Année universitaire : 2021/2022

UNIVERSITE D'ANTANANARIVO

ECOLE SUPERIEUR POLYTECHNIQUE

Domaine : Sciences de l'ingénieur

Mention : Ingénierie des Systèmes

Avancés

Parcours à visée de recherche : Ingénierie des Sciences Cognitives

MEMOIRE DE RECHERCHE EN VUE

D'OBTENTIONDU TITRE DE MASTER A

VISEE DE RECHERCHE

Titre :

INTELLIGENCE ARTIFICIELLE : APPLICATION DANS LE DEVELOPPEMENT DE JEUX VIDEO

Présenté par : **MEFIRE FAGOUOP Junior Cardin**

Date de Soutenance : 05 Juillet 2022

Année universitaire : 2021/2022

UNIVERSITE D'ANTANANARIVO

_____

ECOLE SUPERIEUR POLYTECHNIQUE

_____

Domaine : Sciences de l'ingénieur

Mention : Ingénierie des Systèmes

Avancés

Parcours à visée de recherche : Ingénierie des Sciences Cognitives

MEMOIRE DE RECHERCHE EN VUE

D'OBTENTIONDU TITRE DE MASTER A

VISEE DE RECHERCHE

Titre :

INTELLIGENCE ARTIFICIELLE : APPLICATION DANS LE DEVELOPPEMENT DE JEUX VIDEO

Présenté par : **MEFIRE FAGOUOP Junior Cardin**

Président :
Monsieur ANDRIAMANOHISOA Hery Zo, Professeur

Examinateur :
Monsieur RANDRIAMAMONJY Joharinirina, Docteur en sciences cognitives
Monsieur RABEARIVELO GERICHA, Maitre de conférences
Madame RAKOTONIAINA Rabenoro Barry, Docteur en sciences cognitives

Directeur de mémoire :
Monsieur ROBINSON Matio, Maître de conférences

Remerciements

Cet œuvre est le fruit d'une longue patience et d'un long parcours où j'ai enduré tant de souffrance qui m'a ensuite fortifié, tant d'inquiétude qui m'a rendu confiant tant d'obstacle que j'ai surmonté avec l'aide de mon Seigneur tout puissant. Et c'est ainsi que ce chemin a été traversé avec autant d'espoir, de passion et d'amour

Ainsi « je dédie ce mémoire » à :

- A **DIEU** tout puissant, source de la sagesse et de l'intelligence, pour m'avoir conseillé, donné la bonne santé, la patience et la force d'arriver au bout de cette étude.
- A Monsieur **Mamy Raoul RAVELOMANANA**, Professeur titulaire – président de l'université d'Antananarivo d'avoir accepté mon inscription au sein de l'université.
- A Monsieur **RAKOTOSAONA Rija** Directeur de l'ESPA pour m'avoir fait l'honneur
- d'évaluer mes travaux de recherche.
- A tous les **Membres du jury** merci pour leurs remarques et pour leur contribution à la révision de la forme finale de ce travail
- A mon directeur de mémoire **ROBINSON Matio**, Maître de conférences, Responsable de la mention ISA de m'avoir accompagné tout au long de la rédaction de ce mémoire.
- A la famille **MEFIRE**, merci pour le soutien moral et matériel que vous n'avez pas cessé de m'apporter le long de mes études.
- A la famille **NZOKIZA**, merci d'avoir toujours été là pour moi, de m'avoir donné la chance de faire partie de votre famille.
- A la famille de ma fiancée **MVEMBE Fanny Nicaise**, merci pour votre soutien aussi bien moral que matériel et votre confiance en mon égard.
- A la famille **WETE**, merci pour votre générosité, votre gentillesse, votre disponibilité, et votre bonté.
- A **ONDO**, **PIERRE**, **FERNAND, GUY** merci d'avoir toujours été là.

Table des Matières

Notations et Symbole

BP : Blueprint

BPI_ : Blueprint Interface

WBP_ : Widget Blueprint

BS_ : Blend Space

SK_ : Skeletal Mesh

IA : Intelligence Artificielle

AIC_ : AI Controller

ABP_ : Animation Blueprint

M_ : Material

T_ : Texture

GOAP : Goal Oriented Action Planning

PNJ : Personnage Non Joueur

GAN : generative antagonists networks

MCTS : Monte-Carlo tree search

UMG : Unreal Motion Graphic

UI : User Interface

UE4 : Unreal Engine 4

Table des figures

Introduction générale

L’objectif du présent mémoire est de montrer l’application de l’Intelligence Artificielle dans le développement du jeu vidéo**.**

Il porte dans un premier temps sur la définition et l’origine du terme d’Intelligence Artificielle, concept trouvant ses sources dans l’Antiquité.

