver une sortie. Une fois que le joueur a trouvé la sortie, il affrontera *Hel*, la déesse de la mort seulement après l'avoir vaincu qu'il pourra sortir de cette salle, mais si le héros meurt pendant l'affrontement, il devra recommencer depuis le début et perdra tous ses points de compétences.

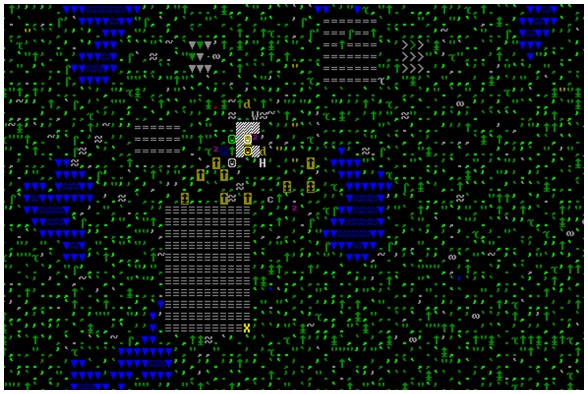


FIGURE 2.1 – Dwarf Fortress - Jeu en pixel art fait par *Bay 12 Games*

Hel est principalement inspiré du jeu *Don't Move* par "stvr" (voir Figure 2.2 page 6). *Don't Move* est un jeu indépendant qui a reçu de très bonnes critiques. Le concept du jeu est simple, si le joueur bouge, il meurt, mais en faisant cela il débloque de nouvelles fonctionnalités du jeu et de nouveaux objectifs. Le but de Hel est de pousser ce concept encore plus loin et de l'améliorer avec plus de fonctionnalités.

1. *Hel* : Nom du jeu-vidéo et aussi nom de la déesse de la mort dans la mythologie nordique
2. Une vue de dessus peut être comparable au jeu Dwarf Fortress. Voir voir Figure 2.1 page 5
3. Rôle Playing Game : Un RPG est un jeu où le joueur fait évoluer son personnage avec ses choix et/ou ses compétences.


FIGURE 2.2 – Don't Move - jeu indépendant développé par "stvr"

## 2.2 Description détaillée

Hel est un jeu solo (1 joueur seulement) qui est en deux dimensions. Le jeu sera développé de manière dynamique et générique qui permettra de facilité l'ajout de contenu. Ce contenu peut être :

- des armes
- des ennemis
- des nouvelles compétences
- de nouveaux types de points de compétences
- des objets de la salle
- des différentes disposition de la salle
- de nouvelles règles de jeu

### 2.2.1 Règles

Le joueur peut : sauter, se déplacer sur l'axe X, se baisser (ceci active la furtivité), attaquer avec une arme, attaquer avec de la magie et bloquer des attaques pour minimiser les dégâts.

Le joueur a 100 points de vie par défaut.

Les dégâts et vie des ennemis seront à définir selon le meilleur équilibre de jeu.

Le joueur possède des sorts qu'il peut placer dans les touches attribuées pour les enclencher.

### 2.2.2 Mécanismes

#### Salle

La salle du jeu change à chaque fois que le joueur meurt. Cette salle peut avoir de nouveaux éléments, de nouvelles textures ou même de nouveaux ennemis.

#### Points de compétences

Le joueur peut gagner des points de compétences en tuant des ennemis ou en trouvant des pièces qui se trouve un peu partout dans la salle. Ces pièces sont défendu par des ennemis et peuvent disparaître si le joueur n'est pas discret. Il ne peut dépenser ces points après sa mort.

#### Compétences

Avant de réapparaître en jeu, le joueur peut attribuer ces points sur des compétences du personnage. Par exemple, le joueur peut dépenser 2 points de compétences sur la compétence "Force" pour avoir plus de dégâts sur les coups du personnage. Voici une liste des compétences :

| Compétence | Description |
|---|---|
| Force | Apporte plus de dégâts sur les coups physiques du personnage. |
| Agilité | Les ennemis remarquent moins le joueur et celui-ci peut se déplacer plus rapidement |
| Vitalité | Apporte plus de points de vie au joueur |
| Magie | Apporte plus de dégâts pour les attaques magiques du joueur |

#### Armes

Les armes sont des objets pouvant être récoltés en explorant la salle. Quand le joueur trouve une arme, cette arme est mise dans l'inventaire du joueur. Le joueur pourra équiper l'arme pour faire plus de dégâts. Les armes ont des statistiques qui augmentent les dégâts physiques et/ou magiques.

#### Sorts

Les sorts sont des pouvoirs magiques que le joueur peut utiliser contre ses ennemis. Les dégâts magiques sont augmentés selon les points de compétences attribués à la magie. Des sorts peuvent être trouvés dans la salle sous forme de parchemin que le joueur peut ramasser, les sorts se placent dans l'inventaire et peuvent être attribués à des touches d'action (Maximum 4, touches par défaut : 1, 2, 3 ,4).

### 2.2.3 Contrôles

| Touche | Description |
|---|---|
| W | Sauter |
| A | Aller à gauche |
| D | Aller à droite |
| S | Se baisser (mode furtif) |
| Clique gauche | Attaquer (dégâts physique) |
| 1 | Sort numéro 1 |
| 2 | Sort numéro 2 |
| 3 | Sort numéro 3 |
| 4 | Sort numéro 4 |
| I | Afficher l'inventaire |
| ESC | Afficher le menu (met le jeu en pause) |

Table 2.1 – Contrôles de bases du jeu

### 2.2.4 Interface utilisateur

L'interface utilisateur doit permettre au joueur de régler plusieurs paramètres et exécuter des actions.

#### Options

Les options de son, vidéo, difficulté et attribution des touches de jeu doivent être disponibles.

#### Actions

Il doit être possible de commencer, sauvegarder et charger une partie, puis de visualiser les scores obtenus des parties terminées.

## 2.3 Environnement de travail

### 2.3.1 Inventaire hardware

— Intel Core i7 2600K @ 3.40GHz
— 8.00Go Dual-Channel DDR3 @ 823MHz
— 1024MB NVIDIA GeForce GTX 550 Ti

### 2.3.2 Inventaire software

— Visual Studio 2013 Professional

— MonoGame (C#)

### 2.3.3 Durée et dates du projet

Le projet se déroule sur une période de 7 semaines avec 40 heures de travail par semaine. Il y a aussi 3 jours fériés pendant le projet. Ce qui fait un total de 256 heures de travail.

Le projet commence le lundi, 13 avril 2015 avec une reddition intermédiaire (documentation + poster) le jeudi, 30 avril 2015, puis la reddition finale le lundi, 1er juin 2015.

# Chapitre 3

# Analyse préliminaire

## 3.1 Analyse de l'existant

Il n'est pas anodin de trouver des jeux similaires dans le marché du jeu-vidéo, en effet, beaucoup d'indépendants et des grandes entreprises produisent ces divertissements. Parmi les plus connus qui peuvent ressembler à Hel, il existe *Diablo III* et *Path of Exile* qui sont des jeux récents. Ils sont similaires par rapport au type de jeu, ce sont des RPG et des Hack and Slash.
*Bastion* et *Transistor* sont également des jeux RPG et indépendant créé par la petite entreprise *Supergiant Games*. Ces jeux sont prisés pour leur art, richesse graphique et histoire passionnante.

## 3.2 Critique de l'existant

**Diablo III** par *Blizzard Entertainment* est actuellement le poids-lourd dans ce type de jeu, avec de hautes qualités graphiques, un contenu riche et une grande communauté active. Il est critiqué de ne pas être assez complexe et d'être trop facile à battre.
**Path of Exile** par *Grinding Gear Games* est le concurrent direct de *Diablo III*, il est valorisé avec sa complexité technique dans le gameplay. Il est critiqué de ne pas être assez accessible aux joueurs lambda et d'être trop difficile.
**Bastion** par *Supergiant Games* est un jeu indépendant ayant gagné plusieurs titres pour son originalité.

# Chapitre 4

# Analyse fonctionnelle

## 4.1 Personnage

### 4.1.1 Général

Un personnage est contrôlé par le joueur, celui-ci peut l'équiper avec divers objets trouvés dans la salle pour augmenter ses caractéristiques. Il peut utiliser des sorts et ses armes pour vaincre ses ennemis. Le personnage possède **100** points de vie par défaut, ces points de vie augmentent si le joueur décide d'améliorer sa vitalité avec des points de compétences ou des objets.
Le personnage possède 4 compétences de bases (voir section 4.2), il peut gagner des compétences en tuant des ennemis ou en trouvant des pièces dans la salle.
Il possède **200** points de mana qu'il peut dépenser pour utiliser des sorts. Le mana se régénéré avec le temps de **20** points de mana par seconde.

### 4.1.2 Inventaire

Le personnage possède un inventaire qui sert à stocker ses objets sur soi. Les objets stockés dans l'inventaire prennent une certaine place à l'intérieur. Il n'est pas de taille infinie.

## 4.2 Compétences

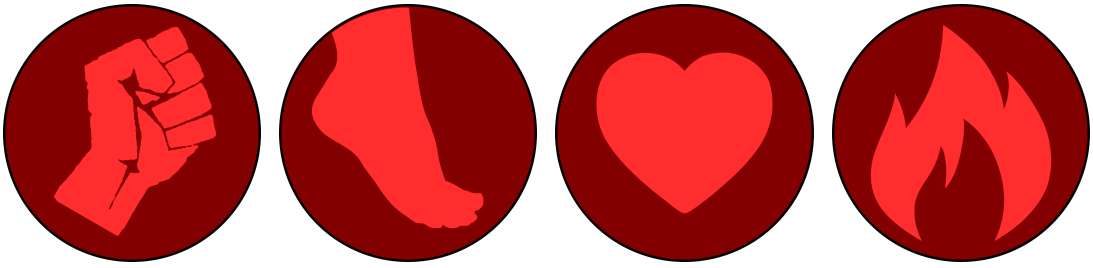
Les compétences sont des caractéristiques pouvant être augmenté par le personnage. Ils permettent d'augmenter ses capacités. Il existe plusieurs compétences qui sont :

**Force** Apporte plus de dégâts sur les coups physiques

**Agilité** Le personnage est plus discrète et peut se déplacer plus rapidement

**Vitalité** Augmente les points de vie du personnage

**Magie** Augmente les dégâts magiques du personnage



(a) Force (b) Agilité (c) Vitalité (d) Magie

FIGURE 4.1 – Logos des compétences

## 4.3 Looting

Le *looting* est le moyen de trouver des objets. Lorsqu'un ennemi meurt ou que l'on ouvre un coffre, par exemple, celui-ci fait tomber des objets. Cette action est appelée le *looting*.

## 4.4 Caractéristiques

Les caractéristiques sont des propriétés qui augmentent les capacités du personnage. Les compétences sont des caractéristiques mais qui peuvent être augmentées par le héros, alors que les caractéristiques peuvent seulement être augmentées par des objets.

**Liste des caractéristiques**

**Force** Le nombre de points de compétences de type *Force* ajouté

**Agilité** Le nombre de points de compétences de type *Agilité* ajouté

**Vitalité** Le nombre de points de compétences de type *Vitalité* ajouté

**Magie** Le nombre de points de compétences de type *Magie* ajouté

**Vitesse d'attaque** Le pourcentage de vitesse d'attaque ajouté au personnage

**Vitesse d'attaque initial** La vitesse d'attaque initial du personnage

**Dégâts** Les dégâts infligés après chaque attaque

**Dégâts magiques** Les dégâts magiques infligés après chaque attaque

**Armure** Déduit les dégâts reçu

**Résistance magique** Déduit les dégâts magiques reçu

**Régénération de vie** Régénère les points de vie

**Régénération de mana** Régénère les points de mana

| | |
|---|---|
| Force | Augmente les dégâts de **1%** |
| Agilité | Diminue la distance de vue des ennemi de **0.0625%**. Il augmente aussi la vitesse de déplacement de **0.075%** |
| Vitalité | Augmente les points de vie de **100** |
| Magie | Augmente les dégâts magiques de **1%** |
| Vitesse d'attaque | Augmente la vitesse d'attaque selon la valeur en **pourcentage** |
| Vitesse d'attaque initial | Vitesse d'attaque initial du personnage |
| Dégâts | S'ajoute au dégâts du personnage avant d'appliquer la *force* |
| Dégâts magiques | S'ajoute au dégâts magiques du personnage avant d'appliquer la *magie* |
| Armure | Diminue les dégâts reçu de **0.075%** |
| Resistance magique | Diminue les dégâts magiques reçu de **0.075%** |
| Régénération de vie | Augmente la régénération de points de vie selon la valeur |
| Régénération de mana | Augmente la régénération de points de mana selon la valeur |
| Vitesse de déplacement initial | Fixe la vitesse de déplacement de base du personnage |
| Vitesse de déplacement | Augmente la vitesse de déplacement |

Table 4.1 – Fonctionnement des caractéristiques

## 4.5 Équipement

Les équipements sont des objets que le personnage peut équiper sur lui-même pour augmenter ses points de compétences. Il existe des différents types d'objets.

