# Projet Tuteuré


**Équipe :**
Max Ducoudré
Mattis Desvilles
Abdel Ben Rejeb
Roni Basak

**Tuteur :**
Florent Madelaine

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL

IUT
SENART
FONTAINEBLEAU

Dépôt GIT : dwarves.iut-fbleau.fr/gitiut/ducoudre/PT2021_DeckBuilder

# Sommaire

# 1. Présentation de God's Raid

## 1-1. Explications du projet

God's Raid est un projet de Jeu Vidéo réalisée dans le cadre de la deuxième année d'IUT Informatique à Fontainebleau.

Ce jeu combine deux genres du domaine :

-Le **rogue-like.** Cela signifie que le jeu est grandement rejouable. Les parties ont vocation à être courtes, mais une partie influencera les parties suivantes. Plus le joueur fera de parties, plus le joueur aura accumulé des bonus et des connaissances pour les suivantes.

-Le **deck-building**. Tout au long d'une partie, le joueur va pouvoir accumuler et choisir des cartes à jouer. Plus la partie avancera et plus son deck (ensemble de cartes) sera grand et efficace pour contrer les obstacles. L'objectif pour le joueur est d'avoir le deck le plus optimisé pour finir la partie.

En plus de cela, le projet comprend aussi la réalisation d'un Site Internet. Ce site permet à la fois de télécharger le jeu, mais aussi de récupérer des informations sur son fonctionnement et sur son style.

## 1.2. Concepts & Objectifs

### 1-2-1. Le Jeu

*A) Objectifs à réaliser*

Ci-dessous le cahier des charges représentant la liste des concepts et objectifs devant être intégré au produit final.

**Le menu du jeu :**

| | |
|---|---|
| **Options** | Pouvoir changer des options comme la **difficulté** (Moins de récompenses à la fin de la partie) |
| **Lancer une partie** | Pouvoir lancer sa partie |
| **Sélectionner héros** | Pouvoir **sélectionner le héros** que le joueur utilisera pendant la partie. (cf : Héros) |
| **Multijoueur** | Pouvoir lancer une partie à **plusieurs joueurs** en **hébergeant un serveur.** |
| **Historique des parties** | Cet historique permet au joueur de **voir sa progression**. (Car le jeu est basé sur sa **rejouabilité**) |
| **Tutoriel** | Pouvoir afficher les **explications** du fonctionnement du jeu. |

**Héros :**

Le concept de héros permet au joueur d'avoir un personnage qui évolue au fur et à mesure des parties.

| | |
|---|---|
| **Points de vie** | Chaque héros a **son nombre de point de vie maximum.** Ces points de vie ne se régénèrent automatiquement durant la partie.<br>S'ils tombent à zéro, la partie est terminée. |
| **Mana** | Chaque héros a un **nombre de mana fixe au début d'une partie.**<br>Le mana **servira en combat pour utiliser les cartes.** |
| **Inventaire** | Un héros peut commencer avec des objets sur lui dès le début de la partie. |
| **Liste de carte** | Chaque héros peut **choisir entre 3 sets de cartes directement misent dans le deck avant chaque partie.** |
| **Passifs / buffs** | Chaque héros a des **effets différents tous le long de la partie** |
| **Arbre de talent** | A la **fin de chaque partie, le joueur obtient des points de talents**.<br>Ces points peuvent être **utilisé dans un arbre de talent pour débloquer** des talents.<br>Ces talents ont **des effets passifs** durant la partie.<br>**Une fois dans la partie**, à chaque niveau gagné, le joueur **peut activer un des talents débloqués**. |

### La partie sur la carte principale :

Possibilités une fois la partie lancée (hors combats)

| | |
|---|---|
| **Consulter l'inventaire** | Pouvoir voir le personnage et ses passifs. Consulter ses **objets activables** et son or. |
| **Inventaire de cartes (deck)** | Au début de la partie, le joueur a très peu de carte, il en **cumulera au fur et à mesure** de son avancement dans la partie. Cet inventaire de carte sera **limité en place**, il devra donc optimiser le choix des cartes de son deck. |
| **Carte du jeu** | La carte est **vue du dessus**.<br>Il y a un **point de départ** et un **point d'arrivé.**<br>Ces deux points sont **séparés par d'autres points** reliés par des **chemins**.<br>Le **joueur est sur un seul point.**<br>Le joueur va pouvoir **décider de la suite de chemin qu'il prendra pour accéder au point final** car il y a plusieurs chemins possibles qui seront **générés procéduralement**. |
| **Les zones (sur la carte)** | Une fois arrivé sur une zone, soit une **boutique** s'ouvre, soit un **évènement narratif** se déclenche, soit **un combat** se lance |
| **Évènement narratif** | Arrivé sur un point narratif : le joueur aura un texte et devra **prendre une décision**. En fonction de sa décision, il obtiendra un passif positif ou négatif, un objet, des cartes, rien du tout… Ex :<br>Un coffre est devant vous.<br>Choix : L'ouvrir \| Passer son chemin |
| **Boutique** | Arrivé dans une boutique, le joueur va pouvoir **dépenser son or** en **achetant des objets ou des cartes.** |

### Les combats :

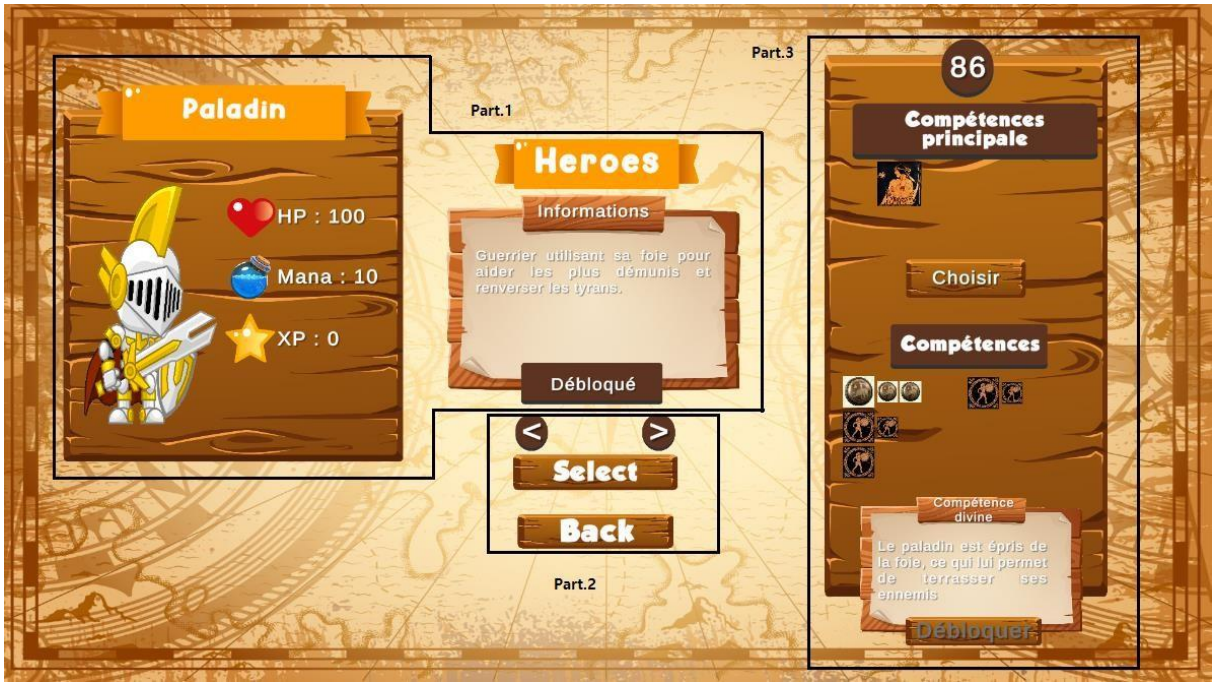
Les combats se déroulent une fois que le joueur s'est déplacé sur un point de combat ou à prit une mauvaise décision sur un point narratif.