Dans le domaine du jeu vidéo, l'Intelligence Artificielle est utilisée pour proposer une expérience de jeu de qualité, par exemple en conférant aux personnages non-joueurs un comportement s'approchant de celui de l'être humain, d'une interaction logique, ou en limitant les compétences du programme afin de donner au joueur un sentiment d’équité...

Deuxièmement montré comment est implémenté une IA dans un jeu à travers un moteur graphique, puis comment fonctionne un moteur graphique. Nous ferons ensuite une présentation détailler du moteur graphique Unreal Engine 4 Et de son système de programmation intégré

Troisièmement Nous nous attarderons principalement sur la création d’un jeu de type Topdown Shooter sur Unreal Engine 4.26. Nous exposerons ainsi tous les différents systèmes d’Intelligence Artificielle mis en jeux et comment communiquent-elles au saint de celui-ci.

1 CHAPITRE 1 : L'Intelligence Artificielle et Système Graphique

1.1 Introduction

L'Intelligence Artificielle est l'ensemble des outils qui vont permettre de reproduire un comportement humain, animal et plus généralement tout système cognitif.

C'est-à-dire tout système complexe de traitement de l'information capable d'acquérir, conserver, utiliser et transmettre des connaissances soit avec un Algorithme soit avec différents outils qui vont permettre de simuler une intelligence.

Un algorithme quant à lui c'est une série d'instructions à suivre dans un certain ordre pour effectuer une tache de manière automatique.

Qui dit automatique dit sans réfléchir, et c'est parfait pour une machine. Une fois qu'on a cette liste d'instruction on le traduit en langage compréhensible par un ordinateur et on obtient un programme informatique. Les ordinateurs exécutent parfaitement et très rapidement tout programme qui décrit un algorithme, D'ailleurs, dans les années 50 les chercheurs se sont mis à rêver, ils ont imaginé des algorithmes qui rendraient les ordinateurs aussi intelligents que l'humain.

Dans ce chapitre, nous allons nous intéresser à la notion d'Intelligence Artificielle dans sa généralité. Et à la notion de moteur graphique

1.2 Généralités

L'Intelligence Artificielle comme on la connaît est un domaine qui chaque jour évolue un peu plus en s'appropriant des capacités cognitives humaines, en les développant et parfois même en surpassant ce que les meilleurs êtres humains dans leur domaine sont capables de réaliser : nous pouvons prendre l'exemple de l'AlphaGo Zero de Google qui, en seulement trois jours et avec pour seules règles celles du jeu de Go, a vaincu à la fois le champion du monde de ce jeu, ainsi que l'ancienne Intelligence Artificielle qui était elle-même dédiée à ce jeu. Les prouesses réalisées par cette IA sont le fruit d'une amélioration quotidienne des techniques et des modèles appliquées.

Mais celle-ci ne se démarque pas seulement dans les jeux et se retrouve dans d'autres domaines : les systèmes de recommandations sur les sites de ventes en ligne ou autres plateformes de streaming utilisent également les intelligences artificielles en se basant à la fois sur les habitudes d'achats / de visionnage et de navigation d'un utilisateur, mais également des autres utilisateurs qui recherchent les mêmes produits / contenus pour proposer une expérience personnalisée à chacun. On les retrouve également dans les assistants virtuels, comme la Google Home ou Alexa d'Amazon, qui sont capables d'analyser les paroles d'un individu, de les comprendre, et de pouvoir répondre à la demande formulée lorsque celle-ci est dans les possibilités de la machine.

Elle a également pour objectif de s'immiscer dans d'autres domaines : celui de la santé, où elle sera utilisée sur les imageries médicales pour détecter des anomalies dans le corps des personnes et analyser plus efficacement le problème que la personne rencontre, ou pour prévenir les possibles maladies qui pourraient se déclarer en cernant les premiers symptômes.

