

# Étude de Unreal Engine 4

Travail de semestre

Yannick BRODARD

18 mars 2015



Enseignants :

- Christophe MARÉCHAL
- Stéphane GARCHERY

**CFPT - École d'informatique**  
Technicien ES 2<sup>ème</sup> année

## Première partie

# Avant-propos

### 1 Résumé

Le but de ce projet est d'étudier l'environnement de développement de jeux, Unreal Engine 4. Les outils, composants et manières de codés sont identifiés. Un jeu exemple est fourni avec l'étude. Ce jeu est une imitation du jeu Galaga. En utilisant les compétences acquises avec la recherche, le jeu est entièrement construit avec le savoir-faire. Le travail s'est conclu avec une recherche me satisfaisant et m'a fait découvrir le monde du développement de jeux conséquents, mais le jeu exemple n'est malheureusement pas terminé et requiert plus de développement.

### 2 Abstract

The objective of this project is to study the development environment for games, here Unreal Engine 4. The tools, components and ways to code are identified. An example of a game is given to the study. This game is an imitation of Galaga. By using the skills acquired with the research, the game is done with this knowledge. The work concludes with a satisfying research work and it allowed me to understand more the development of big games, but the game example wasn't finished and demands more development.

## Table des matières

<b>I</b>	<b>Avant-propos</b>	<b>2</b>
<b>1</b>	<b>Résumé</b>	<b>2</b>
<b>2</b>	<b>Abstract</b>	<b>2</b>
<b>II</b>	<b>Introduction</b>	<b>5</b>
<b>3</b>	<b>Définition du projet</b>	<b>5</b>
3.1	Unreal Engine 4 . . . . .	5
<b>4</b>	<b>Définition des objectifs</b>	<b>6</b>
<b>III</b>	<b>Analyse préliminaire</b>	<b>7</b>
<b>5</b>	<b>Analyse de l'existant</b>	<b>7</b>
<b>6</b>	<b>Environnement de travail</b>	<b>7</b>
6.1	Logiciel . . . . .	7
6.2	Matériel . . . . .	7
6.3	Conditions de travail . . . . .	7
6.3.1	Durée . . . . .	7
6.3.2	Délivrables . . . . .	7
6.3.3	Suivi . . . . .	7
<b>IV</b>	<b>Étude Unreal Engine 4</b>	<b>8</b>
<b>7</b>	<b>Unreal Editor</b>	<b>8</b>
7.1	Barre de contrôle . . . . .	8
7.2	Blueprints (Plans) . . . . .	9
7.3	Mesh (Maillage) . . . . .	10
7.4	Entrées des périphériques . . . . .	11
<b>8</b>	<b>Langage de programmation</b>	<b>12</b>
8.1	Les macros . . . . .	12
8.2	Préfixes des classes UE4 . . . . .	12
8.3	Classes . . . . .	12
8.3.1	Gamemode . . . . .	12
8.3.2	Actor . . . . .	12
8.3.3	Pawn . . . . .	12
8.3.4	Character . . . . .	13
<b>V</b>	<b>Analyse fonctionnelle - Jeu</b>	<b>14</b>
<b>9</b>	<b>Fonctionnalités</b>	<b>14</b>
9.1	Règles du jeu . . . . .	14
9.2	Fonctionnement . . . . .	14
9.2.1	Joueur . . . . .	14
9.2.2	Ennemis . . . . .	15

<b>10 Interface homme-machine</b>	<b>15</b>
10.1 Écran du jeu . . . . .	15
10.2 Menu du jeu . . . . .	16
 <b>VI Analyse Organique - Jeu</b>	 <b>17</b>
<b>11 Arborescence du projet</b>	<b>17</b>
<b>12 Mode de jeu</b>	<b>18</b>
<b>13 Personnage</b>	<b>19</b>
<b>14 Projectiles</b>	<b>22</b>
<b>15 Collisions</b>	<b>22</b>
 <b>VII Tests</b>	 <b>23</b>
 <b>VIII Conclusion</b>	 <b>24</b>
<b>16 Conclusion technique</b>	<b>24</b>
16.1 Bilan de l'étude . . . . .	24
16.2 Bilan du jeu exemple . . . . .	24
16.3 Améliorations . . . . .	24
<b>17 Conclusion personnelle</b>	<b>25</b>
17.1 Bilan personnel . . . . .	25
17.2 Apport personnel . . . . .	25
17.3 Gestion du temps . . . . .	25
 <b>IX Annexes</b>	 <b>26</b>
<b>18 Planning</b>	<b>26</b>
18.1 Initial . . . . .	26
18.2 Réel . . . . .	27
<b>19 Références bibliographiques</b>	<b>28</b>

## Deuxième partie

# Introduction

### 3 Définition du projet

Ce projet a été mis en place pour s'initier au développement de jeux-vidéos avec un moteur de jeu, étant donné que mon travail de diplôme sera de développer un jeu complet.

Pour ce faire, une étude et un travail pratique avec l'environnement Unreal Engine 4 a été proposé.

Dans cette étude, plusieurs aspects de ce moteur seront analysés et documentés, notamment :

- La gestion des modèles 3D
- La création de jeu
- L'interaction utilisateur-machine

Proposé par Monsieur Maréchal, l'intégration d'un périphérique 3D (le *Novint Falcon 3D Touch Controller*) a été prévu pour étudier son fonctionnement et son implémentation sur ce genre d'environnement.

#### 3.1 Unreal Engine 4

Unreal Engine est un moteur de jeu développé par Epic Games<sup>1</sup> qui a été présenté pour la première fois en 1998 avec un jeu FPS<sup>2</sup> nommé *Unreal*. Bien que le moteur était développé principalement pour des jeux FPS, il est aujourd'hui utilisé pour beaucoup d'autres genres de jeux comme :

- Infiltration
- Action
- MMO (Jeux massivement multijoueurs en-ligne)
- Jeux à la troisième personne
- Et bien d'autres...

La version actuelle de UE4<sup>3</sup> est conçu pour fonctionner avec Microsoft DirectX 10 à 12, OpenGL et JavaScript/WebGL.

---

1. <http://epicgames.com/>

2. First Person Shooter : Jeu de tir à la première personne

3. Unreal Engine 4

## 4 Définition des objectifs

Le but est d'étudier l'environnement de Unreal Engine 4 dans ses détails, c'est-à-dire la programmation avec UE4, la modélisation, l'édition de différents scénarios et prise en main de la physique du moteur. Un travail avec la souris *Novint Falcon 3D Touch Controller* est aussi demandé, cette souris 3D permet à l'utilisateur de reproduire des mouvements sur 3 axes qui peuvent être interprétés par l'ordinateur.



FIGURE 1 – Souris Novint Falcon 3D Touch Controller

Une étude comparative du moteur de jeu sera faite avec le moteur Unity 3D, cette étude est confiée à Monsieur Daniel Lopes.

Le mini-jeu est basé sur le jeu *Galaga*. Dans ce jeu 2D, l'utilisateur contrôle un vaisseau qui peut se déplacer sur les axes x et y. Des vagues de vaisseaux ennemis se positionnent devant le joueur pour tenter de l'éliminer. Ils se positionnent en groupe et tirent sur le joueur. Le joueur doit faire de son mieux pour esquiver les tirs et doit détruire tous ses ennemis.



FIGURE 2 – Capture d'écran du jeu Galaga

## Troisième partie

# Analyse préliminaire

## 5 Analyse de l'existant

Concernant le mini-jeu, il y a la plateforme et le moteur sur le quel celui-ci se repose. Unreal Engine 4 permet de développer des jeux de manière beaucoup plus rapide et efficace sans ré-inventer la roue. Ceci permet d'éviter d'écrire des bibliothèques pour, par exemple, gérer l'affichage du jeu.

À propos du jeu en lui-même, il n'y a rien de codé ni préparé. Il y a la conception totale à faire.