| | |
|---|---|
| **Le joueur** | Les **points de vie du joueur ne se régénèrent pas** automatiquement entre les combats.<br>**S'ils tombent à zéro, la partie est terminée.** |
| **Ennemi(s)** | Le joueur vs utiliser ses cartes pour faire tomber les points de vie des ennemis à 0.<br>Les ennemis **n'ont pas de cartes**, ils attaquent directement le joueur.<br>**Leurs attaques et leur cycle d'attaques changera en fonction de qui ils sont et de leur nombre de points de vie.** |
| **La main** | Représente **les cartes** que le joueur **peut jouer pendant son tour** dans un combat. |

| | |
|---|---|
| **Le deck** | Représente les cartes que le **joueur n'a pas en main** pendant un combat. |
| **Les effets de la zone** | Le champ de bataille peut **influencer la partie en appliquant des bonus ou des malus** au participants. |
| **L'influence des objets** | Le joueur peut activer à tout moment des objets activables pendant un combat.<br>Le passif des objets est de mise durant les combats. |
| **Le déroulement des tours** | **Au début de la partie, le joueur pioche x** carte de son deck.<br>Il peut utiliser les cartes de sa main pendant son tour.<br>Au **début de chaque tour, il pioche** une carte de puis le deck. |
| **Le mana** | Nombre de **mana fixe** permettant **d'utiliser ses cartes** durant le combat.<br>Il représente **le coût des cartes**.<br>Il se **régénère au maximum a chaque début de tour.** |

### *B) Déroulement d'une partie*

Avant une partie, le joueur se retrouve sur le menu principal et plusieurs choix s'offre à lui. Il peut soit régler les options - et plus précisément la difficulté des parties qu'il va jouer -, il peut choisir de quitter le jeu ou bien démarrer une partie.




*Menu principal du jeu juste avant le début d'une partie*

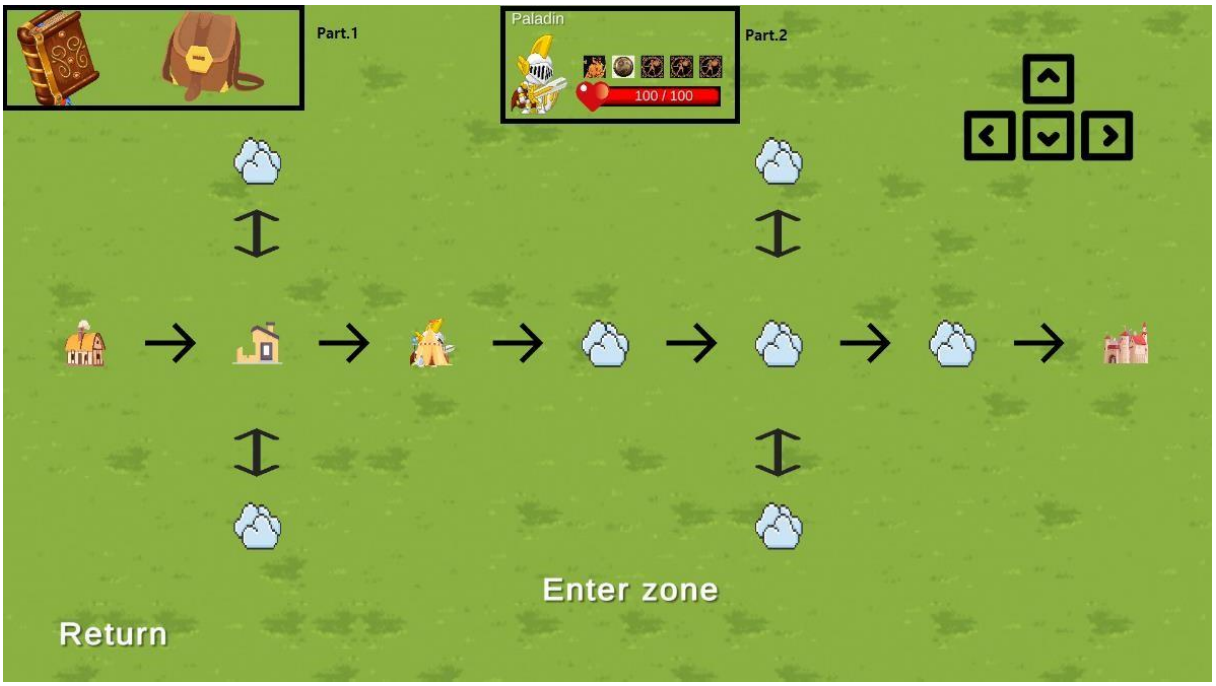
Une fois que le joueur a décidé de lancer une partie, le menu sur l'image ci-dessus s'ouvre. Ce menu permet au joueur de gérer et sélectionner ses **héros**, personnages ayant leurs propres spécificités et caractéristiques qui vont l'accompagner durant toute une partie.

Ce menu se divise en trois parties :
-Part.1 : Représente les informations à propos du héros actuellement sélectionnées. Informations différentes pour chaque héros et importantes pour que le joueur décide quel héros va-t-il prendre

-Part.2 : Cette partie de l'écran permet de changer le héros actuellement sélectionner avec les flèches puis de lancer une partie en cliquant sur « Sélectionner ».

-Part.3 : Cette partie montre elle aussi des informations sur le héros, et plus précisément : ses **compétences**. Les compétences peuvent être débloquées avec des **points d'expérience.** Quand une compétence est débloquée, elle l'est à vie et elle sera appliquée tout au long des parties jouées avec le héros sélectionné. Pour voir l'effet qu'elle produit, il suffit de cliquer sur l'une d'elles et de lire l'encadré blanc.

La partie se commence donc quand le héros est choisi, et c'est à ce moment que le menu ci-dessous s'affiche.




*La carte principale d'une partie*

Ce menu comporte de nombreuses informations et est le menu sur lequel le jouer va passer une grande partie de son temps. En effet, il représente une **carte**, le point de départ étant le point le plus à gauche et l'arrivée étant le point le plus à droite. Le joueur peut donc se déplacer avec les flèches directionnelles d'une **zone** (point sur la carte) à l'autre. La disposition et le nombre de zones sont générées aléatoirement en fonction de la difficulté du jeu.

Pour arriver à la zone de fin, le joueur va donc devoir choisir son chemin et traverser toutes les zones entre la zone de fin et d'arrivée.

Quand le joueur entre dans une zone, trois différentes choses peuvent se passer :
-Un combat peut se lancer ;
-Un magasin peut s'ouvrir ;
-Un évènement aléatoire peut se déclencher.

Avant d'expliquer en détail ce que signifient ces trois types de zones, le joueur peut voir accéder à plusieurs informations sur ce menu qui va lui permettre de prendre une décision sur les zones à traverser et/ou le chemin à emprunter.

Dans la **« Part.1 »** de l'image, on aperçoit deux icônes :
-la première représente le **deck** du joueur (ensemble de cartes à jouer). Ce deck contient la liste des cartes que le joueur pourra utiliser lors d'un combat ;
-la seconde icône représente l'**inventaire** du joueur, cet inventaire contient des objets qu'il pourra vendre, utiliser ou jeter quand il le souhaite durant la partie. Ces objets ont tous un effet différent sur la partie en cours.