On les retrouve dans les nouvelles voitures autonomes, celles qui ont pour objectif de pouvoir déplacer les hommes d'un point A à un point B sans intervention aucune de leur part tout en réduisant les risques d'accidents liés aux états physiques de l'humain. Les drones feront également apparition dans nos vies.

Il est bon aussi de s'intéresser aux limites de l'Intelligence Artificielle pour prendre pleinement conscience de ce qui est faisable de ce qui ne l'est pas. Nous pouvons prendre un exemple très simple : à ce jour, aucune machine dotée d'une Intelligence Artificielle ne peut

savoir si, en général, un programme retournera pour une entrée donnée une réponse ou s'il s'exécutera à l'infini [1].

Certains problèmes connaissent une croissance exponentielle dans leur temps de traitement en fonction de la taille des exemples qui sont donnés. Il est possible qu'un problème ne puisse être résolu dans le temps à l'échelle humaine, et c'est pour cela que des techniques de traitement sont apparues pour pallier ce souci, comme le fait de pouvoir subdiviser un problème en sous-problèmes qui sont quant à eux résolubles.

De nos jours l'état de l'art dans le monde de l'Intelligence Artificielle comporte le Transfer Learning. Il s'agit d'une méthode d'apprentissage dans laquelle on suppose que la connaissance acquise par un modèle entraîné de Machine Learning peut être "transférée" pendant le processus d'entraînement de ce dernier. Cela permet de réduire la quantité de données nécessaires à un modèle pour apprendre une nouvelle tâche. Dans le contexte actuel où l'on cherche à généraliser le plus possible les modèles afin qu'ils puissent traiter plusieurs types de problèmes différents, cela consiste en une étape clé vers ce but

Le Transfer Learning a par ailleurs déjà fait ses preuves : dans le cadre de la détection d'un cancer de la peau, l'InceptionV3 qui a été entraîné dans un premier temps sur Image Net puis sur plus de 100 000 images avec plus de 2000 différentes maladies de la peau a pu surpasser en matière de performance des dermatologistes reconnus. Il s'agit d'une grande avancée dans le traitement des données, et également dans l'objectif de permettre aux hommes d'être assistés par des machines performantes.

Les évolutions dans le domaine de l'Intelligence Artificielle s'accompagnent également des évolutions dans le monde des hardwares : des composants plus puissants, qui permettent des traitements de plus en plus lourds en calculs et de réduire les temps de ce des derniers de façon optimale grâce à la parallélisation, sont à la source même de ces évolutions. Une plus grande parallélisation des traitements signifie un entraînement et une itération des modèles beaucoup plus rapide. C'est une frontière que l'on cherche constamment à repousser et qui chaque année nous offre son lot de surprises.

C'est d'ailleurs grâce à ces améliorations que des intelligences artificielles comme AlphaGo Zero peuvent atteindre un seuil de performance élevé dans un jeu comme le Go qui offre des milliards de possibilités. Cependant, les évolutions en termes de performance peuvent être limitées actuellement. C'est pourquoi en plus de ces composants, il est aussi important

d’imaginer de nouvelles architectures pour ces derniers qui permettent de mettre à profit au maximum leurs capacités.

Dans un contexte plus ludique mais tout aussi incroyable, OpenAI a monté une équipe de cinq agents (Intelligence Artificielle) avec chacun son propre réseau de neurones, entraînés pour leurs faires affrontés des équipes de vrais joueurs dans le jeu Dota 2. Les résultats sont intéressants puisque cette équipe d’agents entraînés a pu battre certaines des équipes qui les ont affrontés.

Un jeu comme Dota 2 est un véritable challenge puisque l’espace des possibilités et des actions de chaque agent sur une map aussi grande (un carré d’environ 220 mètres de côté) est immense et représente un challenge technique de taille, en plus du fait qu’il s’agit d’un jeu à information partielle (jeu où l’on ne possède pas la vision sur tout ce qu’il se passe

Récemment dans un domaine on l’on n’attendait pas forcément sa venue, un tableau ‘'Portrait d'Edmond Bellamy' créé par une Intelligence Artificielle était estimé entre 5000 euros et 10 000 euros, mais a finalement été vendu par une société de vente aux enchères (Christie’s à New York) pour la somme de 432 000 dollars soit 381 000 euros. Le portrait fait partie d’une série de 11 œuvres réalisées par un collectif d’artistes spécialisés dans l’Intelligence Artificielle installé à Paris, généré à partir de 15 000 portraits peints entre le 14ème et le 20ème siècle. Ce n’est pas la première fois que l’on lance une Intelligence Artificielle dans le milieu de l’art, nous avons par exemple le fameux “Deep Dreams” de Google mais c’est la première fois que la vente d’une œuvre artistique réalisée par une IA atteint une telle ampleur.