## 6 Environnement de travail

### 6.1 Logiciel

- Unreal Engine 4<sup>1</sup>
- Outils de Unreal Engine 4
- Visual Studio Professional 2013

### 6.2 Matériel

- PC 1x
  - 1024MB NVIDIA GeForce GTX 550 Ti
  - Intel Core i7 2600K @ 3.40GHz
  - ASUSTeK Computer INC. P8Z68-V LE
  - 8.00 Go Dual-Channel DDR3 @ 824MHz
- Écran 2x
- Clavier 1x
- Souris 1x

### 6.3 Conditions de travail

#### 6.3.1 Durée

Le projet se déroulera sur 48 heures de cours, séparée sur 12 jours et 4 périodes par jour. C'est-à-dire, une fois par semaine, le mercredi matin, 4 périodes seront dédiées à ce projet. Il est aussi possible de travailler sur le projet en dehors des ces tranches horaires.

#### 6.3.2 Délivrables

- Un journal de bord
- Une copie de la documentation
- Un poster qui présente le projet
- La documentation et la source du projet seront déposées sur la plateforme Moodle du CFPT-EI.

#### 6.3.3 Suivi

Ce projet sera évalué et suivi par Monsieur Christophe Maréchal et Monsieur Stéphane Garchery.

---

1. Pour en savoir plus sur Unreal Engine 4 voir la section 3.1 sur la page 5.

## Quatrième partie

# Étude Unreal Engine 4

## 7 Unreal Editor

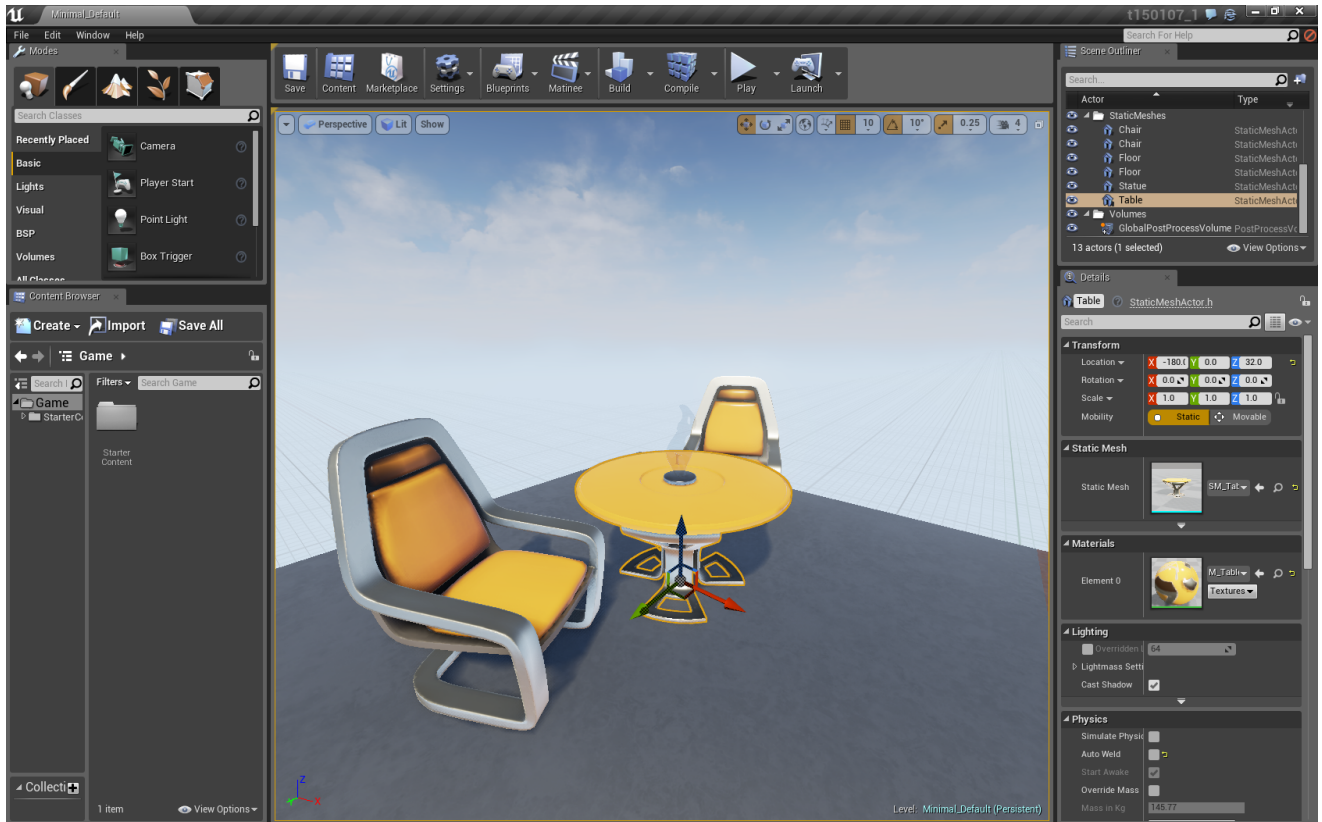


FIGURE 3 – Unreal Editor

### 7.1 Barre de contrôle

La barre de contrôle permet de gérer les points importants du projet, comme la sauvegarde, le contenu, les *blueprints*, la génération et la compilation (voir figure 4).

**Save** permet de sauvegarder le projet.

**Source Control** vérifie l'intégrité du projet.

**Content** affiche les contenus.

**Marketplace** permet de naviguer dans la marché de UE4 pour télécharger du contenu.

**Settings** permet de modifier les paramètres du projet.

**Matinee** permet de gérer les animations.

**Build** génère votre projet.

**Compile** compile votre code C++ pour l'éditeur.

**Play** permet de jouer directement sur l'éditeur.

**Launch** permet de générer la version finale du produit.



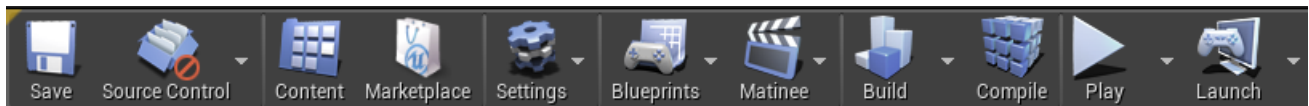
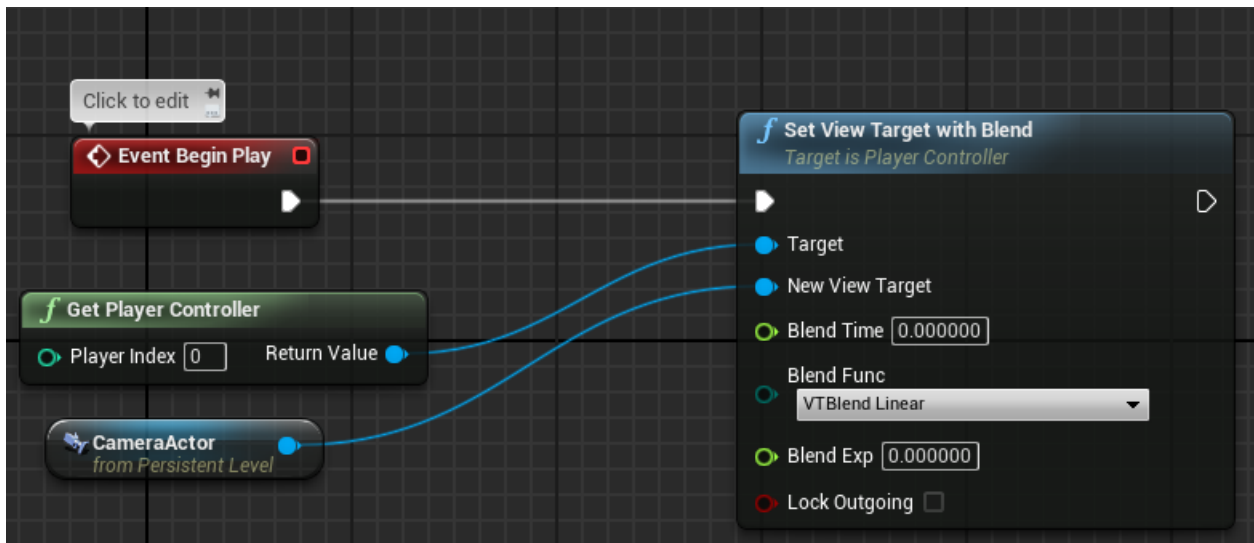


FIGURE 4 – Bar de contrôle de l'éditeur

## 7.2 Blueprints (Plans)

Les *Blueprints* sont des plans qui permettent d'ajouter des fonctionnalités à un objet ou une classe sans devoir passer par du code C++.