Dans la « **Part.2** » de l'image, on peut connaître des informations à propos du joueur : son état de santé, son armure et ses **buffs** (effets appliqués sur le temps au joueur).

L'image ci-dessous représente un **combat** s'étant déclenché lorsque le joueur est entré dans une zone de combat.


*Illustration d'un combat en cours*

Un combat fonctionne de manière « tour par tour » : lorsque le joueur a fini de jouer, les ennemies vont à leur tour jouer, une fois que les ennemies ont joué, le tour revient au joueur qui pourra rejouer à son tour.

Au début du tour d'un joueur, il pioche une carte de son **deck** qui atterrit dans sa **main** (ensemble de cartes pouvant être joué). Ensuite, durant son tour, le joueur peut décider de deux choses :
-soit de jouer une des cartes de sa main (« Part.3 ») contre du **mana** (« Part.1 »), une ressource qui se régénère à chaque début de tour. Le coût en mana de chaque carte est inscrit dessus ;
-soit d'utiliser un objet depuis son inventaire.

L'objectif du joueur est de faire tomber les points de vie de ses ennemies (« Part.4 ») à zéro.

À la fin de son tour, lorsque le joueur clique sur le bouton de fin de tour, les ennemies peuvent commencer à jouer. Le comportement des ennemies change d'un ennemi à l'autre, mais généralement, ils font baisser les points de vie du joueur (« Part.2 »). Lorsque les points de vie du joueur tombent à zéro. La partie se finit.

Si le joueur gagne le combat, il récupère des **pièces** et de l'**expérience,** ces deux récompenses varient en fonction de la difficulté choisie et des ennemis vaincus.

L'objectif du joueur, durant un combat, est donc de vaincre les ennemies en perdant le moins de points de vie et d'objets pour que les combats suivants soient moins risqués.

Si le joueur rentre dans une zone de magasin, alors le menu sur l'image ci-dessous s'ouvre


**Illustration d'un magasin**

Cette zone est un magasin, elle permet au joueur d'acheter et de vendre ses cartes et ses objets. Lorsque le joueur ouvre son **inventaire** ou son **deck,** il peut décider de vendre, ce qui va augmenter son nombre de pièces (visibles depuis l'inventaire).

Avec les pièces, le joueur peut acheter des cartes pour compléter son deck et avoir de meilleures chances durant les combats en cliquant sur la carte désirée. Il peut aussi acheter des objets si son inventaire n'est pas plein en cliquant sur l'objet désiré.

Les objets et cartes disponibles à l'achat sont générés aléatoirement, il est donc possible pour le joueur de cliquer sur le bouton « Reset ». Ce bouton va re-générer les objets et cartes du magasin contre un certain nombre de pièces.

Les prix des objets et des cartes sont par ailleurs ajustés en fonction de la difficulté de la partie.

Une partie se termine lorsque le joueur a gagné le dernier combat de la partie ou que ses points de vies tombent à zéro. Lorsque ça arrive, l'expérience accumulée lors des combats est sauvegardée. Cette expérience pourra être utilisée par le joueur pour débloquer de nouvelles compétences du héros.

### 1-2-2. Le Site Internet

#### *A) Objectifs à réaliser sur le Site Internet*

| | |
|---|---|
| **Accueil** | Entrée du site internet |
| **Création de compte** | Pouvoir créer son compte |
| **Modification du compte** | Pouvoir modifier son compte, comme **ajouter un avatar**, **regarder ses statistiques** |
| **Classement des joueurs** | Récupérer les meilleurs temps des joueurs pour faire un **classement mondial** |
| **Télécharger le jeu** | Un bouton pour télécharger le jeu une fois le compte créé |
| **Encyclopédie des cartes** | Avoir un recueil de toutes les cartes, **classées et rangées** par catégorie sur le site à partir d'une base de données répertoriant toutes les cartes. |

#### *B) Présentation du site*

Avoir un site internet est important pour que le jeu ne soit pas simplement un programme utilisable et connu simplement par ceux qui l'ont créée. Le site permet d'avoir à n'importe qui de voir un genre de « fiche de présentation » du jeu, c'est-à-dire pouvoir récupérer des informations à son propos, savoir ce qu'il est et, bien sûr, l'installer pour l'essayer soi-même.

Ce site permet simplement de gérer l'aspect **distribution** du jeu.

La navigation sur le site internet est divisée en 5 catégories

Accueil :
C'est le point d'entrée du site, il est possible de lire les nouveautés et les mises à jour du jeu.

About :
Cette catégorie permet de répondre à quelques questions qui pourraient venir de la part d'un utilisateur voulant en apprendre plus sur le jeu.

Guide :
La catégorie « Guide » contient un tutoriel sur le fonctionnement du jeu pour qu'un utilisateur ne soit pas perdu lorsqu'il veut commencer une partie.

FeedBack :
Cette catégorie permet à un utilisateur de donner son avis ou sur le jeu ou de rapporter des bugs qu'il aurait rencontrés à l'équipe de développement.

Téléchargement :
Cette catégorie est la plus importante car elle permet à un utilisateur d'accéder au seul moyen de téléchargement du jeu s'il désire y jouer. Elle contient donc un bouton pour installer le jeu.

## 1-3. Répartition du travail & Organisation interne

L'équipe est composée de 4 personnes et la répartition fût la suivante :

Max Ducoudré :
-Conception des diagrammes de classes UML
-Programmation du fonctionnement des objets, des cartes à jouer et des combats

Mattis Desvilles :
-Codage du jeu
-Programmation du fonctionnement de la carte, des héros et du menu principal

Abdel Ben Rejeb :
-Codage des éléments générique (Les éléments spécifiques comme les cartes, les ennemies, les héros, les évènements aléatoires…) -Game & Level Design

Roni Basak :
-Développement du site Internet
-Game & Level Design

**Remarque :**
-Roni Basak a intégré le projet en cours de réalisation et s'est très bien adapté à l'équipe -Il n'étais pas exclu qu'un membre de l'équipe aide les autres sur un domaine même si ce n'est pas dans sa répartition de travail.

# 2. Aspect technique

## 2-1. Organisation du projet & technologies utilisées

### 2-1-1. Le Jeu

Pour la réalisation du programme, du jeu, nous avons utilisé trois choses différentes :

-Le logiciel **Unity :**
Ce logiciel est réputé pour être un outil de création de jeux vidéo. Il permet de gérer la compilation du code et de gérer l'aspect graphique (3D ou 2D) d'un jeu. Pour gérer cet aspect graphique, Unity met à disposition une documentation complète sur son fonctionnement.

-Le **langage C#** :

Ce langage de programmation est le seul compatible avec le logiciel Unity. Il est orienté objet et simple à prendre en main, surtout si on connaît déjà le langage Java.

-L'**XML**
Ce langage a permis pour le projet de stocker les données de tous les objets qui vont être instanciés par la suite en C#. Stocker les données de cette manière nous permet facilement de changer des attributs fixes des objets pour faire de l'équilibrage.

### 2-1-2. Site Internet

Pour réaliser le site internet, les langages HTML5, CSS et Javascript ont été utilisés.

Le CSS pour y appliquer des feuilles de style.

L'HTML5 pour stocker les informations nécessaires au site internet

Le Javascript pour gérer la récursivité du site internet.

L'objectif initial était d'utiliser un framework PHP comme CodeIgniter pour gérer la base de données des cartes du jeu, malheureusement, par manque de temps, cela ne s'est pas fait.