Dans un autre registre nous avons aussi “1 The Road” un livre écrit par une Intelligence Artificielle conçue par Ross Goodwin. Il a également été aidé par le département Art + Intelligence de Google (Google AI). L'IA s'est imprégnée du vocabulaire des classiques de la littérature anglo-saxonne, des phrases, des structures et des rythmes des textes, puis les équipes travaillant sur cette IA ont équipé une Cadillac d’une caméra de surveillance, d’un GPS, d’un microphone et d’une horloge. Ainsi pendant un trajet de New York à la Nouvelle-Orléans (environ 2000km) et durant ce voyage, l'Intelligence Artificielle écrivait un récit, en s'aidant des différents capteurs ajoutés à la voiture. L'IA a intégré des personnages à son récit et à part le retrait de certains passages pour mieux rythmer le texte, aucune réécriture n'a été réalisée.

1.2.1 Intelligence Artificielle

L'Intelligence Artificielle est aussi une discipline de l'informatique qui a pour but de créer des machines intelligentes, en "opposition" avec l'intelligence naturelle des êtres vivants. Le terme a beaucoup évolué au fil du temps, il englobe dorénavant toutes les idées visant à permettre à une machine de pouvoir émuler les capacités cognitives de l'Homme, et de les surpasser. Ce terme "d'Intelligence Artificielle" voit le jour en 1956 après les nombreux travaux débutés post Seconde Guerre Mondiale et constitue l'un des plus récents champs d'études parmi les sciences et l'ingénierie. Cela fait suite aux nombreuses inventions au fil des siècles permettant à des machines de calculer que l'hypothèse qu'elles puissent penser et agir par elles-mêmes virent le jour également [4].

Thomas Hobbes parle notamment d'un "animal artificiel" où les organes seraient remplacés par des parties mécaniques. Cette discipline a permis aux grands esprits de notre époque de pouvoir formuler des idées, au même titre que les penseurs et inventeurs que la Terre a pu accueillir, puisqu'il s'agit d'un domaine universel qui touche à toutes sortes de tâches intellectuelles, et donc à des problématiques diverses, par exemple dans le domaine de la santé, de la sécurité, des arts, etc…

Alan Turing, père de l'informatique moderne, avait déjà établi un ensemble de conditions à réunir pour pouvoir définir une Intelligence Artificielle. Ce fameux Test de Turing qui fait encore office de nos jours comprend six éléments dans sa version complète : la possibilité pour la machine de pouvoir communiquer, ce qui nous a donné le domaine du traitement du langage naturel, la mémorisation des informations, le raisonnement par ces mêmes informations qu'elle a mémorisées, l'apprentissage, avec la détection des invariants et l'extrapolation sur une base de données plus restreintes, la perception et la reconnaissance des objets qui l'entourent, et enfin la manipulation de ces mêmes objets ainsi que son propre déplacement.

Dans le cadre du raisonnement et de la pensée et de la prise de décision, sujet sur lequel nous souhaitons développer notre réflexion, il est nécessaire de savoir que l'important n'est pas de reproduire à la perfection le cheminement de la pensée d'un être humain, mais plutôt d'appréhender les concepts sous-jacents afin de les appliquer pour mettre au point des intelligences artificielles qui ne nous méprennent pas.

L'exemple le plus concret est celui de la recherche de la capacité à pouvoir voler, nombreux sont ceux qui ont cherché à reproduire le mouvement des oiseaux à l'aide de créations qui les copiaient, sans grande réussite. C'est en s'écartant de cette vision, et en adaptant les concepts assimilés aux capacités de l'Homme que nous avons été en mesure de créer des machines pouvant parcourir le ciel.