### Exemple - Configuration d'une caméra statique

FIGURE 5 – *Blueprint* du niveau pour fixer une caméra statique

Avec l'éditeur de *Blueprint*, il est possible de placer directement des fonctions, des actions et des méthodes dans cette interface graphique afin de produire du code pour le jeu (voir figure 5).

*Event Begin Play* est la méthode de démarrage et action une fonction *Set View Target with Blend* qui permet de mettre en place la caméra utilisée par le jeu. *Get Player Controller* est un *getter* qui permet de récupérer le contrôleur du joueur, il est ensuite lié à la cible *target* de la fonction. La caméra de notre scène est liée comme cible pour la vue par défaut.

Il est ainsi possible grâce à l'utilisation de *Blueprints* de programmer un jeu complet sans toucher au code.

### 7.3 Mesh (Maillage)

Le système de maillage est le squelette de votre objet (ou personnage). Il permet de lui donner des logiques de collision et d'animation.

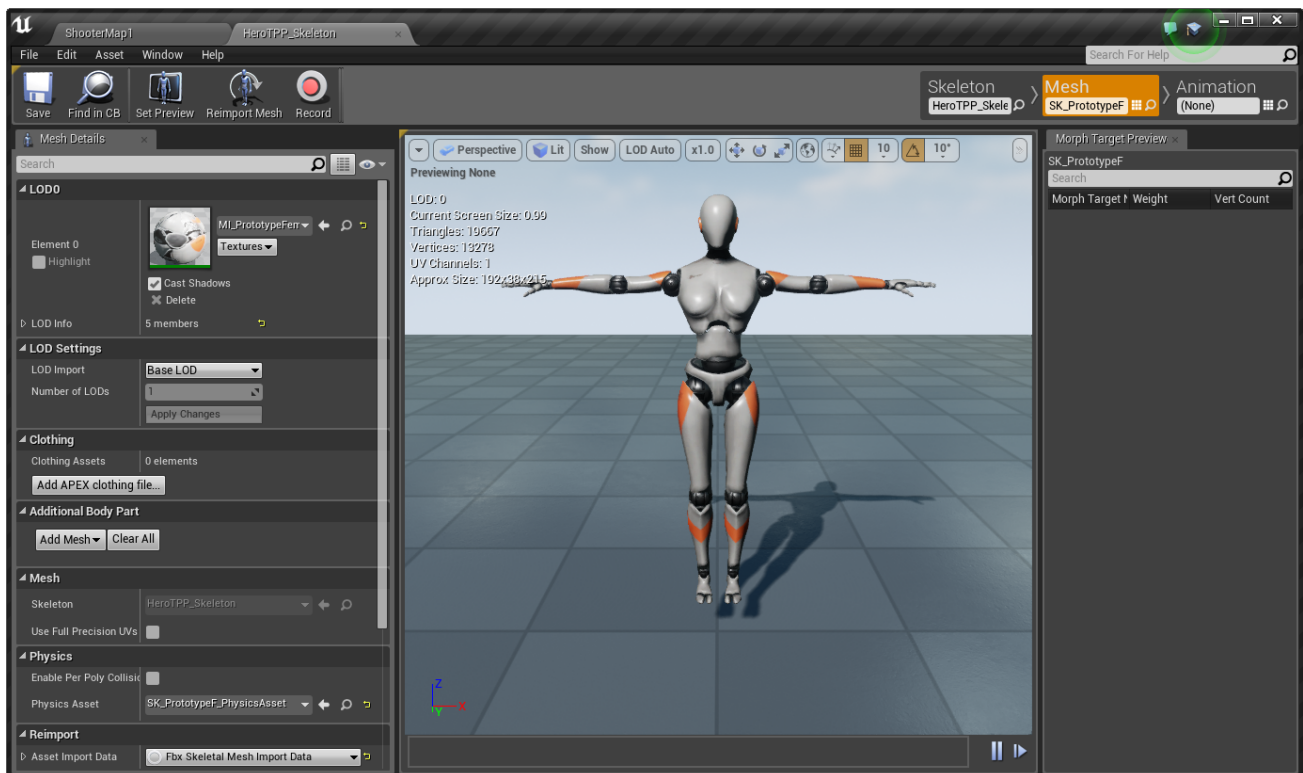


FIGURE 6 – *Matinee* est l'outil utilisé pour l'édition des *meshes*

Sur le coin en haut à droite se trouve 3 modes de vision : "Skeleton", "Mesh" et "Animation" (voir figure 5). "Skeleton" permet de modifier directement le squelette du modèle. "Mesh" permet d'apporter des modifications au modèle dans l'ensemble, incluant les textures et objets supplémentaires. "Animation" permet de gérer les animations du modèle.