Le site internet est hébergé depuis le site de l'IUT informatique de Fontainebleau à l'adresse suivante : https://dwarves.iut-fbleau.fr/~ducoudre/godraid/

## 2-2. Fonctionnement du jeu

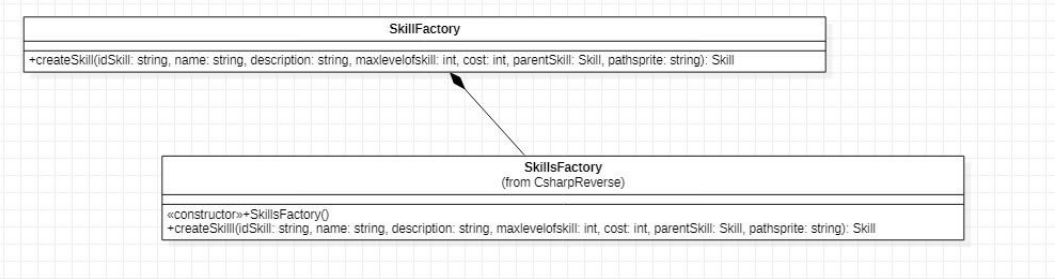
### 2-2-1. Modèle

#### *A) Fonctionnement de l'instanciation des objets*

Le jeu comporte beaucoup d'objets différents à instancier en grande quantité comme par exemple : les différentes cartes, les objets, les zones, les compétences, les héros… Tous ces objets sont uniques et une classe, à chaque fois différente, doit y être rattachée pour qu'un script différent se déclenche entre plusieurs objets du même type.

Pour résoudre ce problème, nous avons utilisé les usines et des fichiers XML, voici l'exemple du fonctionnement des classes « Hero » et « Skill » :

Lorsque le joueur arrive dans l'écran de sélection, un objet du type HeroesFactory s'instancie.




*Classe de l'usine des compétences*

Cet objet permet de lire dans un fichier XML et d'instancier tous les héros et skills qui y sont décrits comme sur l'image ci-dessous.

```
114
115  <Heroe
116          id="archer"
117          nom="Archer"
118          desc="Habile de ses doigts, prêt à utiliser son arc en toutes circonstances"
119          maxHP="100"
120          maxMana="10"
121          sprite="archer_sprite"
122          starterItems="apple,apple,health_potion,health_potion"
123          >
124          <skills
125              numberOfSkill="1">
126              <skill
127                  id="divine_skill"
128                  name="Compétence divine"
129                  desc="Le paladin est épris de la foie, ce qui lui permet de terrasser ses ennemis"
130                  isPassive="true"
131                  maxLevel="1"
132                  cost="3"
133                  isPrimarySkill="true"
134                  sprite="athena_illustration"
135                  starterCards="fireball,fireball,plague,plague,boar,boar"
136                  >
137              </skill>
138          </skills>
139      </Heroe>
140  </Heroes>
```

Ce fichier heroes.xml contient donc tous les attributs d'un objet du type Hero et il est tout à fait possible d'ajouter ou de supprimer des balises <Heroe> dans ce fichier à la condition qu'il respecte la forme décrite ci-dessus.

Chaque « Skill » du héro ne peuvent être simplement décrit par des attributs fixes. Un skill est censé appliquer un effet au héros durant la partie. C'est pour cela que « HeroesFactory » vas utiliser une « SkillFactory ». La méthode createSkill de la « SkillFactory » va utiliser l'ID du skill pour l'instancier correctement.

Un skill fonctionne de la manière suivante : il reste attaché au joueur toute la partie, puis, au début d'un combat, il applique un « Buff » (Effet sur la durée) sur une « Target »(un être vivant sur le champ de bataille). C'est pour cela que l'objet « Skill » contient une méthode abstraite « applyBuff » renvoyant un buff qui s'appliquera sur une cible choisie au début d'un combat.

**Exemple de création d'une compétence:**
Il faut créer un skill avec l'id « dangerous_heart » à appliquer au héro avec l'id « archer ».

On commence par ajouter une balise <skill> dans la balise <skills> elle-même dans la balise <heroe> ou l'attribut « id=archer » dans le fichier heroes.xml

```
<skills
    numberOfSkill="1">
    <skill
        id="dangerous_heart"
        name="Blessure dangeureuse"
        desc="Au début d'un combat, un ennemi obtient poison, il subit 2 de dégâts par tour pendant 2 tours"
        isPassive="true"
        maxLevel="2"
        cost="1"
        isPrimarySkill="false"
        sprite="athena_sword">
    </skill>
</skills>
```

On écrit ensuite un script de cette manière, respectant la description mise en attributs XML

```
public class DangerousHeart : PassiveSkill
{
    public DangerousHeart(string idSkill, string name, string description, int maxlevelofskill, int cost, Skill parentSkill, string pathsprite)
        : base(idSkill, name, description, maxlevelofskill, cost, parentSkill, pathsprite)
    {
    }
    public override void applyBuff()
    {
        int tours = 0;
        int damage = 2;

        if (this.getLevelOfSkill() == 1)
        {
            tours = 2;
        }

        if (this.getLevelOfSkill() == 2)
        {
            tours = 3;
        }

        Target target = GlobalVariables.currentFight.getRandomAliveEnnemy();
        target.addBuff(new BleedingBuff(this, tours, target, damage));
    }
}
```

On relie ensuite l'ID du skill au script en rajoutant une condition dans la méthode de la SkillFactory

```
public class SkillsFactory
{
    public SkillsFactory()
    {
    }
    public PassiveSkill createPassiveSkill(string idSkill, string name, string description, int maxlevelofskill, int cost, Skill parentSkill, string pathsprite)
    {
        if (string.Equals(idSkill, "dangerous_heart"))
        {
            return new DangerousHeart(idSkill, name, description, maxlevelofskill, cost, parentSkill, pathsprite);
        }
```

A noter que ce système est réutilisé de la même manière pour les objets de type « Item », de type « Card » et de type « Ennemy ». La seule différence étant que les usines sont appelées à des moments différents.

### *B) Les Variables Globales*

Certains objets du programme sont appelés à être utilisé un peu partout dans de nombreux contextes différents. C'est pour cela qu'il existe une classe statique nommée GlobalVariables avec de nombreux attributs.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public static class GlobalVariables
{
    public static Party currentParty; // use only during party (after hero selecti
    public static Fight currentFight; // use only during fight (else null)

    // Factories
    public static FightZoneFactory fightZoneFactory;
    public static ShopZoneFactory shopZoneFactory;
    public static MapFactory mapFactory;
    public static ItemFactory itemFactory;
    public static TypeFactory typeFactory;
    public static CardFactory cardFactory;
    public static EnnemiesFactory ennemiesFactory;
    public static SkillsFactory skillsFactory;
    public static ImprovmentSkillsFactory improvmentSkillsFactory;

    public static DeckController deckController;
    public static InventoryController inventoryController;