La finalité n'était pas de nous transformer en oiseaux, mais de pouvoir nous déplacer dans les airs. Nous pensons ainsi que le principe est similaire dans le cadre de la réflexion d'une machine : le but n'est pas de reproduire à la perfection le fonctionnement du cerveau humain, mais plutôt d'adapter les mécanismes qui interviennent aux possibilités des machines que nous construisons.

Pour mieux comprendre la théorie de l'Intelligence Artificielle, il est nécessaire de se replonger dans les origines de cette dernière. Les premiers éléments remontent à Socrate, qui a ouvert le domaine de la logique amenant la tradition logiciste (mathématiques = extension de la logique) pour mettre en place des systèmes intelligents où il y instaura une qualité importante : l'Intelligence Artificielle est subordonnée aux lois de la pensée, c'est à dire que la validité des inférences est importante, mais que la capacité de prendre des décisions quand bien même on ne peut déterminer avec certitude laquelle est la meilleure l'est tout autant. Parfois, certaines décisions prises à la hâte sont plus efficaces que celles reposant sur un temps de réflexion, comme le fait de s'écarter d'un objet chaud lorsque notre corps rentre en contact avec ce dernier, il n'est pas nécessaire de dépenser du temps pour prendre la meilleure décision dans certains cas. Ainsi est venu le concept dit d'agent rationnel : il s'agirait dans un cas pratique d'une Intelligence Artificielle capable d'exécuter les mêmes tâches qu'un être humain tout en conservant un comportement "rationnel". Par rationnel, on entend qu'il serait capable de prendre des décisions lui permettant d'obtenir le plus de succès possible dans la réalisation des tâches que les hommes lui auraient assignées et qu'il tendrait donc vers la meilleure solution prévisible. L'économie a participé d'une certaine façon au développement de l'Intelligence Artificielle grâce à la théorie de la décision qui couple celle des probabilités et de l'utilité. Elle fournit un cadre formel complet pour les décisions en environnement incertain où les probabilités ont la possibilité d'influencer le preneur de décisions. Cela a mené à la création de modèles de prises de décision fondés sur le choix le plus satisfaisant, en somme celui qui se rapproche le plus du comportement humain : la maximisation du gain.

Cela pousse l'étude de la balance vitale de l'Intelligence Artificielle : celle du lien entre la connaissance et l'action. Il n'est a priori pas possible de prendre, le plus souvent, les

meilleures décisions sans avoir vécu ou mémorisé les actions entreprises et leurs conséquences. Toute cette partie de l'apprentissage constitue un sous-ensemble pour l'Intelligence Artificielle qui est le "Machine Learning", ou "apprentissage automatique". Ce champ d'études a pour but d'améliorer les performances des machines à résoudre des tâches de façon généralisée : on ne cherche pas à faire du par cœur, mais à permettre à une machine de pouvoir donner une réponse à un problème grâce à ce qu'elle a appris auparavant par des exemples qu'elle applique sur le cas qu'on lui donne.

Dans cette recherche de la réponse, selon Aristote, deux façons de procéder se posent à nous : pour un problème donné il peut y avoir plusieurs solutions possibles, et dans ce cas nous cherchons constamment à sélectionner celle qui est à la fois la plus facilement applicable, mais également la meilleure. Ou alors il n'y a qu'une seule solution qui ressort pour un problème donné, et dans ce cas précis la démarche consiste à remonter à la cause première afin de déterminer si cette solution est dans un premier temps la seule, et si ce n'est pas le cas si elle est la meilleure. Dans ce système de planification par régression imaginé à l'Antiquité, ce que l'on trouve en dernier dans l'analyse du problème est le premier élément qui débute la réalisation.

Pour W. Ross Ashby dans « Design for à Brain », 'on pourrait créer l'intelligence par l'utilisation de divers dispositifs permettant de maintenir un ensemble de facteurs clés et contenant des boucles de rétroaction qui permettent de garantir un comportement adaptatif stable. Pour faire simple on pourrait comparer son idée au "recuit simulé", un algorithme d'optimisation qui consiste à mettre en œuvre les moyens nécessaires pour trouver un minimum qui ne soit pas juste un minimum local.