— Accessoires
— Armes
— Armures

### 4.5.1 Accessoires

Les accessoires sont des objets spéciaux qui apportent des capacités spéciales au personnage.

— Amulette
— Bagues
— Carquois
— Bouclier
— Source de pouvoir

#### Amulette et bagues

Ces objets apportent aléatoirement 2 à 4 des caractéristiques suivants :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Force | 3 - 10 | 15 - 30 | 45 - 100 |
| Agilité | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitalité | 3 - 10 | 15 - 30 | 45 - 100 |
| Magie | 3 - 10 | 15 - 30 | 45 - 100 |
| Résistance magique | 2 - 5 | 10 - 20 | 30 - 55 |
| Régénération de vie | 10 - 80 | 100 - 450 | 500 - 1000 |
| Régénération de mana | 1 - 5 | 6 - 13 | 14 - 20 |

TABLE 4.2 – Valeurs des caractéristiques d'une bague et d'une amulette

#### Carquois

Le carquois apporte toujours ces caractéristiques :

— Vitesse d'attaque
— Agilité

Il apporte aussi 0 à 3 de ces caractéristiques :

— Force
— Vitalité
— Magie
— Régénération de vie
— Régénération de mana

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Force | 1 - 5 | 7 - 25 | 35 - 80 |
| Agilité | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitalité | 1 - 5 | 7 - 25 | 35 - 80 |
| Magie | 1 - 5 | 7 - 25 | 35 - 80 |
| Vitesse d'attaque | 1.5% - 2.0% | 2.5% - 3.5% | 4.0% - 7.0% |
| Régénération de vie | 10 - 80 | 100 - 450 | 500 - 1000 |
| Régénération de mana | 1 - 5 | 6 - 13 | 14 - 20 |

TABLE 4.3 – Valeurs des caractéristiques du carquois

#### Bouclier

Le bouclier apporte toujours ces caractéristiques :

— Armure
— Vitalité

Il apporte aussi 0 à 3 de ces caractéristiques :

— Force
— Agilité
— Magie
— Résistance magique
— Régénération de vie
— Régénération de mana

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Force | 1 - 5 | 7 - 25 | 35 - 80 |
| Agilité | 1 - 5 | 7 - 25 | 35 - 80 |
| Vitalité | 3 - 10 | 15 - 30 | 45 - 100 |
| Magie | 1 - 5 | 7 - 25 | 35 - 80 |
| Armure | 3 - 10 | 15 - 30 | 45 - 100 |
| Resistance magique | 3 - 10 | 15 - 30 | 45 - 100 |
| Régénération de vie | 10 - 80 | 100 - 450 | 500 - 1000 |
| Régénération de mana | 1 - 5 | 6 - 13 | 14 - 20 |

Table 4.4 – Valeurs des caractéristiques du bouclier

**Source de pouvoir**

La source de pouvoir apporte toujours ces caractéristiques :

— Magie
— Dégâts magiques

Elle apporte aussi 0 à 2 de ces caractéristiques :

— Agilité
— Vitalité
— Vitesse d'attaque
— Régénération de vie
— Régénération de mana

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Agilité | 1 - 5 | 7 - 25 | 35 - 80 |
| Vitalité | 1 - 5 | 7 - 25 | 35 - 80 |
| Magie | 5 - 15 | 20 - 35 | 50 - 120 |
| Vitesse d'attaque | 1.5% | 2.0% - 3.0% | 3.5% - 5.0% |
| Dégâts magiques | 10 - 20 | 30 - 60 | 70 - 120 |
| Régénération de vie | 10 - 80 | 100 - 450 | 500 - 1000 |
| Régénération de mana | 1 - 5 | 6 - 13 | 14 - 20 |

Table 4.5 – Valeurs des caractéristiques de la source de pouvoir

### 4.5.2 Armes

— Épée
— Épée à deux mains
— Hache
— Hache à deux mains
— Baguette magique
— Bâton magique
— Arc

Les armes sont séparées en plusieurs fonctions. En bleu les armes à une main, en rouge les armes à deux mains et en vert les accessoires complémentaires aux armes (voir section 4.5.1). Le personnage peut équiper plusieurs armes en même temps selon les restrictions (voir figure 4.2).

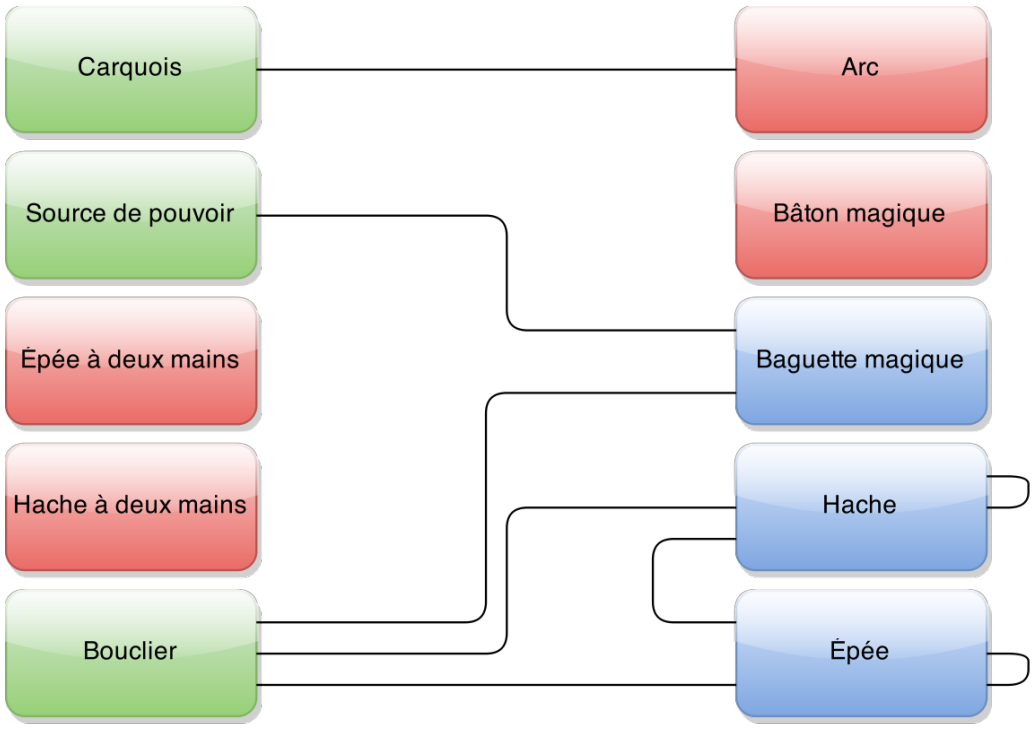



FIGURE 4.2 – Restriction des armes

### Critères d'une arme

Une arme est dotée de caractéristiques obligatoires qui apparaîtront sur toutes les armes lootées dans le jeu :

— Vitesse d'attaque initial

Elle est aussi dotée de caractéristiques optionnelles qui peuvent y figurer, certaines de ses caractéristiques apparaissent obligatoirement selon le type d'objet :

— Dégâts
— Dégâts magique
— Force
— Agilité
— Vitalité
— Magie

### Spécialités des type d'armes

**Épée** Arme à une main avec la plus grande vitesse d'attaque

**Épée à deux mains** Arme à deux mains avec la plus grande vitesse d'attaque

**Hache** Arme à une main avec le plus de dégâts

**Hache à deux mains** Arme à deux mains avec le plus de dégâts

**Baguette magique** Permet d'avoir une source de pouvoir équipé en même temps.

**Bâton magique** Possède de haut dégâts magiques

**Arc** Haut dégâts physique à distance

**Épée**

L'épée apporte toujours ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Vitesse d'attaque initial | 1.40 | 1.40 | 1.40 |
| Dégâts | 30 - 50 | 60 - 90 | 110 - 160 |

TABLE 4.6 – Valeurs des caractéristiques obligatoires de l'épée

Elle apporte aussi 2 à 4 de ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Force | 3 - 10 | 15 - 30 | 45 - 100 |
| Agilité | 1 - 5 | 7 - 25 | 35 - 80 |
| Vitalité | 3 - 10 | 15 - 30 | 45 - 100 |
| Magie | 1 - 5 | 7 - 25 | 35 - 80 |
| Dégâts magiques | 5 - 10 | 20 - 40 | 50 - 80 |

TABLE 4.7 – Valeurs des caractéristiques optionnelles de l'épée

**Épée à deux mains**

L'épée à deux mains apporte toujours ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Force | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitesse d'attaque initial | 1.10 | 1.10 | 1.10 |
| Dégâts | 65 - 110 | 120 - 190 | 230 - 360 |

TABLE 4.8 – Valeurs des caractéristiques obligatoires de l'épée à deux mains

Elle apporte aussi 1 à 3 de ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Vitalité | 3 - 10 | 15 - 30 | 45 - 100 |
| Magie | 1 - 5 | 7 - 25 | 35 - 80 |
| Dégâts magiques | 3 - 15 | 25 - 45 | 60 - 90 |
| Régénération de vie | 5 - 50 | 80 - 275 | 375 - 750 |
| Régénération de mana | 1 - 3 | 4 - 10 | 11 - 15 |

TABLE 4.9 – Valeurs des caractéristiques optionnelles de l'épée à deux mains

**Hache**

La hache apporte toujours ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Force | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitesse d'attaque initial | 1.30 | 1.30 | 1.30 |
| Dégâts | 30 - 50 | 60 - 90 | 110 - 160 |

TABLE 4.10 – Valeurs des caractéristiques obligatoires de la hache

Elle apporte aussi 1 à 3 de ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Magie | 1 - 5 | 7 - 25 | 35 - 80 |
| Vitalité | 3 - 10 | 15 - 30 | 45 - 100 |
| Dégâts magiques | 3 - 10 | 20 - 40 | 50 - 80 |

TABLE 4.11 – Valeurs des caractéristiques optionnelles de la hache

**Hache à deux mains**

La hache à deux mains apporte toujours ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Force | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitalité | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitesse d'attaque initial | 1.00 | 1.00 | 1.00 |
| Dégâts | 65 - 110 | 120 - 190 | 230 - 360 |

TABLE 4.12 – Valeurs des caractéristiques obligatoires de la hache à deux mains

Elle apporte aussi 0 à 2 de ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Magie | 1 - 5 | 7 - 25 | 35 - 80 |
| Dégâts magiques | 3 - 15 | 25 - 45 | 60 - 90 |
| Régénération de vie | 5 - 50 | 80 - 275 | 375 - 750 |
| Régénération de mana | 1 - 3 | 4 - 10 | 11 - 15 |

TABLE 4.13 – Valeurs des caractéristiques optionnelles de la hache à deux mains

**Baguette magique**

La baguette magique apporte toujours ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Magie | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitesse d'attaque initial | 1.40 | 1.40 | 1.40 |
| Dégâts magiques | 30 - 50 | 60 - 90 | 110 - 160 |

TABLE 4.14 – Valeurs des caractéristiques obligatoires de la baguette magique

Elle apporte aussi 1 à 2 de ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Agilité | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitalité | 3 - 10 | 15 - 30 | 45 - 100 |

TABLE 4.15 – Valeurs des caractéristiques optionnelles de la baguette magique

**Bâton magique**

Le bâton magique apporte toujours ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Magie | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitesse d'attaque initial | 1.20 | 1.20 | 1.20 |
| Dégâts magiques | 65 - 110 | 120 - 190 | 230 - 360 |

TABLE 4.16 – Valeurs des caractéristiques obligatoires du bâton magique

Il apporte aussi 1 à 2 de ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Agilité | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitalité | 3 - 10 | 15 - 30 | 45 - 100 |
| Régénération de vie | 5 - 50 | 80 - 275 | 375 - 750 |
| Régénération de mana | 1 - 3 | 4 - 10 | 11 - 15 |

TABLE 4.17 – Valeurs des caractéristiques optionnelles du bâton magique

**Arc**

L'arc apporte toujours ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Agilité | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitesse d'attaque initial | 1.30 | 1.30 | 1.30 |
| Dégâts | 30 - 50 | 60 - 90 | 110 - 160 |

TABLE 4.18 – Valeurs des caractéristiques obligatoires de l'arc

Il apporte aussi 1 à 3 de ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Force | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitalité | 3 - 10 | 15 - 30 | 45 - 100 |
| Magie | 1 - 5 | 7 - 25 | 35 - 80 |
| Dégâts magiques | 3 - 10 | 20 - 40 | 50 - 80 |

TABLE 4.19 – Valeurs des caractéristiques optionnelles de l'arc

### 4.5.3 Armures

- Tête
- Épaules
- Corps
- Mains
- Jambes
- Pieds

**Tête**

Une armure pour la tête apporte obligatoirement ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Vitalité | 3 - 10 | 15 - 30 | 45 - 100 |
| Armure | 3 - 10 | 15 - 30 | 45 - 100 |