## 7.4 Entrées des périphériques

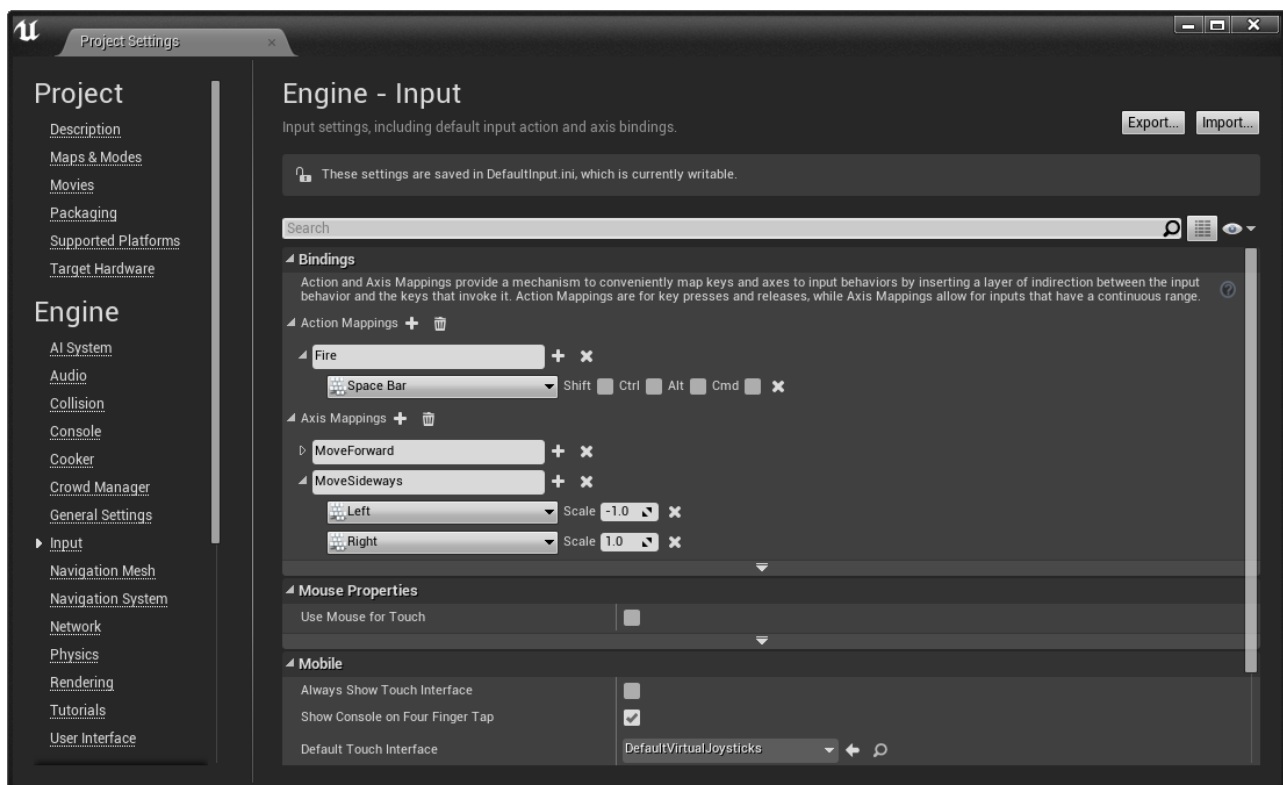


FIGURE 7 – Fenêtre d'édition des entrées de périphériques

Les entrées de périphérique se font dans `"/Edit/Project Settings/Engine/Input/"`. Depuis cette fenêtre il est possible d'ajouter des entrées de périphériques qui pourront être utilisées dans les *blueprints* ou dans le code.

```

1 // Called to bind functionality to input
2 void AShaceShipCharacter::SetupPlayerInputComponent(class
  UInputComponent* InputComponent) {
3   InputComponent->BindAxis("MoveForward", this, &AShaceShipCharacter::
    MoveForward);
4   InputComponent->BindAxis("MoveSideWays", this, &AShaceShipCharacter::
    MoveSideWays);
5   InputComponent->BindAction("Fire", IE_Pressed, this, &
    AShaceShipCharacter::OnFire);
6 }

```

Listing 1 – Exemple de code pour lié des entrées de périphériques à des méthodes de classe

## 8 Langage de programmation

Le langage de programmation utilisé par UE4 est le *C++*, avec UE4 Unreal a décidé de passer tous leurs langages à celui-ci et d'abandonner leur vieux langage de scripting appelé *Unreal Script*.

### 8.1 Les macros

Dans le script C++, Unreal Engine 4 utilise des macros pour identifier le code dans l'éditeur. Grâce à ces macros ces éléments sont modifiables et traitables dans l'éditeur.

Préfixe	Définition
UCLASS()	Se place avant une déclaration d'une classe
UFUNCTION()	Se place avant une déclaration d'une fonction
UPROPERTY()	Se place avant une déclaration d'une variable de classe (Propriété)
UINTERFACE()	Se place avant une déclaration d'interface
USTRUCT()	Se place avant une déclaration d'une structure

TABLE 1 – Liste des macros Unreal Engine 4

### 8.2 Préfixes des classes UE4

Unreal Engine 4 possède des classes pré-faites pour le programmeur. Ces classes sont toujours précédées par un préfixe qui permet de donner petite idée du fonctionnement de la classe. Par exemple si la classe peut être *spawnable*<sup>1</sup> ou non.

Préfixe	Définition
A	S'étend de la classe de base d'objet <i>spawnable</i> du jeu. Ce sont des acteurs et peuvent apparaître directement dans le monde.
U	S'étend de la classe de tous les objets de gameplay. Ces classes ne peuvent pas être instanciées directement dans le monde ; Ils doivent appartenir à un acteur. Ce sont généralement des objets de genre "Composant".

TABLE 2 – Préfixes des classes de Unreal Engine 4

### 8.3 Classes

Il est possible de créer ses propres classes avec l'éditeur. Ceux-ci sont ajoutés au projet et sont modifiables.

#### 8.3.1 Gamemode

Gamemode contient la définition du jeu lui-même, comme les règles, les conditions de victoire, etc. Il met aussi en place les classes par défaut à utiliser pour du gameplay du framework basique, comme *Pawn*, *PlayerController*, et l'*HUD*.

#### 8.3.2 Actor

Actor est la classe de base d'un objet qui peut se placer sur le monde.

#### 8.3.3 Pawn

Pawn est la classe de base de tous les *actor* qui peuvent être contrôlés par un utilisateur ou une IA.

1. Élément pouvant apparaître dans le monde

### 8.3.4 Character

Les *Characters* sont des *Pawns* qui possèdent un maillage, collisions et des mouvements logiques intégrés.

## Cinquième partie

# Analyse fonctionnelle - Jeu

## 9 Fonctionnalités

### 9.1 Règles du jeu

**Introduction** Les règles du jeu sont simple, le joueur contrôle un vaisseau et doit détruire le plus d'ennemis possible avant d'affronter un boss<sup>1</sup>. Chaque ennemi détruit donne des points au joueur.

**But** L'objectif du jeu est de détruire le boss final<sup>2</sup> pour valider la partie et avoir le plus grand score.

**Niveaux** Les niveaux sont des parties que le joueur doit accomplir pour passer au prochain niveau ou au boss final. À chaque fois que le joueur accomplit un niveau, la difficulté augmente.

**Mécanismes de bases** Il y a quelques paramètres que chaque *vivant* du jeu possède.

- **Santé** Les points de vie.
- **Dégâts** Les dégâts infligés à l'adversaire. Représente les points de vie retiré à l'adversaire si le *missile* le touche.
- **Bonus** Les bonus que l'individu possède pour augmenter ses propriétés (Santé et Dégâts).

### 9.2 Fonctionnement

#### 9.2.1 Joueur

**Évolution** Le joueur évolue à travers le jeu grâce à des bonus qu'il peut récupérer. Il y a des différents types de bonus :

- Régénération de santé
- Résistance
- Type de missiles
- Augmentation des dégâts infligés
- Et autres...

**Comportement** Le joueur perd le dernier bonus récupérer et un point de vie lorsqu'il se fait toucher par un ennemi.

**Contrôle** Le vaisseau contrôlé par le joueur se trouve sur le bas de la zone jouable (de l'écran). Celui-ci peut seulement se déplacer sur 2 axes, c'est-à-dire que les mouvements disponibles sont : haut, bas, gauche, droite. L'utilisateur utilise les touches du clavier pour se déplacer dans l'environnement. Les touches par défaut sont :

Touche principale	Touche secondaire	Action
W	FLÈCHE_HAUT	Déplacement vers le haut
S	FLÈCHE_BAS	Déplacement vers le bas
D	FLÈCHE_DROITE	Déplacement vers la droite
A	FLÈCHE_GAUCHE	Déplacement vers la gauche
ESPACE	∅	Tirer

TABLE 3 – Touchent par défaut du jeu

---

1. Un boss est une unité ennemi qui possède plus de points de vie et d'attaques pour détruire le vaisseau du joueur  
2. Le boss final ne possède pas plus de caractéristiques qu'un boss normal, mais permet de terminer le jeu

### 9.2.2 Ennemis

**Comportement** Les ennemis apparaissent selon la progression du joueur ou du temps. Ils se mettent en position au centre de l'écran au dessus du joueur. Ils effectuent ensuite des manœuvres pour essayer de détruire le vaisseau joueur, soit en tirant soit en se déplacent sur l'écran.

**Boss** Les boss sont des ennemi *ultimes* qui se trouve à la fin de chaque niveau. Un boss possède assez de points de vie pour que le niveau final dure environ 2 minutes ou plus (varie selon la difficulté), de divers bonus et des attaques qui lui sont propres.