}
```

*Code de la classe GlobalVariables*

Toutes ces variables sont instanciées quand une partie se lance à l'exception de la variable « currentFight » qui a pour valeur « null » lorsque le joueur n'est pas en combat.

Cette classe est beaucoup utilisée dans le code est cartes, des buffs, des objets et zones d'évènements aléatoire.
**Par exemple**, depuis une le code d'une carte, je peux récupérer la variable currentFight et ensuite aller y trouver la liste des ennemies présent sur le champ de bataille pour leur infliger à tous des dégâts.

### *C) Diagramme de classe du modèle*

Le diagramme de clase complet est disponible dans les annexes

#### La classe « Party » :

Cette classe est la classe centrale, c'est elle qui permet de récupérer toutes les informations sur la partie en cours.

Elle s'instancie quand le joueur a fini de sélectionner son héros et permet en suite d'accéder aux données de la carte et du joueur.

#### La classe « Player » :

Cette classe représente le joueur présent sur la carte et permet d'accéder à son inventaire et son deck qui représente respectivement la liste de cartes et d'objets que possède le joueur.

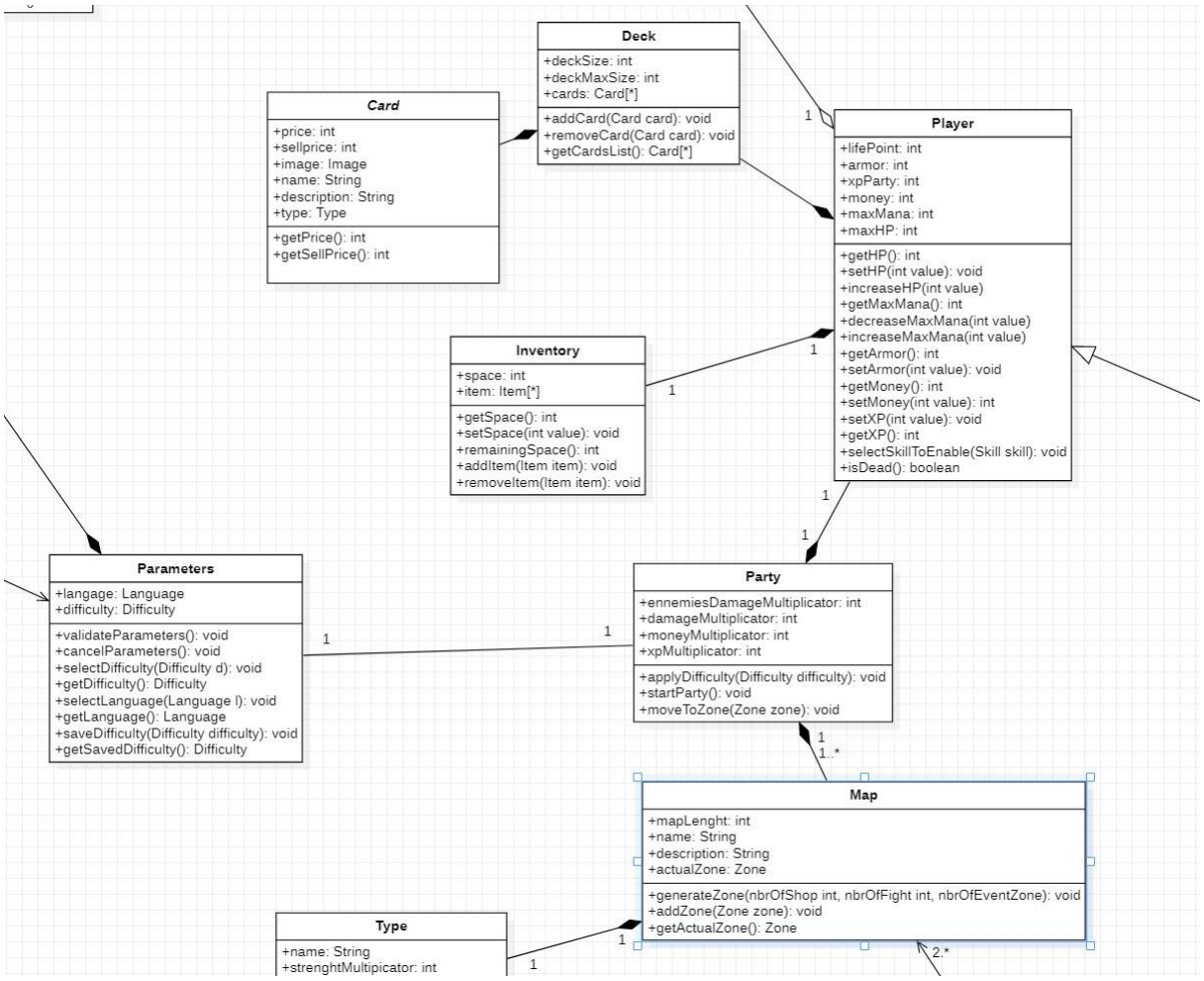



*Illustration des classes Party et Player*

### La classe « Fight » :

Cette classe représente toutes les informations à propos d'un combat. Un objet du type Fight s'instancie quand le joueur entre dans une zone de combat et cet objet devient accessible depuis la variable « currentFight » de la classe statique « GlobalVariables ».

Elle contient notamment un objet de type PlayerFight qui hérite de Player. Cet objet est copie les attributs de l'objet Player et en gagne quelques-uns qui sont propres au combat en cours (comme le mana restant, les cartes piochées…). À la fin d'un combat, la méthode « void giveInfoToPlayer() » s'exécute : elle permet de donner à l'objet du type Player les informations qui ont changés durant le combat (comme les points de vies et l'expérience acquise).