TABLE 4.20 – Valeurs des caractéristiques obligatoire de l'armure pour la tête

Elle apporte aussi 0 à 2 de ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Force | 3 - 10 | 15 - 30 | 45 - 100 |
| Agilité | 3 - 10 | 15 - 30 | 45 - 100 |
| Magie | 3 - 10 | 15 - 30 | 45 - 100 |
| Resistance magique | 3 - 10 | 15 - 30 | 45 - 100 |
| Régénération de vie | 5 - 50 | 80 - 275 | 375 - 750 |
| Régénération de mana | 1 - 3 | 4 - 10 | 11 - 15 |

TABLE 4.21 – Valeurs des caractéristiques optionnelles de l'armure pour la tête

**Épaules**

Une armure pour les épaules apporte obligatoirement ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Armure | 3 - 10 | 15 - 30 | 45 - 100 |

TABLE 4.22 – Valeurs des caractéristiques obligatoire de l'armure pour les épaules

Elle apporte aussi 1 à 3 de ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Force | 3 - 10 | 15 - 30 | 45 - 100 |
| Agilité | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitalité | 3 - 10 | 15 - 30 | 45 - 100 |
| Magie | 3 - 10 | 15 - 30 | 45 - 100 |
| Resistance magique | 3 - 10 | 15 - 30 | 45 - 100 |

TABLE 4.23 – Valeurs des caractéristiques optionnelles de l'armure pour les épaules

**Corps**

Une armure pour le corps apporte obligatoirement ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Armure | 3 - 10 | 15 - 30 | 45 - 100 |

TABLE 4.24 – Valeurs des caractéristiques obligatoire de l'armure pour le corps

Elle apporte aussi 1 à 3 de ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Force | 3 - 10 | 15 - 30 | 45 - 100 |
| Agilité | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitalité | 3 - 10 | 15 - 30 | 45 - 100 |
| Magie | 3 - 10 | 15 - 30 | 45 - 100 |
| Resistance magique | 3 - 10 | 15 - 30 | 45 - 100 |
| Régénération de vie | 5 - 50 | 80 - 275 | 375 - 750 |
| Régénération de mana | 1 - 3 | 4 - 10 | 11 - 15 |

TABLE 4.25 – Valeurs des caractéristiques optionnelles de l'armure pour le corps

**Mains**

Une armure pour les mains apporte obligatoirement ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Agilité | 3 - 10 | 15 - 30 | 45 - 100 |
| Armure | 3 - 10 | 15 - 30 | 45 - 100 |

TABLE 4.26 – Valeurs des caractéristiques obligatoire de l'armure pour les mains

Elle apporte aussi 0 à 2 de ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Force | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitalité | 3 - 10 | 15 - 30 | 45 - 100 |
| Magie | 3 - 10 | 15 - 30 | 45 - 100 |
| Resistance magique | 3 - 10 | 15 - 30 | 45 - 100 |

TABLE 4.27 – Valeurs des caractéristiques optionnelles de l'armure pour les mains

**Jambes**

Une armure pour les jambes apporte obligatoirement ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Armure | 3 - 10 | 15 - 30 | 45 - 100 |

TABLE 4.28 – Valeurs des caractéristiques obligatoire de l'armure pour les jambes

Elle apporte aussi 1 à 3 de ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Force | 3 - 10 | 15 - 30 | 45 - 100 |
| Agilité | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitalité | 3 - 10 | 15 - 30 | 45 - 100 |
| Magie | 3 - 10 | 15 - 30 | 45 - 100 |
| Resistance magique | 3 - 10 | 15 - 30 | 45 - 100 |

TABLE 4.29 – Valeurs des caractéristiques optionnelles de l'armure pour les jambes

**Pieds**

Une armure pour les pieds apporte obligatoirement ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Agilité | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitesse de déplacement initial | 6% - 12% | 6% - 12% | 6% - 12% |
| Armure | 3 - 10 | 15 - 30 | 45 - 100 |

TABLE 4.30 – Valeurs des caractéristiques obligatoire de l'armure pour les pieds

Elle apporte aussi 0 à 2 de ces caractéristiques :

| Niveau | 1 | 2 | 3 |
|---|---|---|---|
| Force | 3 - 10 | 15 - 30 | 45 - 100 |
| Vitalité | 3 - 10 | 15 - 30 | 45 - 100 |
| Magie | 3 - 10 | 15 - 30 | 45 - 100 |
| Resistance magique | 3 - 10 | 15 - 30 | 45 - 100 |

TABLE 4.31 – Valeurs des caractéristiques optionnelles de l'armure pour les pieds

## 4.6 Sorts

### 4.6.1 Sort – Bleed

*Bleed* ou saigné en français, est un sort qui inflige des dégâts continuent à un ou plusieurs ennemis. Il inflige **200%** de dégâts par seconds pendant **5** secondes dans un rayon de **10** mètres.
Le sort coûte **50** mana.

Disponible pour les armes suivants :

- Épée à une main
- Hache à une main
- Épée à deux mains
- Hache à deux mains

### 4.6.2 Sort – Cripple

*Cripple* ou estropier en français, est un sort qui ralentit la vitesse de déplacement de l'ennemi de **80%** et aussi sa vitesse d'attaque de **80%** pendant **5** secondes. Le sort afflige aussi **140%** de dégâts.
Le sort coûte **50** mana.

Disponible pour les armes suivants :

— Épée à une main
— Hache à une main

### 4.6.3 Sort – Rapid Attack

*Rapid Attack* ou attaque rapide en français, est un sort qui augmente la vitesse d'attaque du personnage de **50%** pendant **5** secondes.
Le sort coûte **50** mana.

Disponible pour les armes suivants :

— Épée à une main
— Hache à une main
— Arc

### 4.6.4 Sort – Smash

*Smash* ou écrasé en français, est un sort qui inflige **300%** de dégâts dans un rayon de **10** mètres.
Le sort coûte **50** mana.

Disponible pour les armes suivants :

— Épée à deux mains
— Hache à deux mains

### 4.6.5 Sort – Knock-out & Freeze

*Knock-out* ou assommé en français, est un sort qui assomme les ennemis pendant **3** secondes et inflige **140%** de dégâts dans un rayon de **6** mètres.
Le sort coûte **50** mana.

Disponible pour les armes suivants :

— Épée à deux mains
— Hache à deux mains

### 4.6.6 Sort – Freeze

*Freeze* ou gelé en français, est un sort qui gèle les ennemis pendant **3** secondes et inflige **140%** de dégâts magiques dans un rayon de **6** mètres.
Le sort coûte **50** mana.

Disponible pour les armes suivants :

— Bâton magique
— Baguette magique

### 4.6.7 Sort – Armor-up

*Armor-up* ou augmenter l'armure en français, est un sort qui augmente l'armure et la résistance magique du personnage de **100%** pendant **5** secondes.
Le sort coûte **50** mana.

Disponible pour les armes suivants :

— Bâton magique
— Baguette magique

### 4.6.8 Sort – Fast

*Fast* ou rapide en français, est un sort qui augmente la vitesse de déplacement du personnage de **100%** pendant **5** secondes.
Le sort coûte **50** mana.

Disponible pour les armes suivants :

- Épée à une main
- Hache à une main
- Arc

### 4.6.9 Sort – Zone-Damage

*Zone-Damage* ou dégâts de zone en français, est un sort qui inflige **600%** de dégâts dans un rayon de **15** mètres.
Il inflige **600%** de dégâts magique si le personnage est équipé d'un bâton magique ou d'une baguette magique.
Le sort coûte **50** mana.

Disponible pour les armes suivants :

- Bâton magique
- Baguette magique
- Arc

## 4.7 Ennemis

Il y a 3 ennemis de bases de différentes tailles : petite, moyenne et grande.

## 4.8 Monde

Le monde du jeu est basé sur un concept simple. Le personnage commence dans un village (base principale) où il peut trouver un coffre et un portail pour se téléporter. Le portail mène vers un niveau (le terrain de combat) généré d'où il ne peut plus revenir dans le village sans mourir. Ce niveau est composé de 3 zones de difficultés, chacun plus difficile que l'autre.

### 4.8.1 Village

En cours d'un jeu, le joueur apparaît dans un village où il y a a disposition, un coffre de stockage et des téléporteurs pour aller dans les différentes cartes.

# Chapitre 5

# Analyse organique

## 5.1 Monogame

Monogame fournit des fonctionnalités indispensables pour le développement d'un jeu. Il offre la boucle de jeu de base qui permet de mettre à jour tous les éléments du programme (voir figure 5.1) ainsi que beaucoup de classe utiles pour le développement de jeux.

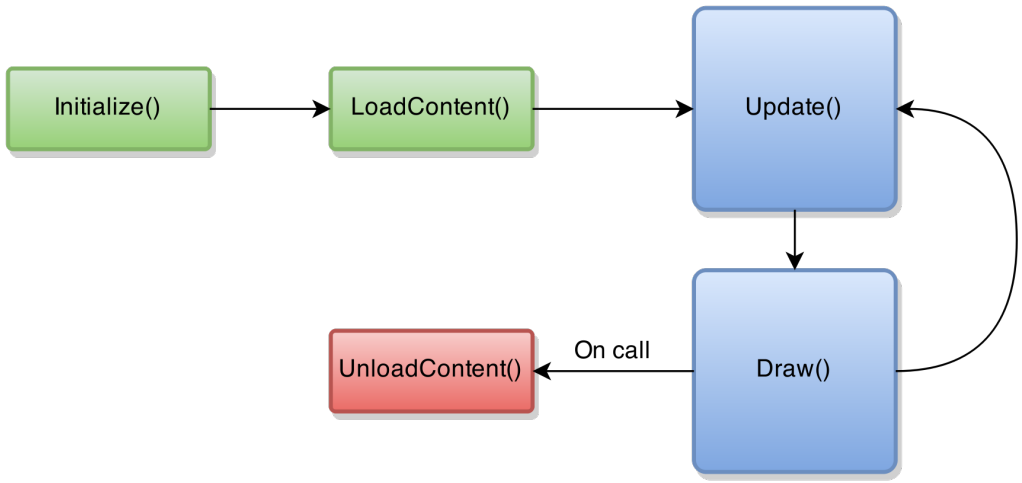



FIGURE 5.1 – Boucle du framework MonoGame

### 5.1.1 Les boucles de jeu

#### La boucle *LoadContent*

La boucle *LoadContent* sert à initialiser du contenu. Cette méthode peut se retrouver dans des classes pour initialiser le contenu de la classe comme les textures par exemple. La méthode est appelée une fois lors de l'initialisation.

#### La boucle *UnloadContent*

La boucle *UnloadContent* sert à retirer les variables comme les textures pour être sûr de leurs déchargements. La méthode est appelée seulement implicitement ou lors de la fermeture du programme.

#### La boucle *Update*

La boucle *Update* sert à mettre à jour tous les contenus non-graphique du jeu. Cette méthode peut se retrouver dans des classes qui nécessitent d'être mis-à-jour régulièrement quand il est possible. Elle est appelée constamment par Monogame.

#### La boucle *Draw*

La boucle *Draw* dessine le contenu sur l'écran du jeu. Cette méthode peut se retrouver dans des classes pour dessiner son contenu. Il est appelé constamment quand il est possible, car la boucle Update possède la priorité et si celui-ci est chargé, Monogame ignorera la méthode *Draw*.