## 10 Interface homme-machine

### 10.1 Écran du jeu

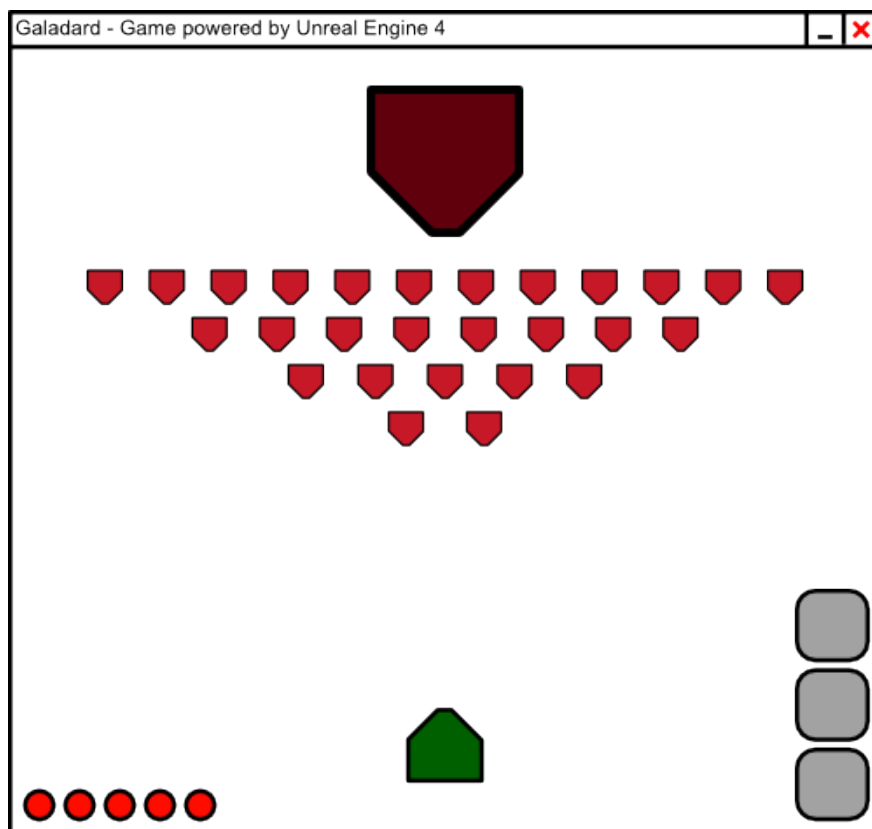







FIGURE 8 – Maquette de l'interface du jeu

-  Vaisseau du joueur
-  Point de vie du joueur
-  Bonus que le joueur possède
-  Unité ennemi
-  Boss ennemi

## 10.2 Menu du jeu

Le menu du jeu est très basique, il possède :

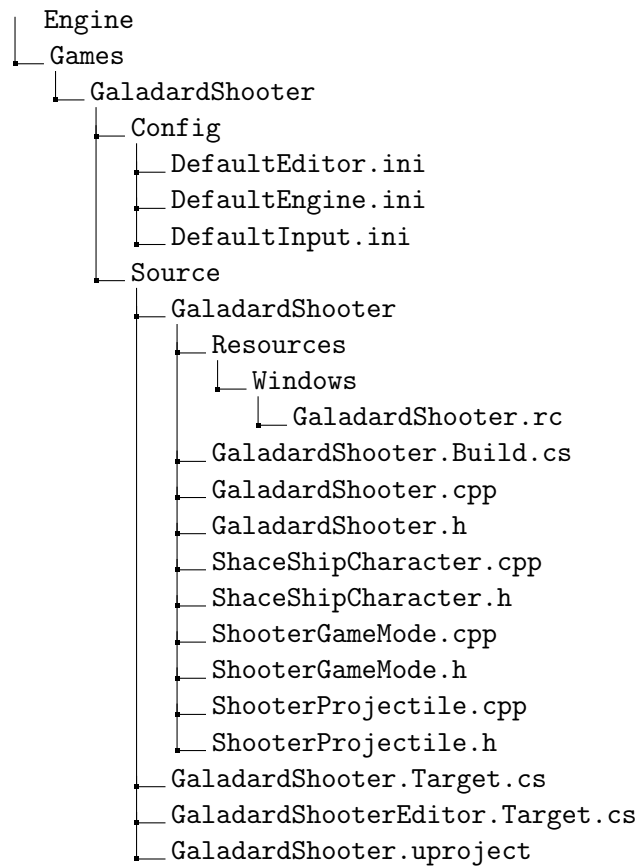
- Bouton pour démarrer une nouvelle partie
- Slider pour régler la difficulté du jeu
- Bouton pour afficher les scores
- Bouton pour quitter l'application



## Sixième partie

# Analyse Organique - Jeu

## 11 Arborescence du projet



## 12 Mode de jeu

Le mode de jeu par défaut de UE4 est remplacé par une classe vide pour retirer tous les fonctionnalités par défaut qui sont indésirés.

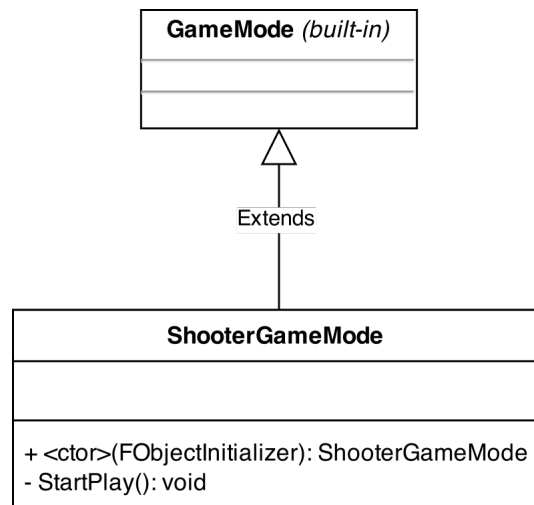


FIGURE 9 – Diagramme de classe de *ShooterGameMode*

Le constructeur permet de mettre en place un *blueprint* qui ajoute le joueur dans le monde. Le *StartPlay* permet de démarrer le jeu et d'afficher un message de confirmation sur l'écran pendant 5 secondes.

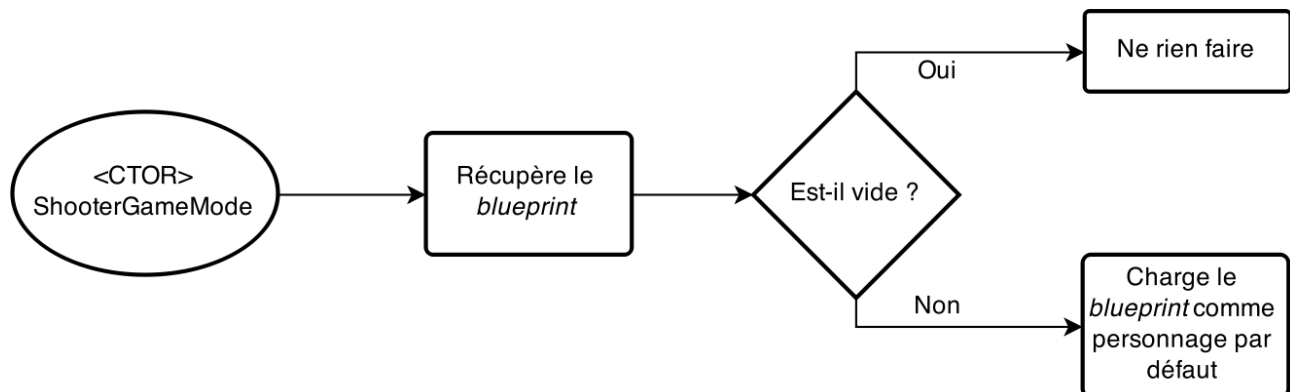


FIGURE 10 – Diagramme de flux du constructeur de la classe *ShooterGameMode*

## 13 Personnage

Le personnage du jeu est associé au modèle prototype de Unreal Engine, parce qu'il serait beaucoup trop long et complexe de faire un modèle et ça ne correspond pas au métier d'informaticien. Le maillage du modèle est chargé dans un *blueprint* (voir figure 11).