Si l'on adhère à cette idée, la création d'une Intelligence Artificielle devient la conception d'un système au comportement optimal et dans ce cas, pourquoi l'IA et la théorie de la commande optimale, qui est une théorie visant à déterminer la commande qui minimise ou maximise un critère de performance avec potentiellement des contraintes, forment-elles deux domaines différents malgré de nombreux points communs ? De fortes similitudes existent entre les techniques mathématiques qui leur étaient familières et les différents types de problèmes abordés par ces deux domaines. Les outils de la théorie de la commande sont plus axés pour les systèmes qui se décrivent sous forme d'ensembles fixes de variables, alors que l'Intelligence Artificielle a été conçue en partie pour échapper aux limites de ces outils. Cela permet ainsi la possibilité d'étudier différents problèmes comme le langage et la planification qui tombaient hors du champ de la théorie de la commande optimale.

L'histoire de l'Intelligence Artificielle est une alternance de progrès très médiatique mais aussi de très grosses déceptions. Là nous sommes de nouveau en phase de boom.

Nous serons tentés de nous demander « mais nos ordinateurs sont-ils intelligents ou pas ? ». Pour cela il faudrait clairement comprendre et définir ce qui est l'intelligence biologique pour pouvoir définir, en regard ce qui est l'Intelligence Artificielle. Avant être un excellent joueur d'échecs était une manifestation d'intelligence, aujourd'hui ça ne suffit plus puisque la machine le fait mécaniquement. En fait, la définition de l'intelligence biologique suit l'évolution de l'Intelligence Artificielle, et c'est passionnant comme question.

1.2.2 Intelligence biologique

En même temps que l'Intelligence Artificielle sont nées « les sciences cognitives », qui étudient l'intelligence humaine. Elles ont permis d'identifier les différents mécanismes qui sont à l'œuvre dans le fonctionnement de la pensée. Cette pensée qui nous permet de percevoir, de parler, de bouger, mémoriser, planifier, de faire preuve d'abstraction parfois ou de créativité pour les meilleurs d'entre nous : Ce sont nos fonctions cognitives.

Pour simuler l'intelligence humaine, l'idée était de reproduire les fonctions cognitives, pour y parvenir les chercheurs ont privilégié deux approches qui sont un peu à l'image de notre façon d'apprendre. Soit par la transmission de connaissance, soit par tâtonnement à partir de l'observation du monde et de l'expérimentation, les deux sont compatibles.

- **L'approche symbolique**

Elle essaie d'imiter comment l'humain raisonne logiquement et utilise ses connaissances. Comme nous l'explique THALITA FIRMO DRUMOND Docteur INRIA, un certain nombre de savoirs nous sont transmis par un ensemble de règles, de procédures. C'est la manière privilégiée par le savoir scolaires.

Comme apprendre à poser une addition par exemple, lorsque je pose une addition je peux expliquer comment je m'y prends étape par étape, il est donc ensuite possible d'écrire un programme expliquant à la machine comment procéder étape par étape, la machine pourra ensuite l'exécuter plus rapidement que moi sans étourderie. Cette approche a donné les systèmes experts, qui essaient de reproduire le raisonnement d'un spécialiste comme un médecin ou un juriste pour résoudre un problème. C'est cette approche-là qui a eu le vent en poupe dans les années 80

- **L’approche Numérique**

D’autres savoirs naissent de l’expérience pour lesquels il est beaucoup plus difficile d’expliquer comment on s’y prend comme par exemple, faire du vélo ou reconnaitre un chat. On aura beau vous expliquer comment faire du vélo mais c’est en pédalant que vous saurez pédaler. Il faut que vous en fassiez l’expérience.

Ça c’est l’autre approche (l’approche numérique ou connexionniste) qui elle tente d’imiter comment l’humain apprend par essai et erreur à partir de ses expériences.

C’est cette approche-là qui est sur le devant de la scène aujourd’hui « l’apprentissage machine et les fameux réseaux de neurones ».

Fréderic Alexandre, qui est chercheur en neurosciences computationnelles nous donne les éléments de réponses suivants :

Être intelligent c’est savoir répondre rapidement à une situation complexe. Être intelligent c’est trouver la réponse satisfaisante plutôt que la réponse optimale. Être intelligent c’est savoir trouver ce qui est important dans une information. Être intelligent c’est savoir s’adapter si les conditions changent où réagir à une situation imprévue et tout ça en utilisant une stratégie caractéristique du vivant qui mélange émotion, motivation, créativité. La force et la spécificité de l’intelligence biologique que n’a pas l’Intelligence Artificielle, c’est la complémentarité de ses aspects émotionnels et logiques (le bon sens).