### La classe « Target » :

Cette classe est la classe parente de tous les êtres vivants du projet (les compagnons du joueur, les ennemies et le joueur lui-même) et contient des attributs propres à eux tous comme l'armure, les points de vies…

Elle contient des méthodes qui permettent de gérer ces attributs. Comme la méthode « takeDamage() » qui inflige des dégâts à l'armure puis aux points de vie quand l'armure est à 0 ou bien des méthodes de Get et Set pour changer les attributs directement.

Un objet du type Target est majoritairement utilisé par les cartes qui ont une cible bien spécifique.

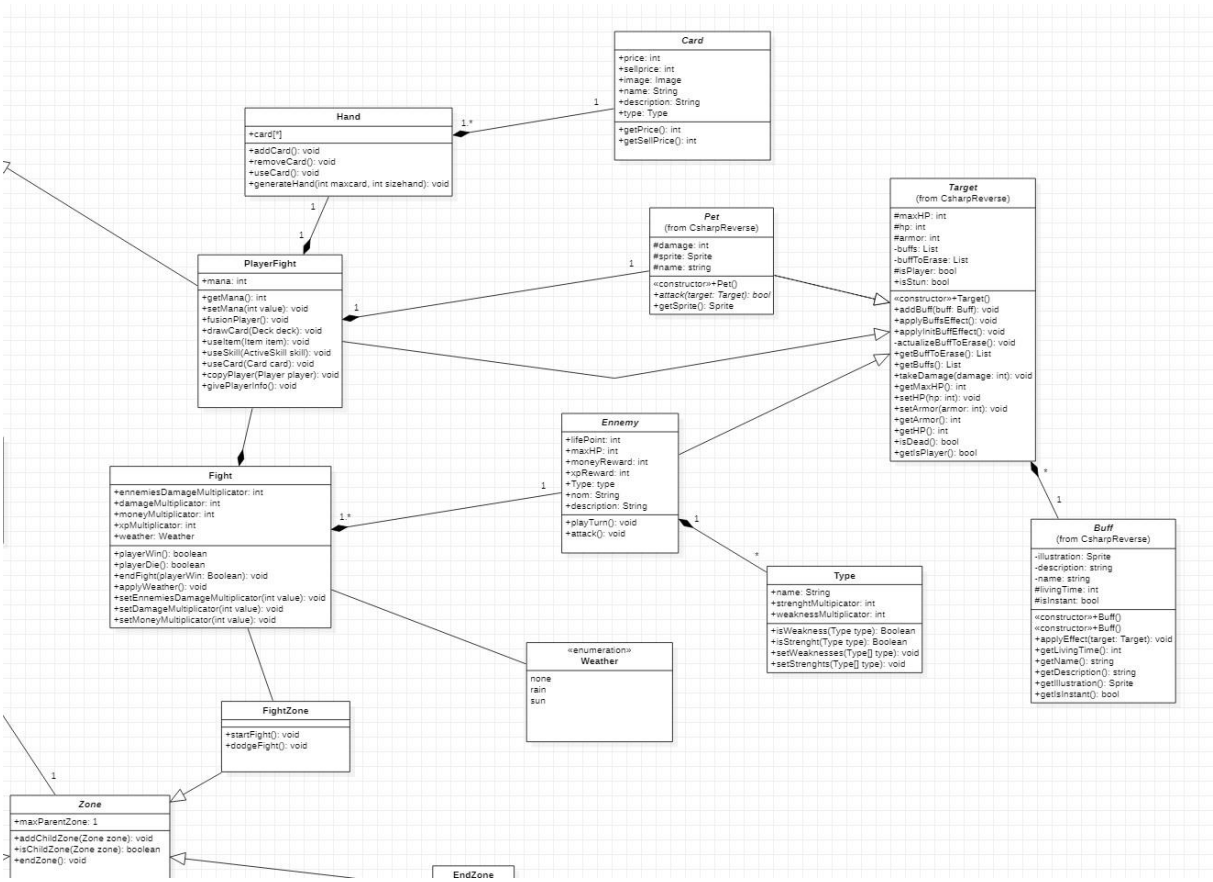



*Illustration des classes reliés au système de combat*

### *2-2-2. Fonctionnement des Vues et des Contrôleurs*

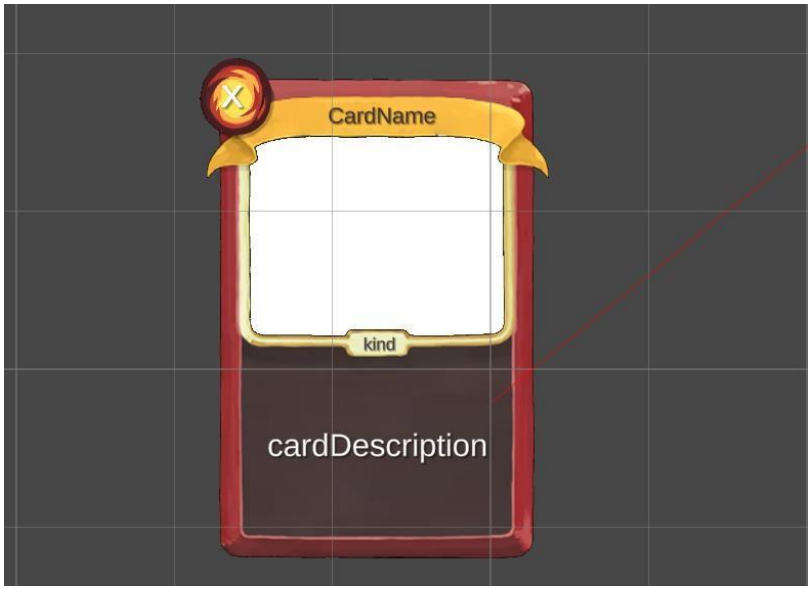
#### *A) Génération des vues*

Unity fonctionne de la manière suivante, un projet contient des scènes et une scène contient des GameObject. Un GameObject est une classe représentant quelque chose de présent dans l'environnement (Un bouton, une image, le point de vue du joueur …). Une scène contient de nombreux GameObject qui peuvent être imbriqués les uns dans les autres.

Avec Unity, il est possible de créer un GameObject et de lui ajouter des composants (tel que des scripts, une position dans l'espace, un texte…). Tous ces éléments sont modifiables via l'interface graphique, mais il est aussi possible de récupérer les GameObject depuis le code pour modifier ses composants.

Pour réaliser les vues sur Unity, nous avons procédé en deux étapes. Prenons l'exemple d'une barre d'une carte à jouer.

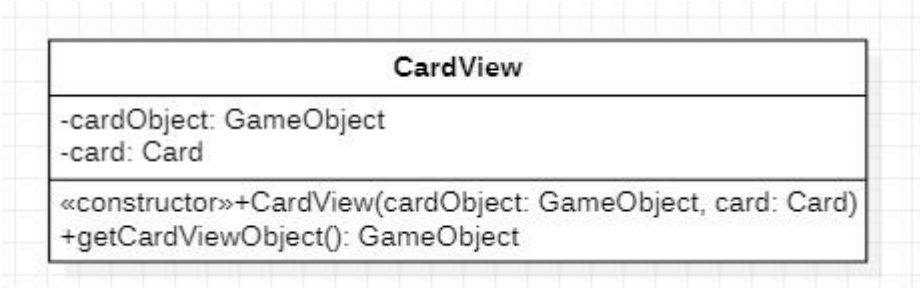
Tout d'abord, nous avons utilisé l'interface graphique de Unity pour créer un modèle de carte générique comme sur l'image ci-dessous en lui ajoutant des composants à la main et en lui donnant la forme qu'on souhaitait.




*Modèle générique de carte à jouer*

Comme on peut le voir, chaque élément de cette carte ont une valeur fixe (la description, le coût en mana, le nom, l'illustration…). Cette carte nous permet d'avoir une base pour créer toutes les autres.

Voici la classe CardView :




*La classe CardView permettant de gérer l'affichage d'une carte*