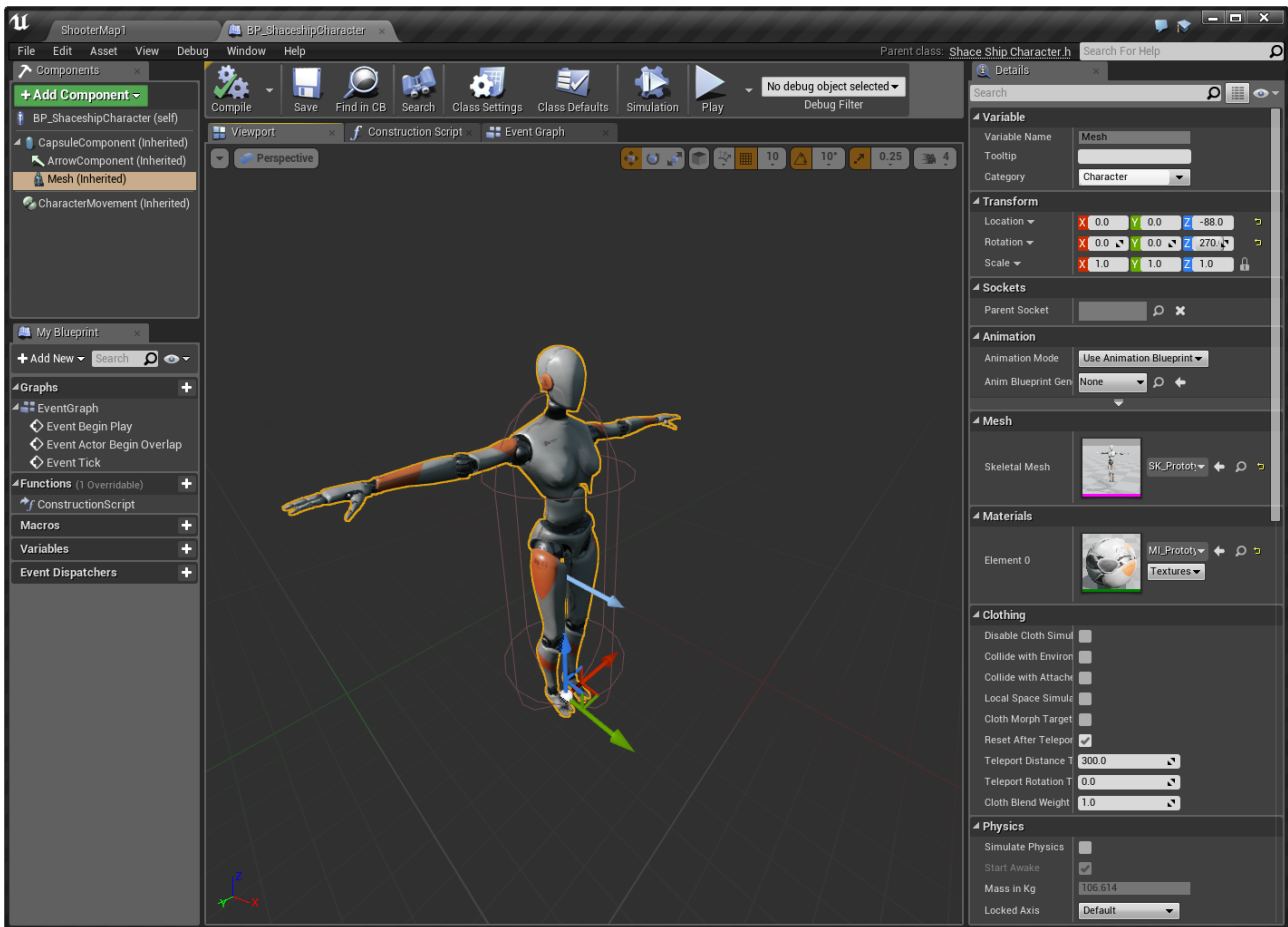


FIGURE 11 – *Blueprint* du personnage utilisé dans le jeu

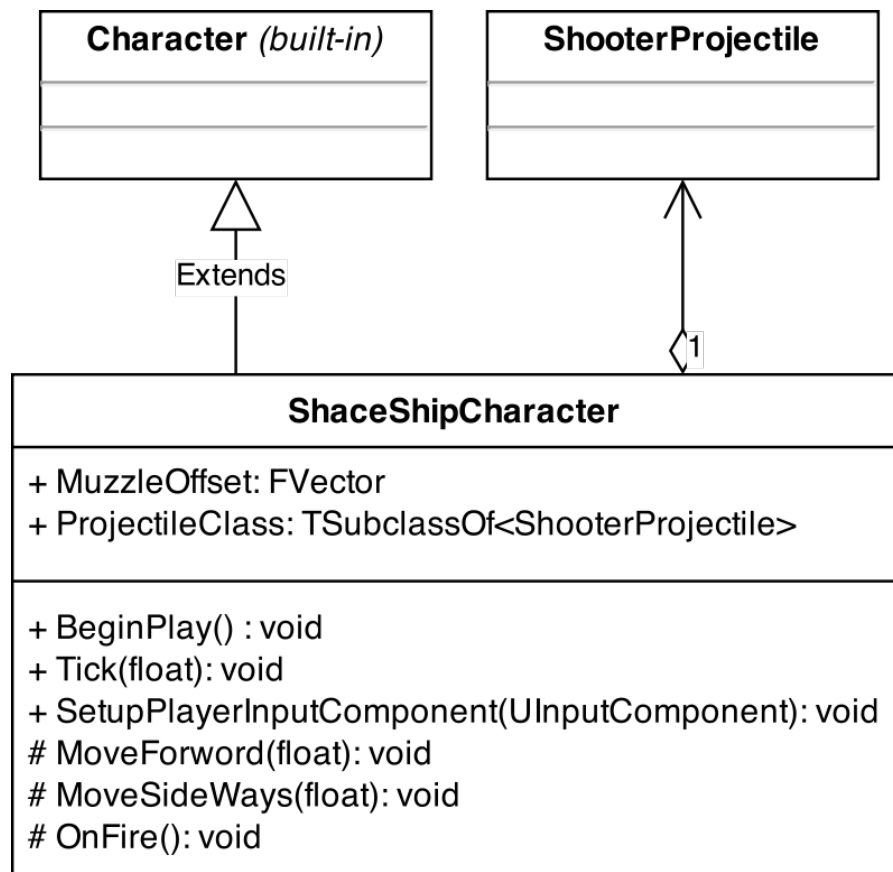
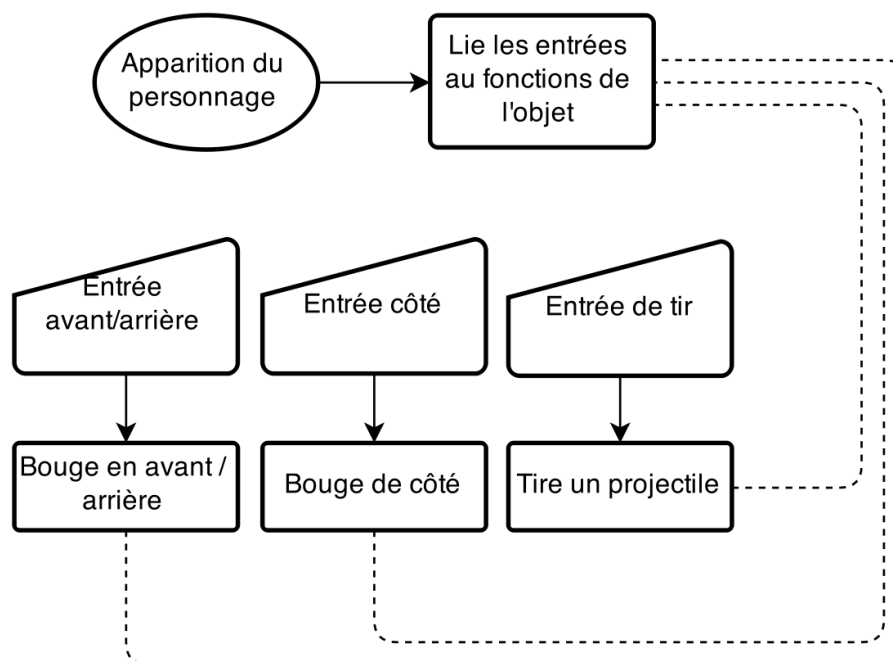
Pour contrôler ce personnage, une classe a été créée. Il hérite de la classe *Character* qui est une classe intégrée à Unreal Engine qui définit tous les personnages réagissant aux fonctions logiques du moteur, par exemple, la physique et le déplacement. La classe du personnage, *ShaceshipCharacter*, possède des actions de mouvement et de tir (voir figure 12).