## 5.2 La structure du projet

### 5.2.1 Architecture du jeu

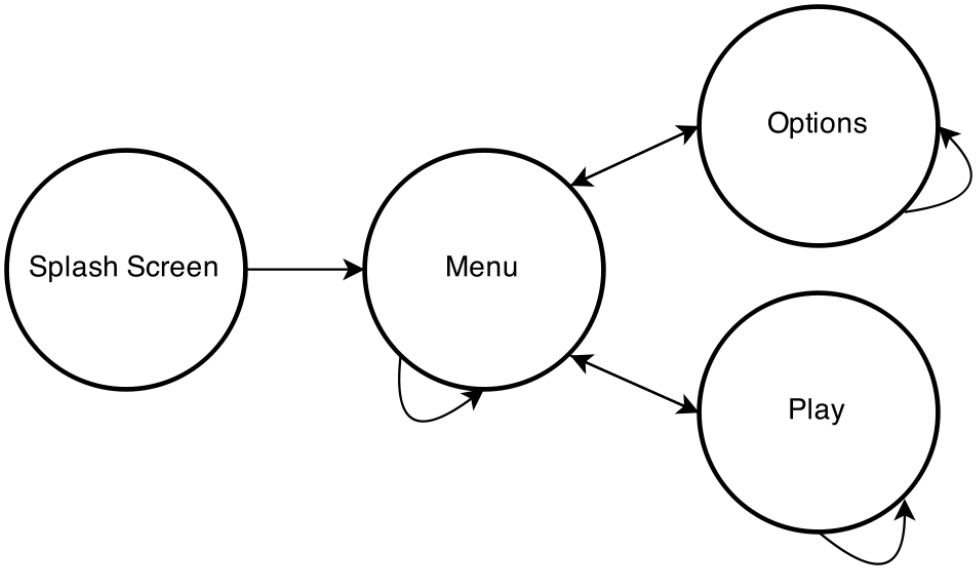



FIGURE 5.2 – Différents états du programme

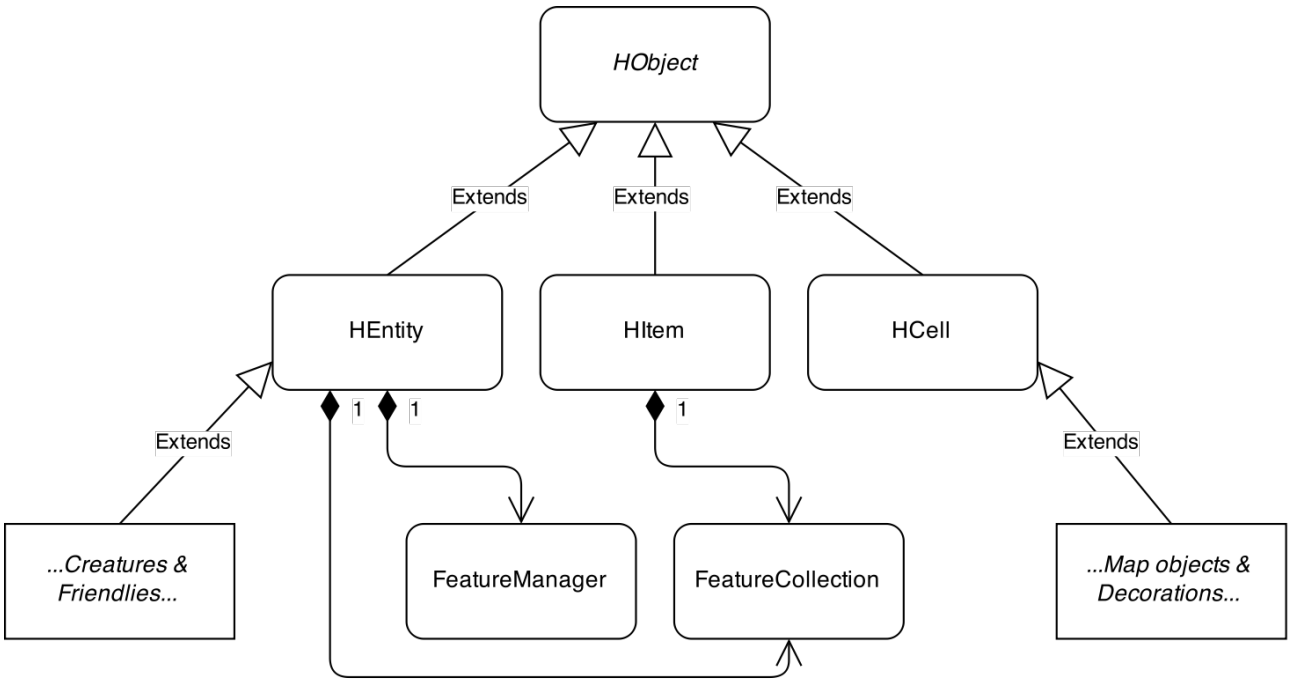



FIGURE 5.3 – Schémas de hiérarchisation des objets du monde

### 5.2.2 Classe de jeu de base

Tous les éléments du jeu héritent de la classe HObject, ce qui permet une grande simplification pour le développement (voir figure 5.4). Cette classe abstraite présente :
(***f*** signifie que c'est une méthode de la classe et ***p*** signifie que c'est une propriété de la classe.)

- ***p*** **IsWalkable** permet de savoir si l'élément peut être superposé avec un autre.
- ***p*** **Position** position précise relative à la carte.
- ***f*** **LoadContent** permet de charger du contenu.
- ***f*** **UnloadContent** permet de décharger du contenu.
- ***f*** **Update** permet de mettre à jour l'objet.
- ***f*** **Draw** permet de dessiner l'objet sur l'écran.

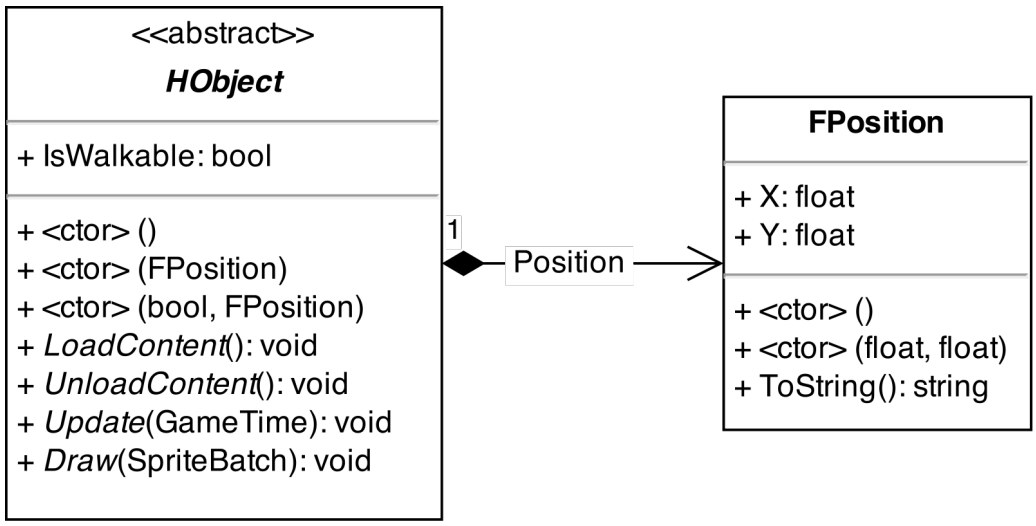



FIGURE 5.4 – Diagramme de classe de HObject

## 5.3 Gestion de la carte

La carte est constituée d'un tableau à deux dimensions de classe *HCell*. Il possède une hauteur et une largeur ainsi qu'une liste de tous les objets et ennemis présent. Des fonctions de bases pour *get/set* les cellules sont nécessaires :

- Prendre une cellule **désignée**
- Prendre les cellules **voisines** d'une cellule
- Prendre les cellules **voisines non-accessible** d'une cellule
- Prendre le **nombre** de cellules voisines non-accessible d'une cellule
- Prendre la cellule se trouvant **dessous** d'une cellule
- Prendre la cellule se trouvant **dessus** d'une cellule
- Prendre la cellule se trouvant **à gauche** d'une cellule
- Prendre la cellule se trouvant **à droite** d'une cellule
- Prendre une cellule **aléatoire** et qui est **accessible**

### 5.3.1 Cellules

Les cellules héritent de *HObject*[1] et ils possèdent des limites et un type. Les limites sont du type *FRectangle*[2] ce qui permet de détecter les intersections avec les autres objets. Le type de la cellule est un string qui permet de l'identifier, ce string peut être lié à une texture du *TextureManager*[3].

### 5.3.2 Génération de la carte

La génération est faite grâce à un algorithme d'automate cellulaire modifié pour créer des apparences de cavernes. Chaque cellule, selon son état et ses voisins peut changer. Dans ce cas, ce sont des cellules carrées placées dans un tableau à deux dimensions. Une cellule possède donc au maximum 8 voisins en comptant les diagonales. Il est bien sûr possible d'augmenter l'étendue des voisins pour compter plus de variables et augmenter la taille des salles de la caverne.
Afin de créer un effet aléatoire, il est nécessaire d'initialiser le tableau avec environ 40% de cellules pleines et avec des bords pleins, afin d'éviter que le joueur ne parte de la carte. L'algorithme peut

1. Voir section 5.2.2 pour les objets de bases
2. Voir section 5.5.2 pour la classe float rectangle
3. Voir section 5.6.2 pour la gestion de texture

ensuite être appliquer au tableau. Il est nécessaire de l'appliquer plusieurs fois pour augmenter les aspects *naturels* d'une caverne (voir figure 5.5).

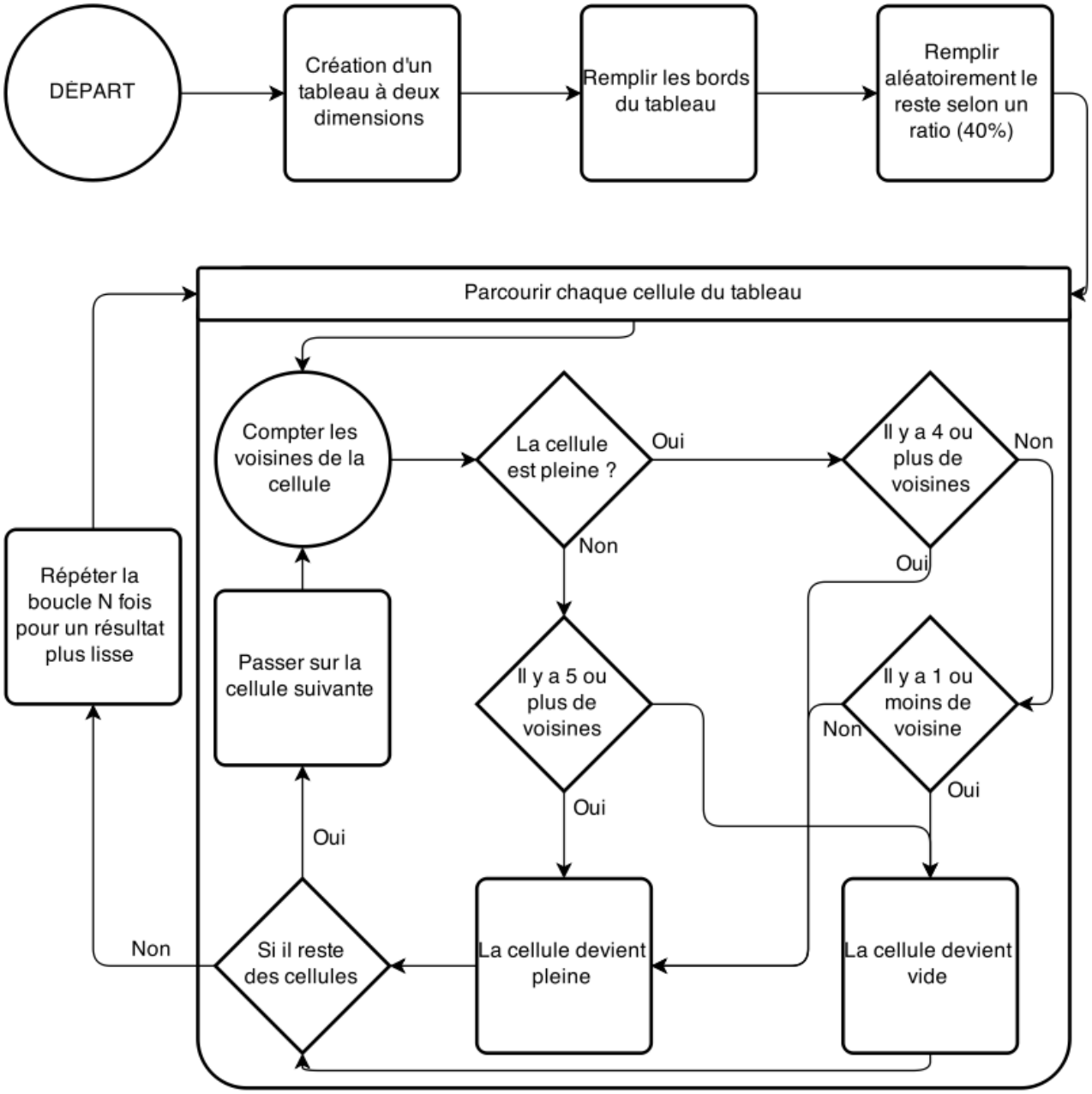



FIGURE 5.5 – Diagramme de flux de la génération de carte (cavernes)

## 5.4 Éléments du jeu

### 5.4.1 Caractéristiques

Les caractéristiques font l'essence même du jeu. Il faut qu'il soit placé dans toutes les entités du jeu, parce que ceux-ci se reposent dessus pour tous leurs actions. Les objets ou "items" sont des choses que les entités peuvent porter pour augmenter leur caractéristique globale (voir figure 5.6). Pour gérer tous les calculs entre les caractéristiques, un manager de caractéristiques doit être placé sur l'entité. Le manager analysera tous les éléments possédant des caractéristiques et fera une somme globale pour fixer les caractéristiques finales de l'entité.