Cette classe prend en paramètre la classe Card - qui est le modèle contenant toutes les informations sur une carte – puis clone le GameObject de carte vide depuis la scène pour ensuite lui donner les valeurs présentes dans le modèle en argument. Nous récupérons donc le GameObject représentant une carte se basant sur un modèle instancié dans la mémoire.

Ce mécanisme a été utilisé notamment pour les barres de vies et les personnages présents sur une scène.

### *B) Gestion des contrôleurs*

Unity propose une classe abstraite se nommant MonoBehaviour : chaque classe héritant de cette classe abstraite peut être rattachée à un GameObject en tant de composant.

MonoBehaviour contient de nombreuses méthodes, mais nous n'avons utilisé que deux d'entre elles : la méthode « void Start() » et la méthode « void Update() ». Dans une classe héritant de MonoBehaviour, il est possible de réécrire ces méthodes en leur appliquant du code. Tout le code contenu dans « Update() » sera exécuté à chaque frame et tout le code contenu dans « Start() » sera exécuté une seule fois quand le GameObject qui y est rattaché à la classe est chargé.

Dans le projet, les classes de contrôleur héritent toutes de MonoBehaviour et sont rattaché au GameObject contenant tous les autres GameObject sur la scène.

Ces classes nous permettent donc d'exécuter le code des modèles et des vues pour avoir un programme dynamique.

Par exemple, pour les combats : Lorsqu'un joueur entre dans une zone de combat, le GameObject principal nommé « FightScene » se charge et exécute le code dans la méthode « Start() » contenu dans la classe « FightController » elle-même rattachée au GameObject « FightScene ».

Ensuite, dans la méthode « Update() » s'exécute à toutes les frames et permet l'actualisation de l'affichage et l'exécution du code du modèle en fonction de ce que le joueur fait sur l'écran.

Dans le projet, il existe donc un contrôleur principal par scène qui exécute tout le code qui y est rattaché. Ci-dessous une image montrant les classes contrôleurs dans le projet :