Lors de la création du personnage, le moteur lie les entrées spécifiées dans le système à l'objet si celui-ci possède les mêmes noms (voir figure 13).

Les mouvements latéraux et avant / arrière sont gérés avec le *Controller*, une propriété que les classes héritant de *Character* possèdent, qui permet de gérer le contrôle du personnage avec les entrées de périphériques. Pour le fonctionnement des mouvements latéraux voir le listing 2 et la figure 14.

Le tir du personnage est des objets (*ShooterProjectile*) qui sont créés par le biais du personnage. Le projectile est tiré en avant et apparaît sur la position du *muzzle*<sup>1</sup> qui évite que celui-ci n'apparaisse dans le personnage.

1. La position définie dans l'éditeur du *blueprint* du personnage

FIGURE 12 – Diagramme de classe de *ShaceShipCharacter*FIGURE 13 – Diagramme de flux du démarrage de *ShaceShipCharacter*

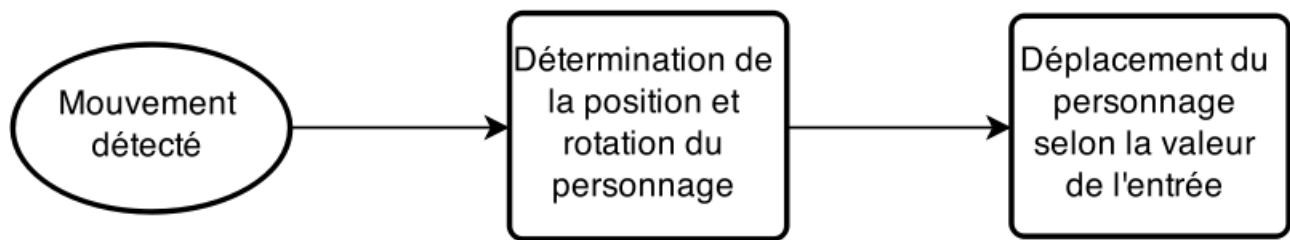


FIGURE 14 – Diagramme de flux du déplacement du personnage

```
1 // Moves the character sideways depending on the input value
2 // Determines the direction the character is facing and applies
3 // the correct movement and direction to the character
4 void ASpaceShipCharacter::MoveSideWays(float Value)
5 {
6     if ((Controller != NULL) && (Value != 0.0f))
7     {
8         // find out which way is right
9         const FRotator Rotation = Controller->GetControlRotation();
10        const FVector Direction = FRotationMatrix(Rotation).GetScaledAxis(
11            EAxis::Y);
12        // add movement in that direction
13        AddMovementInput(Direction, Value);
14    }
```

Listing 2 – Gestion des mouvement latéraux

## 14 Projectiles

Les projectiles sont des objets qui sont utilisés par le personnage, contrôlé par le joueur, pour tirer. Ils sont instanciés au moment du tir dans la direction que le personnage fait face.

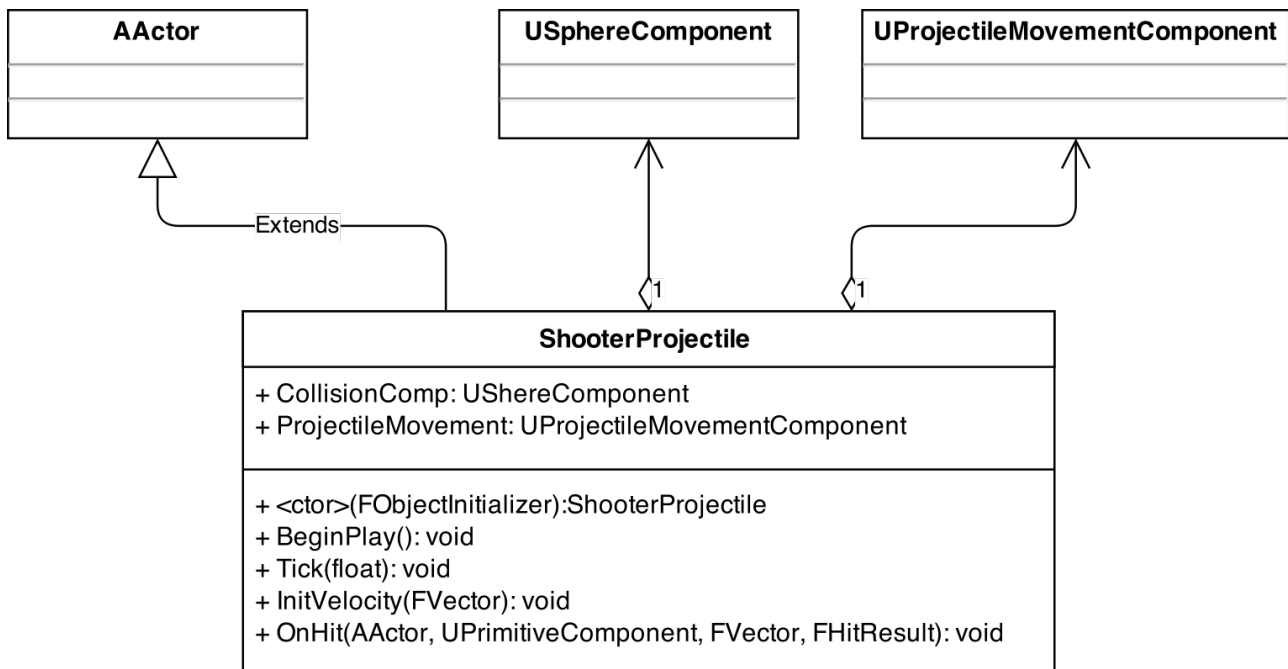


FIGURE 15 – Diagramme de classe de *ShooterProjectile*

## 15 Collisions

Pour faire des collisions avec les objets personnalisés, il est nécessaire de directement modifier les fichiers d'initialisation du moteur.