En fait pour éviter les fantasmes véhiculés par l’intelligence générale, dont on est loin il vaudrait mieux parler « d’intelligences artificielles ». Toutes les intelligences artificielles que nous croisons dans la vraie vie ne sont juste « techniques » et spécifiques. Elles ne peuvent faire que la tâche pour laquelle elles ont été programmées ou paramétrées.

Un programme qui bat le champion du monde d’échecs ne sait pas reconnaitre un canard. En revanche pour une tache précise elles sont ultra performantes et souvent meilleures que nous.

1.3 Les Systèmes Graphiques

Un moteur graphique est un logiciel, la plupart du temps très puissant qui a la faculté de permettre la création d'éléments graphiques 3D et jeux vidéo. Sans moteur graphique il serait donc impossible de créer un jeu vidéo. Un moteur graphique n'a cependant aucun lien avec un « moteur » à proprement dit. Ce n'est pas une machine qui fait du bruit et qui produit de l'énergie.

Il existe cependant différents moteurs graphiques qui permettent toute sortes de créations. Cela va d'un simple jeu en HTML 5 tel qu'une copie de Flappy Bird jusqu'à Crisis 3 ou FIFA 22.

Un moteur graphique est cependant très difficile à faire tourner sur un PC et même sur un Mac d'entrée de gamme. Très compliqué voire impossible de faire fonctionner Unreal Engine 4 sur un MacBook Pro doté de 4GO de mémoire vive avec le processeur Intel I5 cadencé à 2.5 Giga Hertz avec une carte graphique Intel HD graphics 4000.
Il vous faudra au minimum 8GO de RAM pour faire fonctionner ce type de logiciel accessible à tous mais qui a tout de même servi à la création du jeux mythique FORTNITE.

1.3.1 Un petit aperçu des moteurs graphiques

- Unreal Engine => qui a servi à faire Unreal Tournament 3
- RPG Maker => Idéal pour faire des copies de Pokémon
- Cryegine => Le moteur utilisé pour réaliser Crysis
- Unity 3D => Rien à voir avec AC Unity mais très accessible
- Source de valve => bon pour les mods
- Game Maker => qui a servi à faire Hotline Miami
- Frosbite 3 => Moteur graphique de BF4
- Source Engine => Titanfall

L'objectif des moteurs graphiques est de rendre l'expérience la plus immersive et le plus réel possible. Le moteur est une notion vague du jeu vidéo pas très évidente a cerné. A plus forte raison maintenant avec l'évolution des machines et des programmes. Là où la relation de dépendance commence et où elle ce fini n'est pas facile à voir, votre ordinateur n'est qu'un tas de (0 et 1) sans structure n'y chef d'orchestre il n'y a rien qui marcherait dans votre ordinateur. Le moteur agit alors comme un chef de haut niveau puisqu'il

s'occupe de la gestion des fichiers du jeu, mais il dépend lui-même de plus petits chef (pilotes, Librairies…) qui eux aussi dépend d'autre chef d'orchestre encore plus petits.

Tout ceci formant ce que l'on appelle le Framework. Le Framework qui nous servira d'exemples pour définir le moteur, a pour rôle de nous aider à constituer un objet complexe, un programme en l'occurrence.

On peut définir un Framework par son habilité a facilité une tache mettant en place certain automation logique. Le moteur est un genre de Framework de haut niveau alloué à la gestion de données d'un jeu pour éviter la multiplication de taches redondantes à codées, aussi bien dans la création que pour la lecture des données.