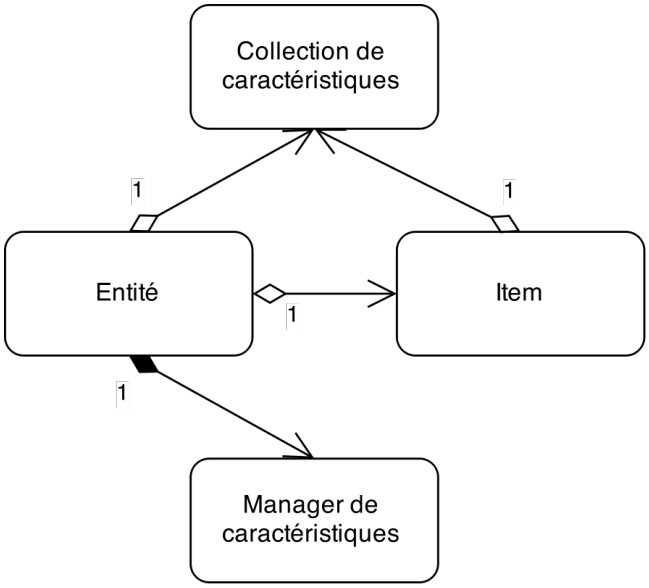



FIGURE 5.6 – Schémas simple de fonctionnement des caractéristiques

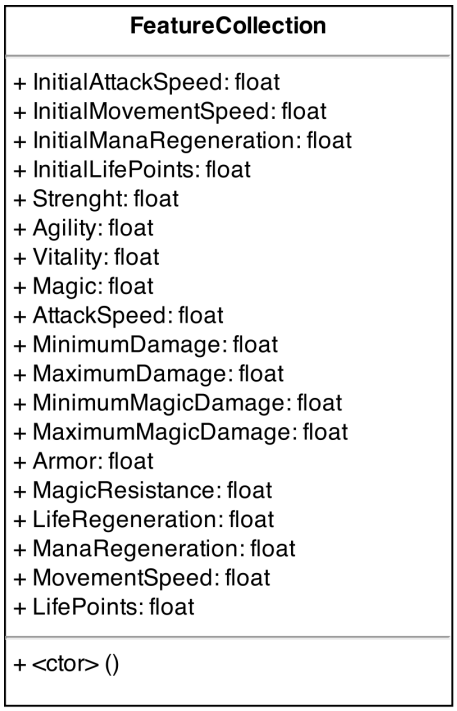



FIGURE 5.7 – Diagramme de classe de la collection de caractéristiques

Le *FeatureCollection* est la classe qui permet de stocker ses caractéristiques. Elle ne fait rien de spécial sauf stocker ses valeurs. C'est pourquoi trois propriétés de caractéristiques existent sur une entité (voir listing 5.1) :

**InitialValue** Les caractéristiques initiales de l'entité

**MaximizedFeatures** Les caractéristiques maximum en tenant compte des sorts et des objets qu'il porte.

**ActualFeatures** Les caractéristiques en tenant compte des sorts et des objets qu'il porte et aussi de dégâts et effets néfastes de l'entité.

Voir figure 5.7 pour le diagramme de classe d'une collection de caractéristiques.

Listing 5.1 – Exemple de calcul de la force maximal d'une entité

```
/// <summary>
/// Calculates the total strenght within the items, spells and initial
    values
/// </summary>
/// <returns>Total strenght</returns>
public float GetTotalStrenght() {
  // Gets the initial strenght of the entity
  float str = this.InitialFeatures.Strenght;
  // For each currently active spell, we add the strenght of it
  for (int i = 0; i < this.ActiveSpells.Count; i++)
  {
    str += this.ActiveSpells[i].Features.Strenght;
  }
  // For each worn item of the entity, we add the strenght of it
  for (int i = 0; i < this.ActiveItems.Count; i++)
  {
    str += this.ActiveItems[i].Features.Strenght;
  }
  return str; // return the total

}
```

### 5.4.2 La classe entité

La classe entité (*HEntity*, voir figure 5.8) est une classe abstraite qui forme la base de toutes les entités "intelligentes" du jeu. C'est-à-dire des unités qui possèdent des caractéristiques pour interagir avec le joueur. Pour une explication de InitialFeatures, ActualFeatures et MaximizedFeatures, voir la section

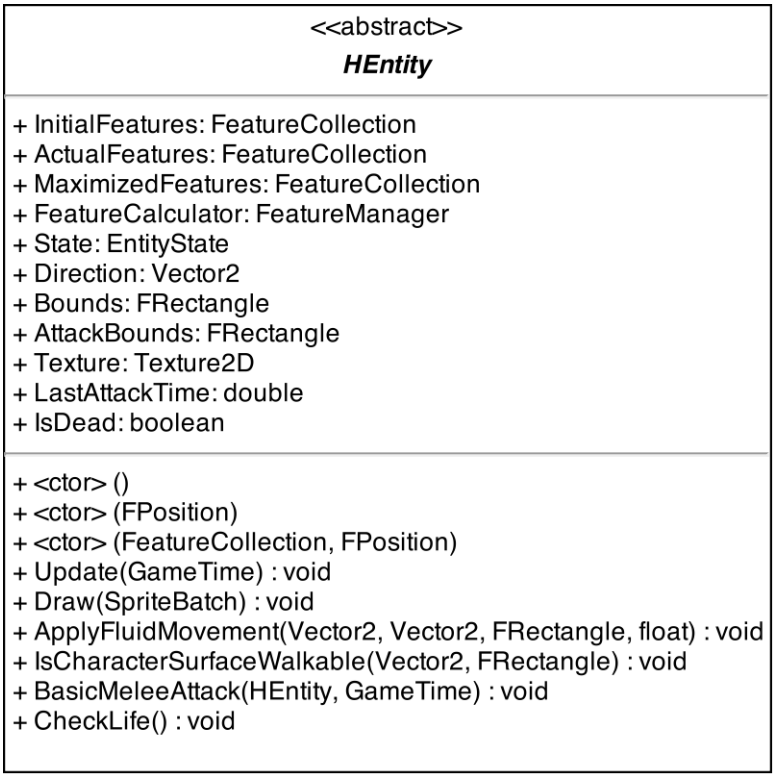



FIGURE 5.8 – Diagramme de classe d'une entité du jeu

5.4.1 sur les caractéristiques.

#### Les états

Une entité doit également posséder un état pour savoir ce qu'il est en train de faire. Ses états sont les suivants :

**S0 Idle** Il ne fait rien

**S1 Running** Il est en déplacement

**S2 MeleeAttacking** Il exécute une attaque au corps-à-corps

**S3 RangeAttacking** Il exécute une attaque à distance

**S4 SpellCasting** Il exécute un sort

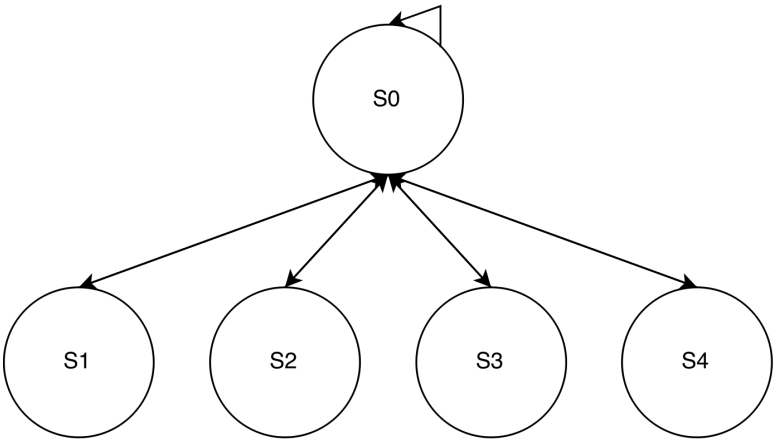



FIGURE 5.9 – Machine d'état d'une entité

#### Le déplacement d'une entité

Il est aussi important mettre en place un mécanisme de déplacement de l'entité (voir section 5.4.4) et avec ça une direction au quelle l'unité fait face. Cette direction doit être mise-à-jour à chaque déplacement. La direction sert de référence pour tirer des projectiles dans le bon endroit.

#### Les limites de collisions

Les limites ou *Bounds* servent de zone de collision. Tous autres objets du jeu possédant la propriété *IsWalkable* (voir section 5.2.2) à false, se trouvera bloqué par celui-ci. De même que tout projectile ou attaque fait dans la zone engendreront des dégâts à l'entité.

#### La portée d'action

Une entité doit pouvoir interagir avec les autres, c'est pourquoi il faut aussi ajouter des limites d'attaques ou *attack bounds*. Ce qui permet de connaître la portée des attaques de cette entité. Lorsqu'une autre entité se trouve dans cette zone celui-ci peut se faire attaquer. Par exemple, lorsqu'une portée d'une entité touche les limites (correspondant à la texture) d'une autre entité, celui-ci peut désormais l'attaquer (voir figure 5.10).

#### La texture

Les textures sont simples d'intégration puisqu'il fait partie de fonctionnalités que monogame offre. Il faut alors charger une *Texture2D* et le dessiner. Les animations ne sont pas faites parce que ce n'est pas une priorité.

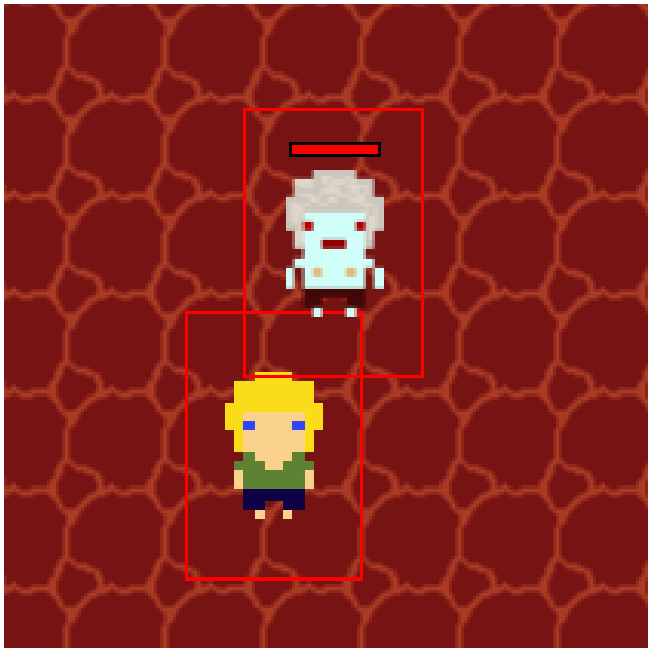


FIGURE 5.10 – Portée des entités

Listing 5.2 – Affichage de la texture sur l'écran

```
1 Vector2 position =
     ScreenManager.Instance.GetCorrectScreenPosition(boundsPosA,
     PlayScreen.Instance.Camera.Position);
2 spriteBatch.Draw(this.Texture, position, Color.White);
```

#### Le temps de dernière attaque

Il est important de garder en mémoire le temps de la dernière attaque d'une entité, c'est ce qui permet de régler le temps qu'il y a entre chaque attaque avec les caractéristiques de celui-ci. Pour connaître le nombre d'attaques par seconde ($v$), il fait diviser 1 par la vitesse d'attaque de l'entité ($a$). ($v = \frac{1}{a}$). Si le temps entre la dernière attaque et le temps actuel est plus grand ou égal à $v$ ($t_1 - t_2 >= v$), l'entité est libre d'effectuer une action.

#### La mort de l'entité

Si l'entité se retrouve avec 0 dans ses points de vie, celui-ci est mort. Une fois cette propriété à *true*, l'unité se fait effacer par le gestionnaire de partie.

### 5.4.3 Personnage jouable

Le personnage jouable est une entité qui est contrôlée par l'utilisateur. Il possède donc des méthodes et propriétés qui lui sont propres. Le personnage a en plus :

- Des sorts
- Des objets (*items*)
- Des actions par périphériques d'entrée

### 5.4.4 Déplacement

Le déplacement d'une entité se fait avec des vecteurs. La méthode prend deux vecteurs : la position et le point de déplacement.

$$\vec{v} = \vec{v_1} - \vec{v_2}$$

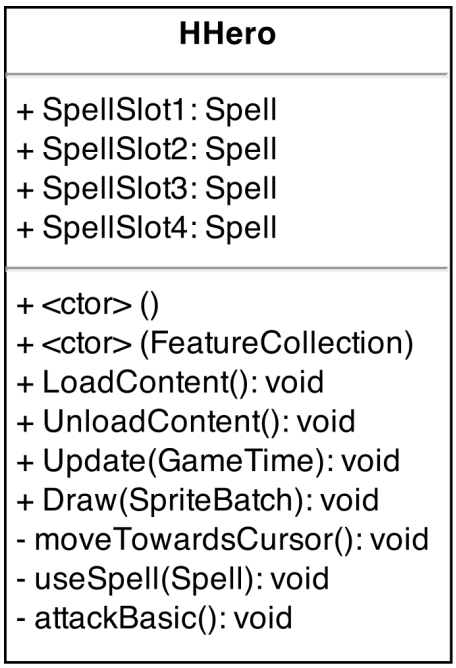



FIGURE 5.11 – Diagramme de classe du personnage jouable

Il faut ensuite normaliser ce vecteur qui est la direction. $|\vec{v}|$. Pour connaître la nouvelle position selon le temps et vitesse de déplacement cette formule est appliquée :

$$\vec{v} * \Delta t * s$$

$\vec{v}$ : Vecteur de direction normalisé
$\Delta t$ : Temps écoulé depuis le dernier *Update*
$s$ : Vitesse de déplacement de l'entité
Il faut ensuite addition ce vecteur à la position actuelle du personnage.