Il faut ajouter un canal de réponse par défaut au profil de collision du moteur et d'indiquer quel éléments y sont affecté (voir listing 3).

```

1 // Declacration of the new channel
2 [/Script/Engine.CollisionProfile]
3 +DefaultChannelResponses=(Channel=ECC_GameTraceChannel1, Name=
   Projectile)
4
5 // Elements added to the channel
6 +Profiles=(Name="Projectile", CollisionEnabled=QueryOnly, ObjectTypeName=
   Projectile, CustomResponses=( \
7   (Channel=Static, Response=ECR_Block), \
8   (Channel=PawnMovement, Response=ECR_Block), \
9   (Channel=Dynamic, Response=ECR_Block), \
10  (Channel=PhysicsBody, Response=ECR_Block), \
11  (Channel=VehicleMovement, Response=ECR_Block), \
12  (Channel=Destructible, Response=ECR_Block) \
13  ))
14 }
```

Listing 3 – Paramètres à ajouter au moteur par défaut

## Septième partie

## Tests

Tests	Réalisation	Résultat attendu	Résultat obtenu
Le personnage se déplace latéralement	Les flèches gauche et droite sont actionnées lors du jeu	Le personnage se déplace latéralement	OK
Le personnage peut avancer et reculer	Les flèches haut et bas sont actionnées lors du jeu	Le personnage se déplace en avant et en arrière	OK
Le personnage peut tirer	La barre d'espace est actionnée lors du jeu	Le personnage tir un projectile	OK
Le personnage peut mourir	Le personnage se prend des tirs ennemis	Le personnage meurt lorsqu'il recoit trop de tirs	Les points de vie ne sont pas implémentés
Obtention de bonus	Le personnage se déplace sur un bonus	Le personnage gagne plus de dégats avec ses projectiles lorsque qu'il attrape un bonus	Les bonus ne sont pas implémentés
Ennemis tirent	Observé les ennemis lors du jeu	Les ennemi doivent tirer des projectiles	Les ennemis ne sont pas implémentés
Les projectiles ont des collisions	Le personnage tir un projectile et observe le comportement de celui-ci	Le projectile doit rebondir sur les bords	OK

## Huitième partie

# Conclusion

### 16 Conclusion technique

#### 16.1 Bilan de l'étude

La cible visée de l'étude était de connaître de manière plus approfondie les méthodes utilisées dans le monde professionnel du jeu vidéo de grande taille. Bien que ces produits soient simplifiés et optimiser pour le développement de ces produits divertissants, ceci demande tout de même un travail considérable et bien réfléchi. La recherche ma permis de comprendre ces fonctionnements de manière explicite. Il est clair qu'avec ce travail, je n'ai qu'éraflé la pointe de l'iceberg, mais j'ai tout de même appris et observer beaucoup de ces accessoires et ça m'a permis d'acquérir les connaissances de fondamentales. Il est très intéressant d'observer que ce genre de développement avec un tel outil est considérablement différent par rapport au développement d'applications lambda.

#### 16.2 Bilan du jeu exemple

Le but de ce jeu était de tester mes connaissances avec les données acquises de l'étude. Le résultat final montre clairement qu'il faut apporter un travail plus important pour présenter un produit vendable. Celui-ci est malheureusement non-acquis, car le travail fourni était insuffisant, la durée du projet était mal calculée et le jeu ne correspond pas à l'analyse fonctionnelle faite, mais j'ai tous de même pu développer un semblant de prototype d'un jeu. J'ai pu apprendre de mes erreurs de planifications et même de programmation. Unreal Engine 4 s'est avéré comme un éditeur très complexe à utiliser par rapport à *Unity* (des cours de *Unity* ont été donnés par M. Aliprandi, et il s'est montré plus simple d'utilisation de mon point de vue personnel).

#### 16.3 Améliorations

Il serait possible de fournir d'amples améliorations pour le jeu comme : les ennemis, une IA, etc. mais ceci aurait demander beaucoup plus de temps de travail (je donne une estimation d'environ 1 mois de plus).



## 17 Conclusion personnelle

### 17.1 Bilan personnel

Sur l'étude, je suis satisfait de mes recherches, car ceci a pu me faire changer d'avis par rapport au développement de gros jeux et ma plutôt faite tourner ma tête en direction de *Unity* qui est plus ciblée pour les jeux de petite taille (bien qu'il soit possible de faire des jeux de tailles importantes avec l'outil).

Je suis assez déçu de moi-même par rapport au développement du jeu exemple de cette recherche, car il ne correspond en rien avec le produit que je voulais produire. Malheureusement j'ai été emporté par plusieurs événements qui ne m'ont pas permis de fournir la quantité et qualité de travail souhaité.

### 17.2 Apport personnel

La recherche a été faite avec l'aide du site officiel de Unreal Engine et de quelques tutoriels trouvés sur Internet. La totalité du jeu a été fait par moi-même avec les composants et fonctionnalités apprises pendant l'étude, aucun tutoriel ne correspondait au jeu que je voulais réaliser donc j'ai dû me lancer par instinct pour le réaliser.

### 17.3 Gestion du temps

Une estimation de 70% du travail fourni était dédié à l'étude, mais le travail fourni en lui-même n'était pas suffisant à cause d'événements personnels. La majorité du jeu a été fait en dehors des cours dédiés au projet.

## Neuvième partie

## Annexes

## 18 Planning

## 18.1 Initial

Planning de gantt - Étude de Unreal Engine 4														
	03.12	10.12	17.12	06.01	07.01	13.01	14.01	21.01	28.01	04.02	18.02	04.03	10.03	18.03
Préparation préliminaires														
Étude														
Analyse fonctionnelle														
Analyse organique														
Développement														
Tests														
Documentation														

FIGURE 16 – Planning initial du projet

## 18.2 Réel

Planning de gantt - Étude de Unreal Engine 4														
	03.12	10.12	17.12	06.01	07.01	13.01	14.01	21.01	28.01	04.02	18.02	04.03	10.03	18.03
Préparation préliminaires														
Étude														
Analyse fonctionnelle														
Analyse organique														
Développement														
Tests														
Documentation														

FIGURE 17 – Planning final du projet

## 19 Références bibliographiques

1. Mike McShaffry and David "Rez" Graham. 2013. *Game Coding Complete, Fourth Edition*. Boston : Course Technology PTR.
2. Scott Rogers. 2010. *Level Up! The Guide To Great Video Game Design*. Chichester : John Wiley & Sons, Ltd.
3. Mat Buckland. 2005. *Programming Game AI by Example*. Sudbury : Wordware Publishing, Inc.
4. UDK - Tutorials List. Visité le 06/01/2015.  
[http://www.worldofleveldesign.com/categories/cat\\_udk.php](http://www.worldofleveldesign.com/categories/cat_udk.php)
5. Platformer Started Kit. Visité le 06/01/2015.  
<http://udn.epicgames.com/Three/DevelopmentKitGemsPlatformerStarterKit.html>
6. Setting Up Visual Studio for UE4. Visité le 07/01/2015.  
<https://docs.unrealengine.com/latest/INT/Programming/Development/VisualStudioSetup>

## Table des figures

1	Souris Novint Falcon 3D Touch Controller . . . . .	6
2	Capture d'écran du jeu Galaga . . . . .	6
3	Unreal Editor . . . . .	8
4	Bar de contrôle de l'éditeur . . . . .	9
5	<i>Blueprint</i> du niveau pour fixer une caméra statique . . . . .	9
6	<i>Matinee</i> est l'outil utilisé pour l'édition des <i>meshes</i> . . . . .	10
7	Fenêtre d'édition des entrées de périphériques . . . . .	11
8	Maquette de l'interface du jeu . . . . .	15
9	Diagramme de classe de <i>ShooterGameMode</i> . . . . .	18
10	Diagramme de flux du constructeur de la classe <i>ShooterGameMode</i> . . . . .	18
11	<i>Blueprint</i> du personnage utilisé dans le jeu . . . . .	19
12	Diagramme de classe de <i>ShaceShipCharacter</i> . . . . .	20
13	Diagramme de flux du démarrage de <i>ShaceShipCharacter</i> . . . . .	20
14	Diagramme de flux du déplacement du personnage . . . . .	21
15	Diagramme de classe de <i>ShooterProjectile</i> . . . . .	22
16	Planning initial du projet . . . . .	26
17	Planning final du projet . . . . .	27

## Liste des tableaux

1	Liste des macros Unreal Engine 4 . . . . .	12
2	Préfixes des classes de Unreal Engine 4 . . . . .	12
3	Touchent par défaut du jeu . . . . .	14

## Listings

1	Exemple de code pour lier des entrées de périphériques à des méthodes de classe . . . .	11
2	Gestion des mouvement latéraux . . . . .	21
3	Paramètres à ajouter au moteur par défaut . . . . .	22