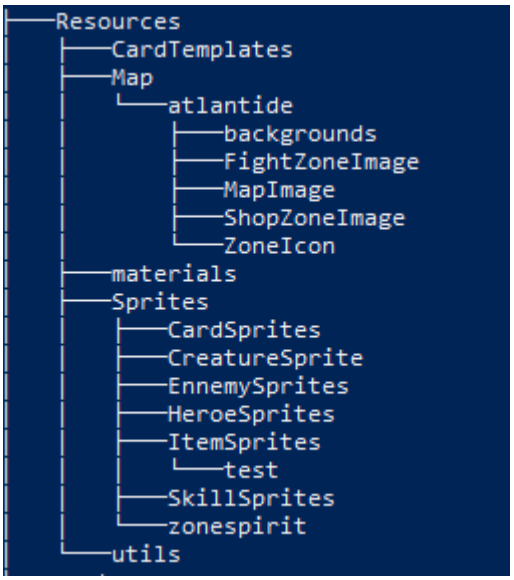
Les classes DeckController et InventoryController ne sont pas rattachés à une scène spécifique car chacune d'elles peuvent être présentes sur plusieurs scènes différentes.

*La liste des contrôleurs est disponible en annexe*

### *2-2-3. Arborescence du projet*

L'intégralité des fichiers qui ne sont pas générés automatiquement pas Unity sont placés dans un répertoire nommé assets. Il est subdivisé en plusieurs parties :

#### A) Répertoire Ressources




Le répertoire « Ressource » contient toutes les images et sprites 2D importés. Il contient 3 répertoires principaux :

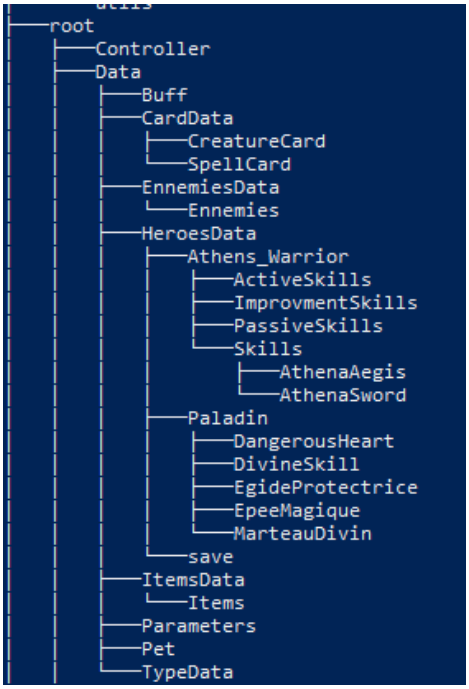
- « Sprites » : Regroue chaque sprites des objets dans le jeu (les items, les créatures, les personnages…) ;

-« Map » : Contient tous les sprites sur la carte de la campagne vue du dessus. Ce répertoire contient plusieurs répertoires nommés en fonction de l'ID de la carte sélectionnée :

-« utils » : Contient des images réutilisables un peu partout (comme les fonds, les boutons, les icones d'inventaire et de deck…).

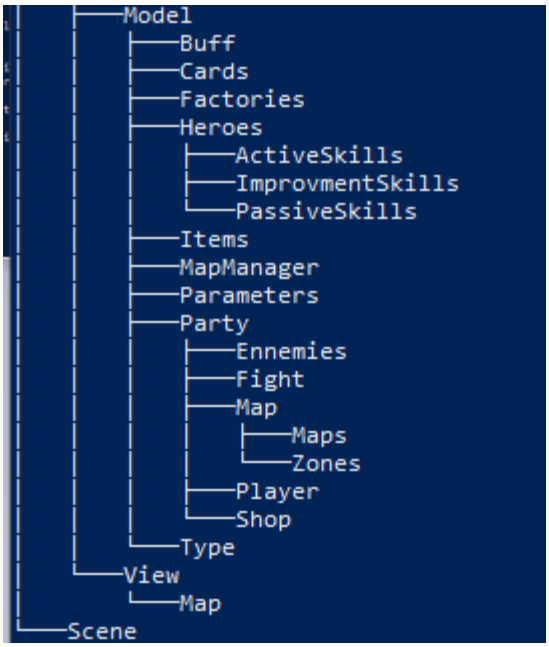
#### B) Répertoire Root

Ce répertoire contient tous les fichiers sources du projet




Le répertoire « Data » dans root contient toutes les données stockées en XML et toutes les classes génériques.

Le répertoire « Controller » contient toutes les classes hésitantes de « MonoBehaviour » rattachée chacune à un GameObject Unity.



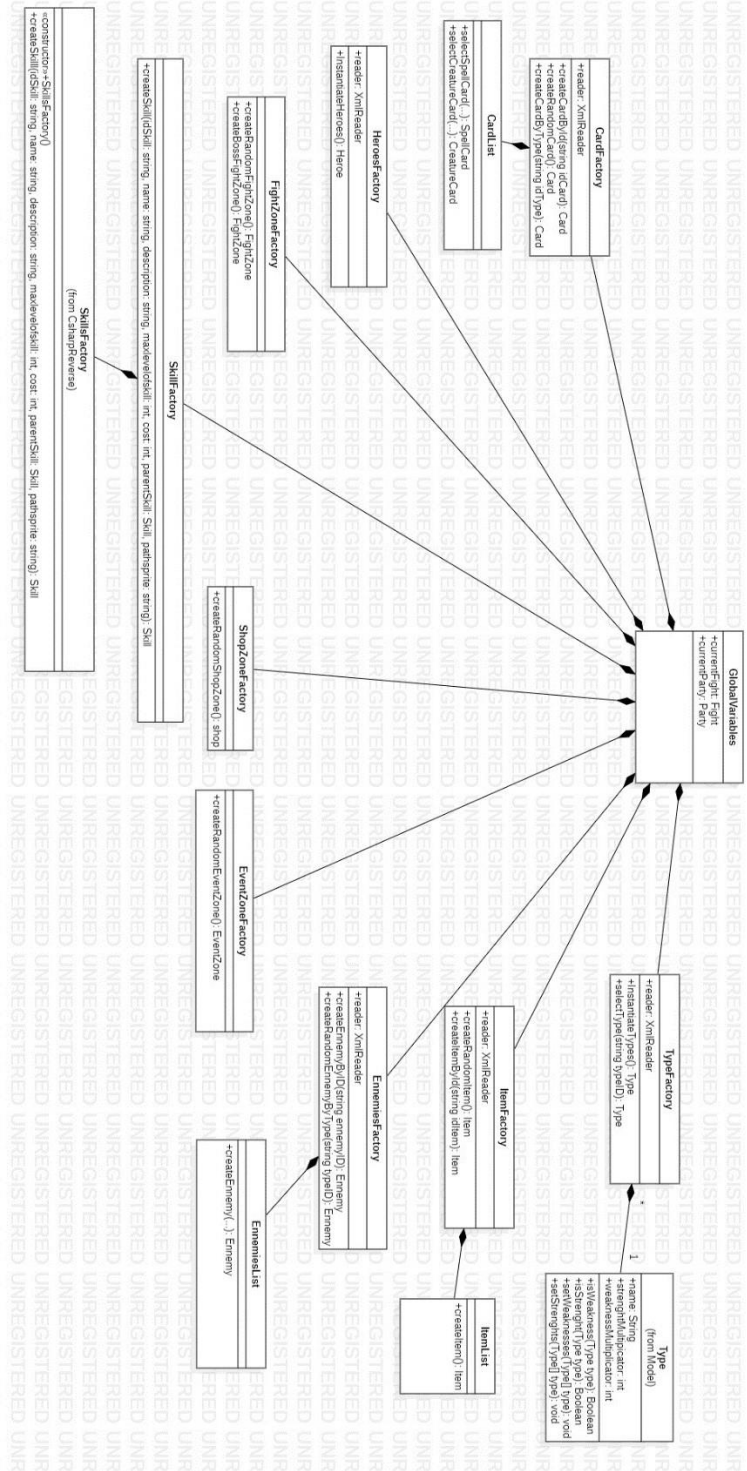

Le répertoire « Model » contient toutes les classes C# classiques. Elles permettent de stocker les données en cours d'une partie et de les faire communiquer. Les méthodes de ses classes sont donc appelées par les classes contrôleurs.

Le répertoire « Scene » contient tous les fichier « .scene » propre à Unity.

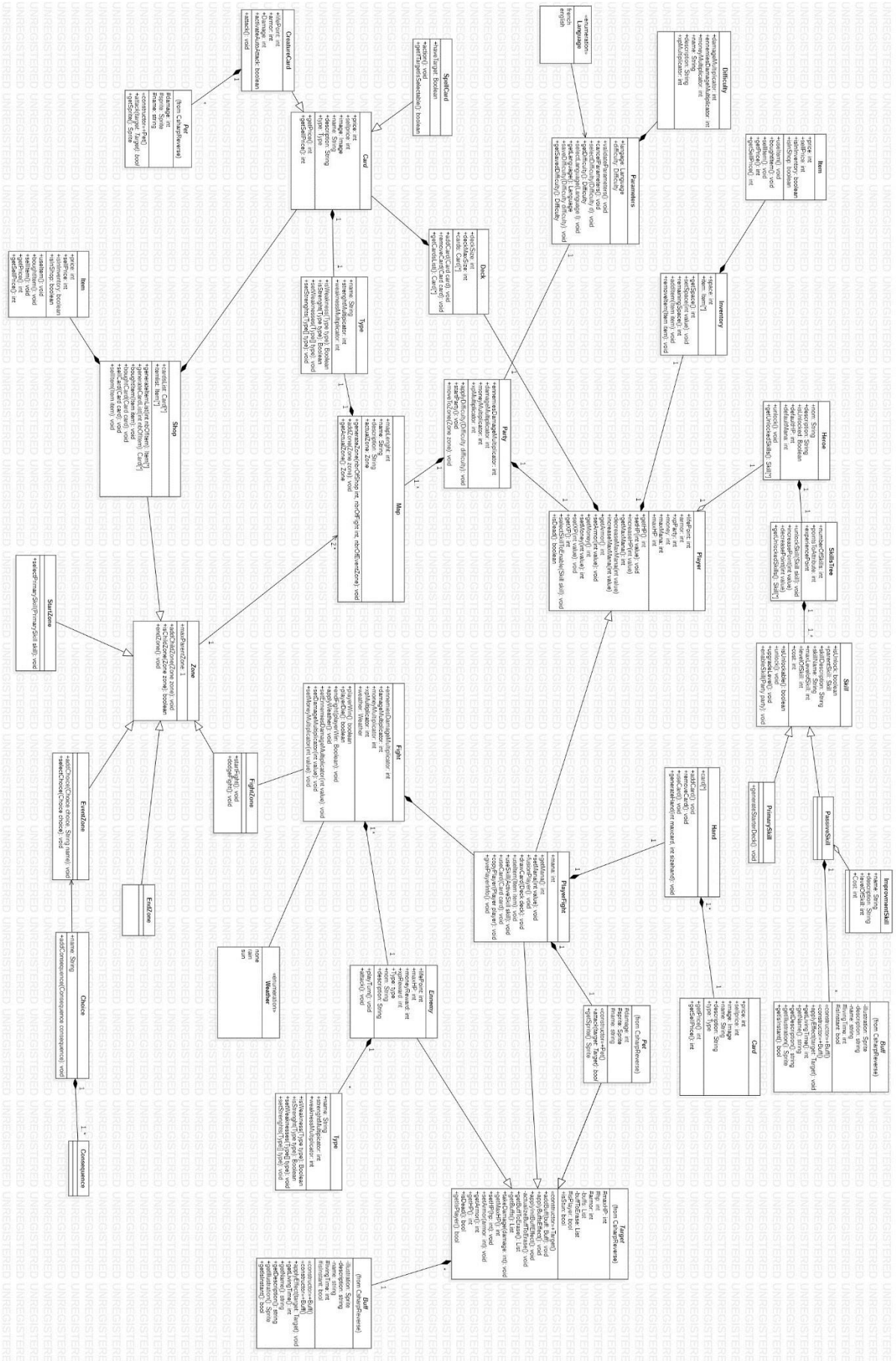
# 3.Annexes

## 3-2. Diagrammes

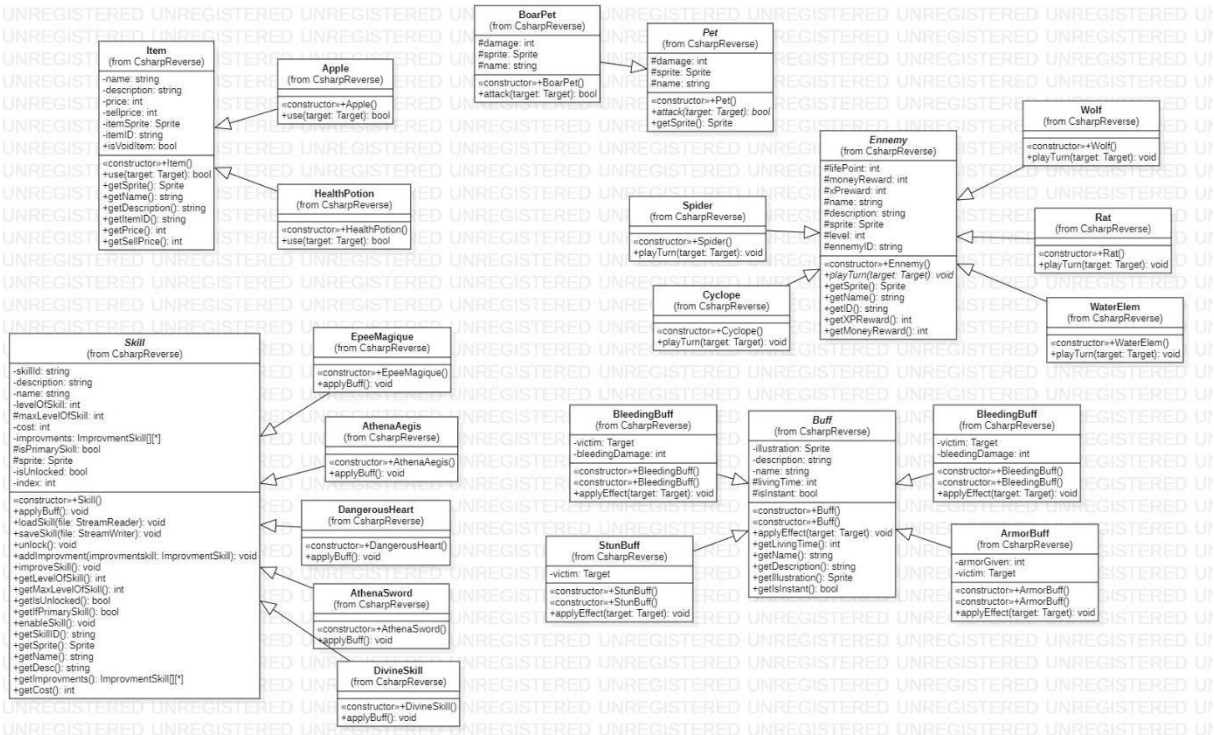
### 3-2-1. Usines



## 3-2-2 : Diagramme de classe modèle



## 3-2-3 : Diagramme de classe modèles génériques



## 3-2-4 Diagramme de classe des contrôleurs :