#### Gestion des obstacles

Lors d'un déplacement il est important de prendre en compte les obstacles autour de l'entité, c'est pourquoi il faut faire une vérification de la nouvelle position avec les éléments du jeu, cellule de la carte et autres entités. Si celui-ci est obstrué, on retire un des axes du vecteur de déplacement et réessaye, s'il est de-nouveau obstrué, il faut remettre l'axe retiré et essayer avec l'autre. S'il n'y a toujours pas de solution, il ne faut pas valider le déplacement (voir figure 5.12).

### 5.4.5 Ennemis

Les ennemis sont des entités possédant une "intelligence" contrôlée par l'ordinateur. Cette unité possède plus de zone de détection pour faire illusion d'une intelligence pour que le jeu soit plus jouable. Les propriétés ajoutées sont :

**FieldOfView** La zone de détection

**IsAlerted** L'ennemi a détecté le joueur

Pour que cette entité détecte des joueurs, une zone de détection. Cette zone de détection est faite avec un FRectangle (voir section 5.5.2) qui permet de voir si les limites du joueur et de l'ennemi se touchent. Une fois détecter cette zone agrandie et l'entité passe en mode alerte jusqu'à qu'il perde de vue le joueur (voir figure 5.13).

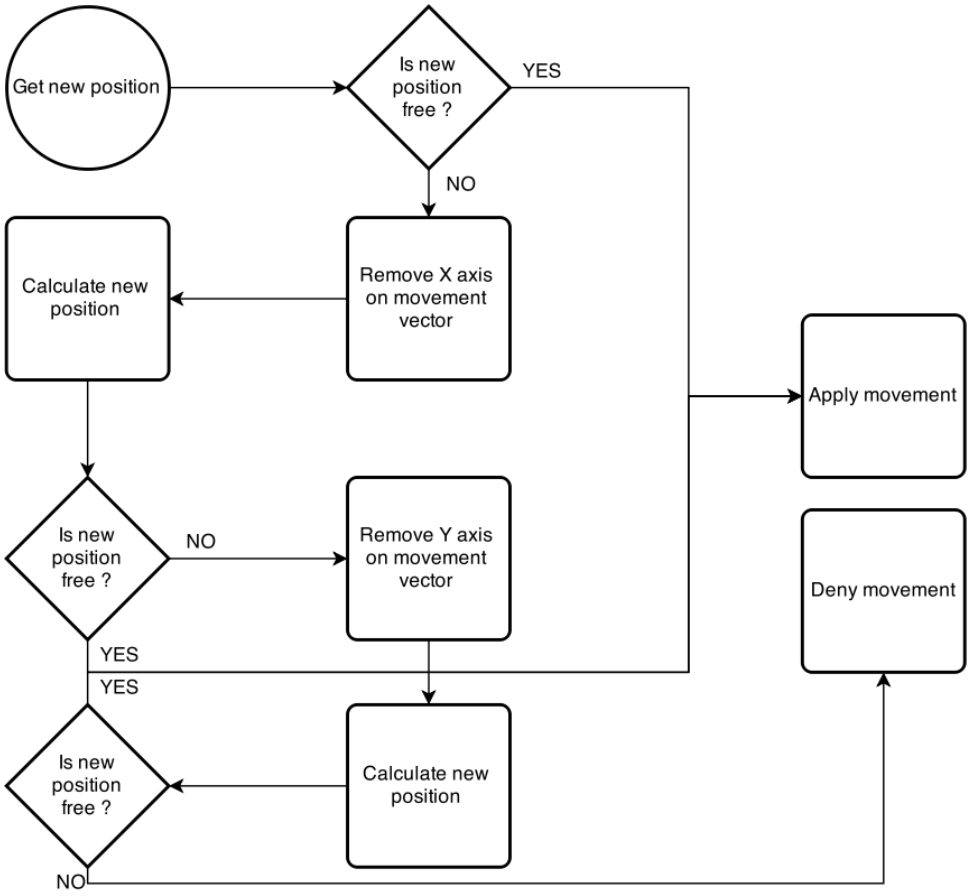



FIGURE 5.12 – Diagramme de flux du management d'obstacle

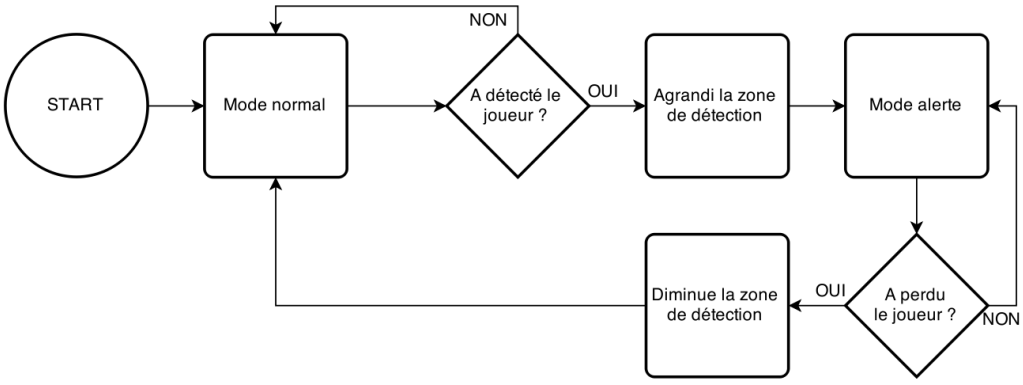



FIGURE 5.13 – Diagramme de flux de la détection des hostiles

### 5.4.6 Objets - *items*

Les *items* sont des objets que le personnage jouable peut ramasser et stocker dans son inventaire. Il peut de même les équiper sur soi. Les items possèdent des caractéristiques (voir section 5.4.1), une fois que le personnage joueur équipe un item, il ajoute les caractéristiques de l'item sur soi grâce au manager de caractéristiques (voir listing 5.1 pour un exemple de calcul du manager).

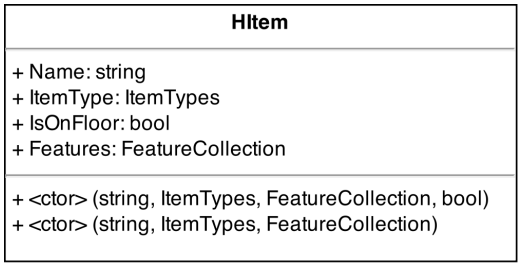



FIGURE 5.14 – Diagramme de classe de HItem

### 5.4.7 Attaques

#### Corps-à-corps

Les attaques corps à corps peuvent seulement toucher une autre entité. La cible choisie doit se trouver dans la zone d'attaque au corps-à-corps. Lui infligeant ainsi des dégâts selon les caractéristiques. Voir listing 5.3.

Listing 5.3 – Attaque au corps-à-corps

```
1  /// <summary>
2  /// Basic melee attack
3  /// </summary>
4  /// <param name="target">Targeted enemy</param>
5  public void BasicMeleeAttack(HEntity target, GameTime gameTime)
6  {
7    // Gets the current time of the game
8    double currentTime = gameTime.TotalGameTime.TotalSeconds;
9    // Calculates the number of seconds per attack
10   float secPerAttack = 1f / this.ActualFeatures.AttackSpeed;
11
12   // If the attack can be made
13   if (currentTime - this._lastAttackTime >= secPerAttack)
14   {
15     float minDmg = this.ActualFeatures.MinimumDamage;
16     float maxDmg = this.ActualFeatures.MaximumDamage;
17     float receivedDamage = (float)this._rand.NextDouble();
18     receivedDamage = (maxDmg - minDmg) * receivedDamage + minDmg;
19     float realReceivedDamage =
         this.FeatureCalculator.GetReceivedPhysicalDamage(receivedDamage);
20     realReceivedDamage = realReceivedDamage *
         (this.ActualFeatures.Strenght / 100 + 1);
21     target.ActualFeatures.LifePoints -= realReceivedDamage;
22     this._lastAttackTime = currentTime;
23   }
24 }
```

#### Distance

Les attaques à distance peuvent seulement toucher une autre entité avec le projectile. Le projectile ne peut pas aller plus loin que la zone d'attaque à distance et peut seulement toucher une entité, lui infligeant ainsi des dégâts selon les caractéristiques.

### 5.4.8 Sorts

## 5.5 Outils de développement

### 5.5.1 Primitives 2D

Malheureusement, Monogame n'offre pas de primitives 2D pour dessiner sur l'écran, c'est-à-dire les points, les ligne, les rectangles, les polygones et les ellipses. C'est pourquoi il est nécessaire de faire une classe primitive 2D (voir figure 5.15).

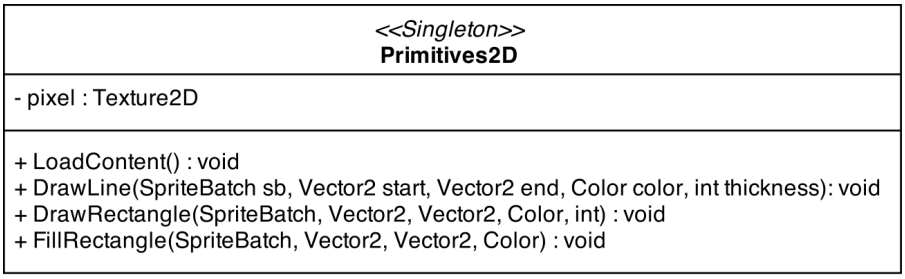



FIGURE 5.15 – Diagramme de classe des primitives 2D

#### Les méthodes

**LoadContent** est ce qui permet de créer le pixel par défaut avec une teinture blanche. Ce pixel du type Texture2D est indispensable, parce que c'est celui-ci qui sera manipulé pour faire des diverses formes.

**DrawLine** dessine une line entre deux points spécifiés.

**DrawRectangle** dessine un rectangle entre deux points spécifiés.

**FillRectangle** dessine un rectangle plein entre deux points spécifiés.

### 5.5.2 Intersections

Les intersections sont la base des interactions entre les éléments du jeu. C'est pourquoi chaque entité et cellule du jeu possèdent des limites. Monogame n'offre qu'une classe Rectangle avec des entiers, mais pour plus de précisions il faut des nombres à virgule. C'est pourquoi la classe FRectangle (voir figure 5.16) est faite. Il possède aussi une méthode d'intersection qui détecte si le rectangle même croise un autre.

### 5.5.3 Gestion des entrées

Il a fallu penser façon la plus efficace pour gérer les entrées. L'objectif est d'éviter de mettre des entrées directement dans la classe. C'est pourquoi la classe InputManager est faite. Il permet de mettre à jour tous les entrées et de les placer dans des listes correspondantes :

**DownKeys** Les touches qui viennent d'être appuyées.

**PressedKeys** Les touches qui sont appuyées.

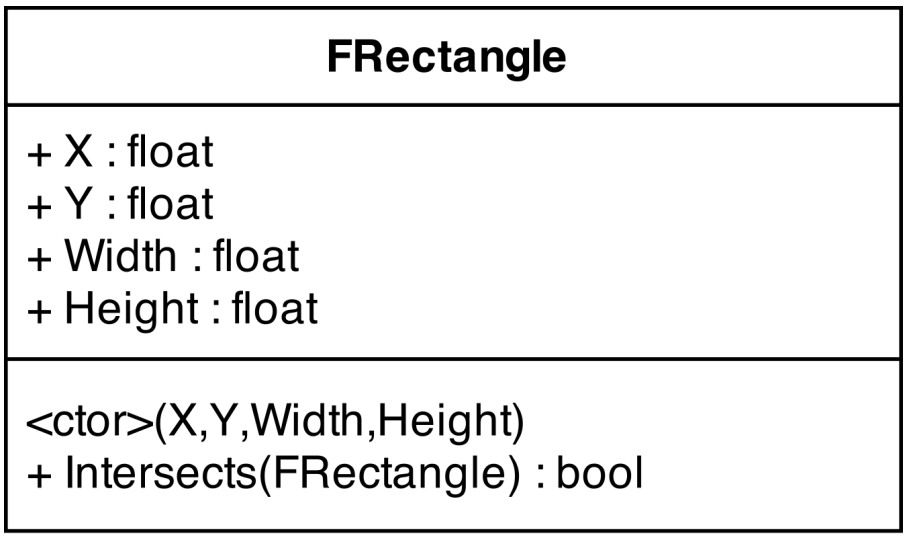



FIGURE 5.16 – Diagramme de classe de FRectangle

**ReleasedKeys** Les touches qui viennent d'être relachées.

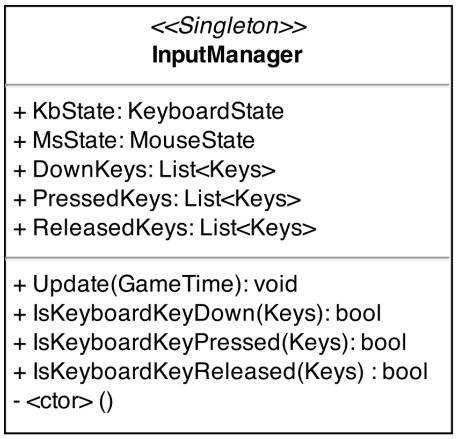



FIGURE 5.17 – Diagramme de classe du manager d'entrées

Les touches ne peuvent pas se retrouver dans plusieurs listes en même temps.

Listing 5.4 – Mise-à-jour des entrées

```
1  /// <summary>
2  /// Updates the states of the keyboard
3  /// </summary>
4  private void UpdateKeyboardInput()
5  {
6    if (MainGame.Instance.IsActive)
7    {
8      this.KbState = Keyboard.GetState();
9
10     // Verifies all the keys of the keyboard
11     foreach (Keys key in Enum.GetValues(typeof(Keys)))
12     {
13       // If the key is not pressed and it is not in the release key
           list and is in one of the other two list
```

```
14        // we add it to the released key list and remove it in the others
15        if (this.KbState.IsKeyUp(key) &&
              !this.IsKeyboardKeyReleased(key) &&
16        (this.IsKeyboardKeyPressed(key) || this.IsKeyboardKeyDown(key)))
17        {
18          this.DownKeys.Remove(key);
19          this.PressedKeys.Remove(key);
20          this.ReleasedKeys.Add(key);
21        }
22        else if (this.KbState.IsKeyUp(key) &&
              this.IsKeyboardKeyReleased(key)) // If it is already in the
              released key list
23        {         // remove it
24          this.ReleasedKeys.Remove(key);
25        }
26        else
27        {
28          // If the key is down and is not yet pressed
29          if (this.KbState.IsKeyDown(key) &&
                !this.IsKeyboardKeyDown(key) &&
                !this.IsKeyboardKeyPressed(key))
30          {
31          // remove it from the other lists (to be sure)
32            this.ReleasedKeys.Remove(key);
33            this.PressedKeys.Remove(key);
34            this.DownKeys.Add(key); // and add it to the down key list
35          }
36          else if (this.KbState.IsKeyDown(key) &&
                !this.IsKeyboardKeyPressed(key)) // If it's already in the
                down key list
37          {

                // and isn't in the pressed list
38            this.DownKeys.Remove(key); // remove it from the other lists
39            this.ReleasedKeys.Remove(key);
40            this.PressedKeys.Add(key); // add it to the pressed list
41          }
42        }
43      }
44    }
45    else
46    this.KbState = new KeyboardState();
47  }
```

### 5.5.4 Sérialisation XML

Il est important de pouvoir sérialisé les objets en XML. Ceci permet d'ajouter du contenu facilement et sans toucher au code source. C'est pourquoi la classe XmlManager est générique. Il permet de sérialiser tous les objets si ceux-ci sont appropriés pour la sérialisation (voir figure 5.18).

**Load** permet de charger un XML et de rendre un objet (voir listing 5.5).

**Save** permet de sauvegarder un objet en XML.

Listing 5.5 – Désérialisation d'un XML

```
T instance;
using (TextReader reader = new StreamReader(path))
{
  XmlSerializer xml = new XmlSerializer(TypeClass);
  instance = (T)xml.Deserialize(reader);
}
return instance;
```

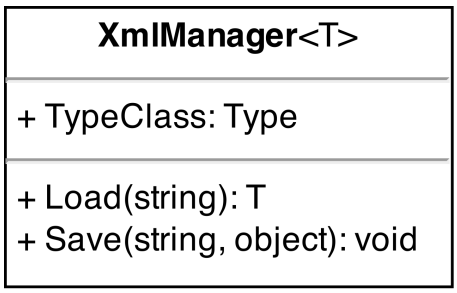



FIGURE 5.18 – Diagramme de classe du manager de sérialisation XML

## 5.6 Vue

### 5.6.1 Gestion de l'écran

Pour centralisé les différents modes d'affichages (écran de présentation, menu et jeu), un gestionnaire d'écran qui centralise tous ces modes est nécessaire. Ce gestionnaire s'accroche sur les boucles de monogame et possède un écran actif. Cet écran actif est mise-à-jour à travers ces boucles, ce qui permet une transition fluide entre les modes. La taille d'un écran est également mémorisée dans ce gestionnaire pour garder une uniformité des écrans (voir figure 5.19).

#### Les transitions

Pour passé d'un mode d'affichage à un autre, une transition est nécessaire. C'est pourquoi la classe ScreenManager contient un compteur qui garde le temps passer sur un mode d'affichage. Avec une méthode, il suffit de changer l'écran actuel après un certain temps ou lorsqu'un évennement a eu lieu. Voir le listing 5.6.

Listing 5.6 – Transition d'écran

```
/// <summary>
/// Transitions the screen to another one
/// </summary>
/// <param name="nextScreen"></param>
public void Transition(GameScreen nextScreen)
{
  // resets the transition variables
  this._transitionTime = 0;
  this._transitionDelayActive = false;
  this._transitionFirstCount = -1.0d;

  this.UnloadContent(); // unloads the content of the current screen
  this._currentScreen = nextScreen; // place the new screen
  this._currentScreen.LoadContent(); // loads the new screen
}
```

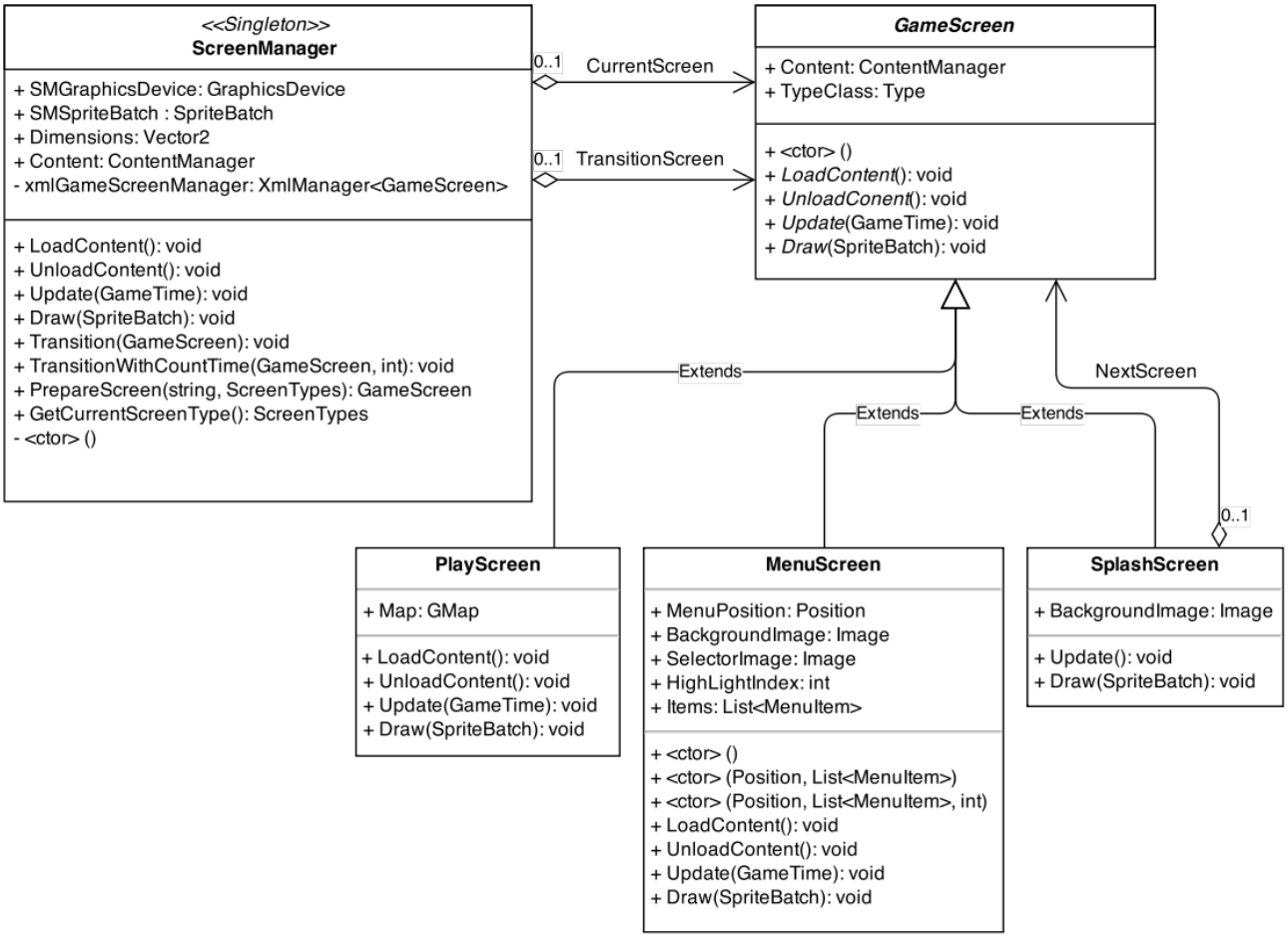



FIGURE 5.19 – Diagramme de classe de la gestion d'écrans

```
16
17 /// <summary>
18 /// Activates a screen transition for the specified time
19 /// </summary>
20 /// <param name="nextScreen">Next screen that will appear</param>
21 /// <param name="time">Time before transition</param>
22 public void Transition(GameScreen nextScreen, int time)
23 {
24   this._transitionScreen = nextScreen;
25   this._transitionTime = time;
26   this._transitionDelayActive = true;
27   this._transitionFirstCount = -1.0d;
28 }
```

### L'écran de base

L'écran de base (GameScreen) est une classe absraite pour garder une même base des écrans. Un écran possède un manager de contenu qui permet la gestion, chargement et déchargement des contenus comme : les textures et les fonts. Une variable de type y est ajouté pour plus facilement identifier le réel type d'écran. Bien sûr, la boucle monogame est aussi ajoutée, pour offrir des possibilités de développement des écrans.

### L'écran de présentation

L'écran de présentation ou *SplashScreen*, est un simple écran possédant une image de fond qui porte le nom de l'auteur ainsi que son logo. Cet écran est en mode transition, cet-à-dire qu'il changera tout seul de mode ou lorsque l'utilisateur appuie sur une touche.

### L'écran de menu

Le menu sert d'instance d'interaction entre le joueur et le programme. C'est pourquoi pour favoriser l'avancement du projet, il faut que le menu soit générique. C'est pourquoi son contenu s'ajoute avec des fichiers XML à l'aide du XMLManager, voir listing 5.7.

Listing 5.7 – Menu en XML

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <MenuScreen>
3    <Image>
4    <Path>SplashScreen/main2</Path>
5    <Position>
6    <X>640</X>
7    <Y>360</Y>
8    </Position>
9    </Image>
10   <Items>
11     <MenuItem>
12     <Image>
13       <Text>Play</Text>
14       <Position>
15         <X>100</X>
16         <Y>200</Y>
17       </Position>
18     </Image>
19     <LinkType>1</LinkType>
```

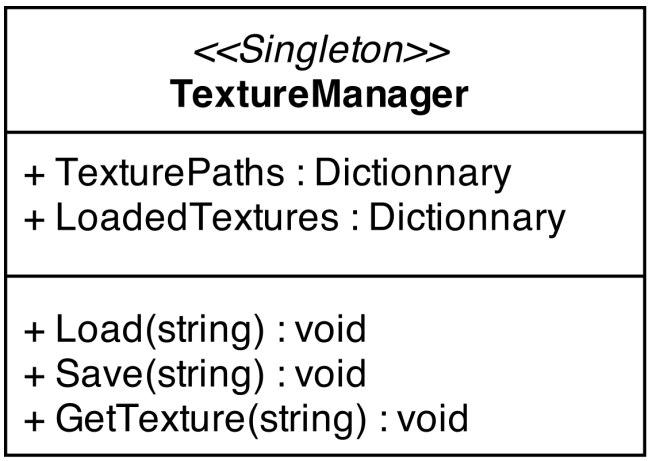



FIGURE 5.20 – Diagramme de classe de la gestion de textures

```
20      <Id>1</Id>
21      </MenuItem>
22      <MenuItem>
23      <Image>
24        <Text>Options</Text>
25        <Position>
26          <X>100</X>
27          <Y>230</Y>
28        </Position>
29      </Image>
30      <LinkType>1</LinkType>
31      <Id>2</Id>
32      </MenuItem>
33    </Items>
34  </MenuScreen>
```

### 5.6.2 Gestion des textures

Pour maintenir un jeu générique, il faut aussi rendre la texture disponible depuis un endroit pour le centraliser. C'est pourquoi les textures sont tous chargé avec des fichiers XML au lancement du jeu. Ce qui permet une modification facile des textures (voir figure 5.20). La classe charge d'abord une liste XML qui contient tous les liens des textures, ensuite il charge les textures.

# Chapitre 6

# Conclusions

## 6.1 Apport personnel

Dans le tableau ci-dessous, une décortication des différents points du projet avec une estimation de mon apport personnel.

| Élément | Apport | Commentaires |
|---|---|---|
| **Outils** | **∼60.0%** | |
| Class FRectangle | 80% | Uniquement la partie d'intersection avec un autre rectangle a été récupérée. |
| Primitives2D | 60% | La création d'un pixel et la fonction DrawLine ont été récupéré. |
| Gestionnaire d'entrées | 100% | Monogame ne répondait pas à mes demandes, j'ai dû créer une classe adaptée à mes besoins. |
| Gestionnaire XML | 0% | Le gestionnaire XML générique a été récupérée. |
| **Vue** | **∼87.5%** | |
| Gestion d'écran | 90% | Ces classes ont été créées avec de l'aide de concept de développement Monogame. |
| Textures | 100% | La gestion des textures et leurs traitements sont entièrement faites par moi-même. |
| Affichage | 100% | Tous les éléments affichés lors d'un jeu. |
| Menu | 60% | Le menu a été faite avec l'aide d'un tutoriel. |
| **Éléments du jeu** | **∼100.0%** | Tous les éléments propres au jeu ont été faites par moi. Les entités, les objets, la carte, les déplacements, les zones de détections, etc. La génération de la carte en forme de cavernes a été faite à l'aide de théories. |
| **Total** | **∼ 90.0%** | Une grande majorité du code, de la construction et de l'architecture du jeu a été faites par moi-même. J'ai utilisé de l'aide sur Internet juste par question de dépannage quand je n'avais pas de solution. |

Table 6.1 – Estimation de l'apport personnel

## 6.2 Conclusion

Le travail s'est bien déroulé, mais malheureusement l'estimation du temps de travail a été mal calculée de ma part. Je me suis retrouvé avec une trop grosse quantité et je n'ai pas pu le gérer. Ce qui en fait que le travail de répond pas au cahier des charges. Les parties manquantes sont : la gestion d'objets pour le personnage, la progression du personnage avec les compétences et le boss final.
La partie qui m'a pris le plus de temps était la vue. Monogame tient ses bases de l'ancien framework de jeu XNA de Microsoft, celui-ci ne dessine que le contenu sur l'écran avec des textures et ne gère pas nativement les primitives 2D et les textes, c'est pourquoi il m'a fallu un certain temps d'adaptation.
Un temps considérable a aussi été donner pour la réflexion sur la construction du projet, parce que c'est la première fois que je développe un jeu de cette envergure sur ce framework.

## 6.3 Bilan personnel

Je suis satisfait de mon travail, même s'il reste encore des modifications à faire par rapport a l'architecture du programme. Je suis légèrement déçu de moi-même, car je connais mes compétences et je sais que si j'aurais mis plus de travail dès le début, le travail serait probablement complet. En effet, j'ai trop accumulé de travail sur la dernière partie du diplôme ce qui a causé du bâclage sur mon travail au niveau du code et surtout du rapport.

## 6.4 Perspectives du projet

Depuis le début du projet, je savais déjà que c'est quelque chose que j'ai envie de continuer, c'est pourquoi je vais le terminer et le rendre gratuitement sur Internet lorsque que le projet est terminé.
Ce projet m'a permis de comprendre mieux le développement de jeux-vidéo, ce qui est ma passion. C'est pourquoi je tiens à le terminer et d'en commencer d'autres.

# Chapitre 7

# Annexes

## 7.1 Planning

Le planning du projet se décompose en plusieurs parties :

**Code backbone** Cœur du projet sur lequel le jeu se repose
**Code essentiel** Éléments du jeu qui sont essentiels
**Contenu** Contenu du jeu (Graphiques, Objets, etc.)
**Tests** Tests pour vérifier l'intégralité du jeu
**Poster** Poster préparer pour le projet
**Documentation** Documentation technique du projet

### 7.1.1 Initial

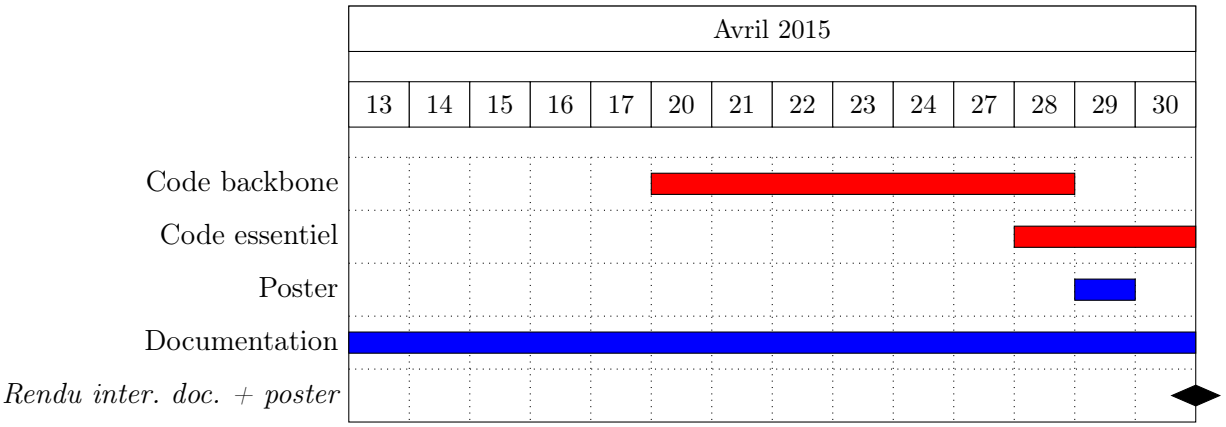



FIGURE 7.1 – Planning initial du mois d'avril

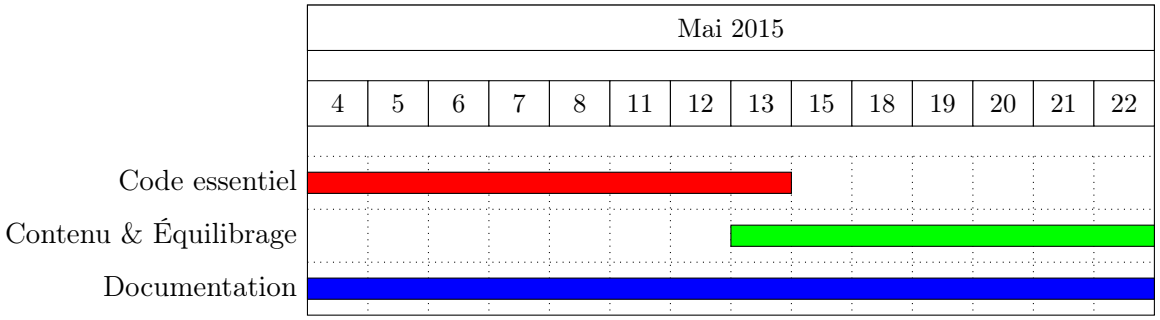



FIGURE 7.2 – Planning initial de la première partie du mois de mai

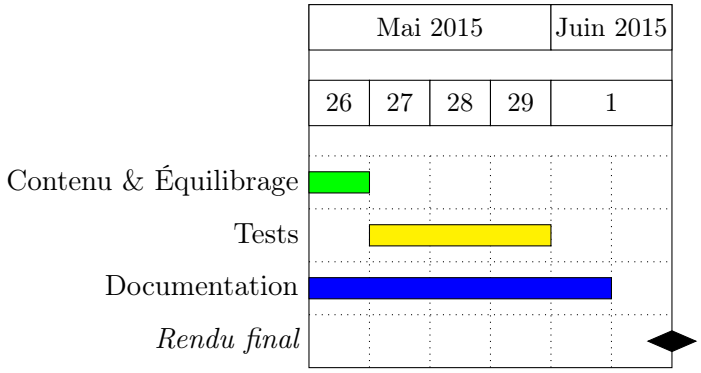



FIGURE 7.3 – Planning initial de la deuxième partie du mois de mai et le mois de juin

### 7.1.2 Final

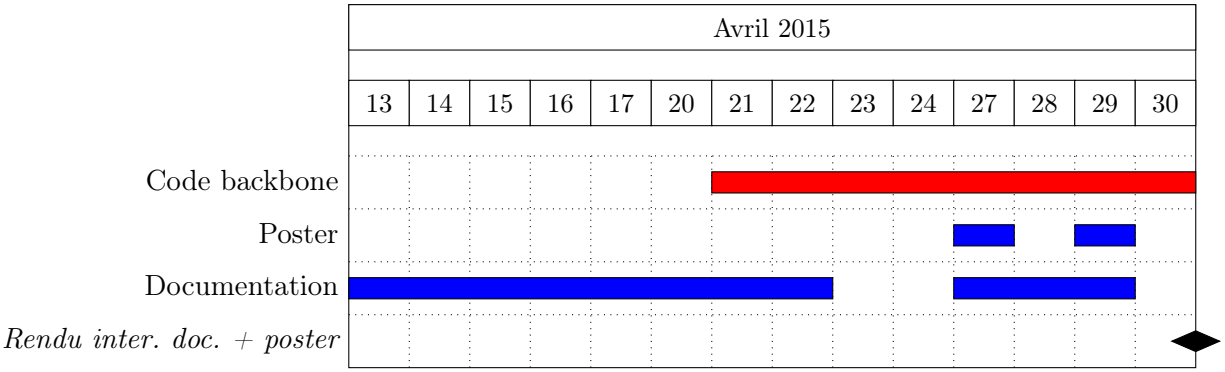



FIGURE 7.4 – Planning final du mois d'avril

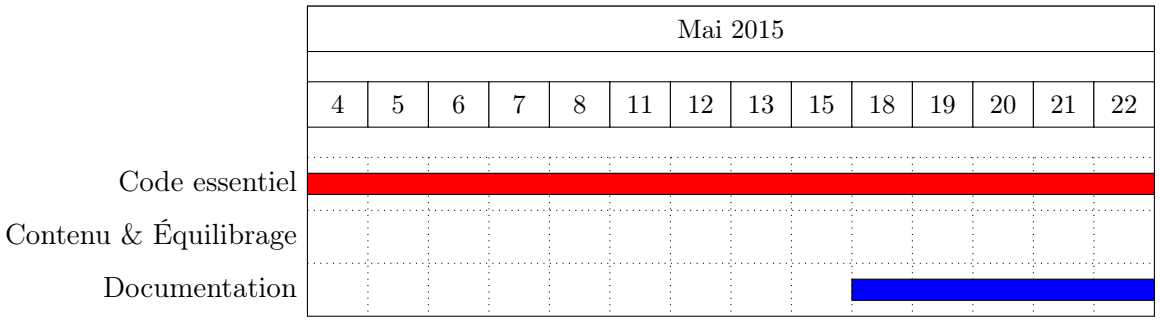



FIGURE 7.5 – Planning final de la première partie du mois de mai

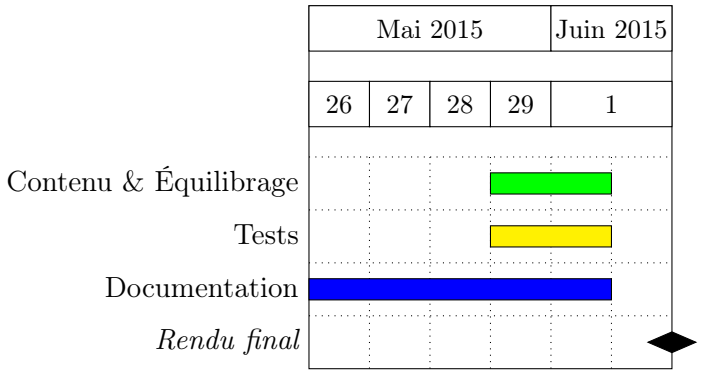



FIGURE 7.6 – Planning final de la deuxième partie du mois de mai et le mois de juin

## 7.2 Sources


— Automate cellulaire, *Wikipédia*, `http://en.wikipedia.org/wiki/Cellular_automaton`, Mai 2015

— Mouvements vectoriels, *Stack Exchange*, `http://gamedev.stackexchange.com/questions/13326/how-do-i-generate-projectiles-toward-the-mouse-pointer`, Mai 2015

— Tutoriels Monogame, *RB Whitaker's Wiki*, `http://rbwhitaker.wikidot.com/monogame-tutorials`, Mai 2015

— Tutoriels Monogame, *Coding Made Easy*, `http://www.codingmadeeasy.ca/`, Mai 2015

— Projet GitHub de Hel, *Yannick R. Brodard*, `https://github.com/brodardy/Hel`, Juin 2015

# Table des figures

# Liste des tableaux

# Listings