Avant l'invention de ceci les jeux étaient codés d'un bloc, il fallait tout faire de A à Z et on ne pouvait pas réutiliser son code pour un autre jeu en moins d'en faire un clone. Ce qui veut dire qu'à chaque nouvelle idée de jeu il fallait repenser entièrement le système d'affichage, la gestion des inputs le son …

Non pas que l'on n'en avait pas envie à l'époque d'avoir des outils clé en main pour concevoir des jeux, les mémoires des machines de l'époque étaient beaucoup trop faibles pour supporter la structure de données complexe qu'un moteur nécessite. Avec l'âge d'or du jeu vidéo les premiers moteurs de jeux commence a pointé le bout de leur nez au siens des compagnies de jeux vidéo étant réserver qu'à leur propre usage. Ce n'est pas ces moteurs qui nous intéressent ici, il y a peu d'informations à leur sujet puisqu'il reste bien souvent à la discrétion de l'entreprise l'ayant conçu. Il existait bien dans les années 80 des systèmes de créations de jeux ouverts au grand publique mais ceux si ne faisaient rien de mirobolant car ils n'offraient pas la même liberté d'action que les moteurs de jeux plus modernes. Ce qui va nous intéresser ici sera les moteurs qui furent ouvert au public pour la création de jeu non spécifique à un type de gameplay.

1.4 Conclusion

Dans ce chapitre il était question pour nous de mettre en lumière la notion d'Intelligence Artificielle et l'utilité des systèmes graphique, grâce à l'étude mené et à nos recherches nous pouvons dire que l'industrie des moteurs graphiques on contribuer de prêt ou de loin a facilité l'étude dans le domaine de l'Intelligence Artificielle. Et transportant cela sur un nouveau terrain qui est la modélisation visuel

2 CHAPITRE 2 : Présentation détailler du moteur graphique Unreal Engine 4 Et de son système de programmation intégré

2.1 Introduction

Développer par le studio Epic Game, Unreal Engine est l'un des moteurs les plus populaires dans le monde de la création et du montage graphique. Fondée en 1991 par le programmeur Tim Sweeney, Epic Games est au départ une jeune startup dont la stratégie marketing est entièrement basée sur le principe du shareware. Fin 1994, James Schmalz crée un prototype rudimentaire de jeu de flipper en trois dimensions afin de développer ses champs de compétences. Fan inconditionnel de Doom, il décide de s'en inspirer et commence à écrire un programme capable d'afficher des textures et de les étirer de façon à simuler un effet 3D.

Début 1995, il parvient finalement à créer un moteur 3D fonctionnel dont il fait une première présentation en créant un niveau représentant un château médiéval. Sweeney se montre alors très intéressé par son projet et se joint à lui pour la programmation du moteur. Il prend notamment l'initiative de créer un éditeur de niveau, afin de simplifier le processus de création des niveaux, d'implémenter un moteur physique et d'optimiser le système de rendu.

À cette époque, Doom représente la référence du jeu de tir à la première personne. Grand admirateur de John Carmack pour sa contribution à la création des moteurs 3D de Wolfenstein 3D et Doom, Tim Sweeney a l'ambition de concurrencer ce grand nom du jeu vidéo.

Il décide donc de prendre en main la programmation du moteur graphique et de le pousser au maximum de ses capacités. Dans cette optique, le moteur est modifié afin de gérer une profondeur des couleurs de 16-bit (65 536 couleurs, appelées « couleurs vraies ») en lieu et place des graphismes 8-bit usuels supportés par les standards VGA et SVGA (qui n'affichaient que 256 couleurs). Après 18 mois de travail acharné et sept refontes du code source, il parvient enfin à obtenir une version stable et performante du moteur. Réalisant les possibilités offertes par cette technologie, Schmalz et Sweeney décident dès 1994 de l'utiliser pour créer un jeu vidéo qu'ils baptisent Unreal et qui donnera son nom au moteur de jeu. Après la publication d'une première version de démonstration de la technologie utilisée pour celui-ci en 1995, Unreal est publié par GT Interactive le 22 mai 1998, devenant ainsi une vitrine pour le moteur développé par Epic Games.

Lorsqu'Unreal sort en mai 1998, le jeu est plutôt bien accueilli par la presse spécialisée et les joueurs, mais sa réputation est peu à peu ternie par les lacunes de son mode réseau qui rebutent les joueurs expérimentés. Conscient de ces lacunes, les développeurs se focalisent donc sur la correction des problèmes entachant le mode multijoueur. Epic Games commence alors à réfléchir à une extension officielle destinée à améliorer le code réseau du jeu et son gameplay en multijoueur. Dans ce cadre, Tim Sweeney et Steve Polge se focalisent sur l'amélioration du code réseau et du moteur de jeu de l'UnrealEngine 1.

Les délais étant moins contraignants que ceux imposés pour Unreal, les programmeurs disposent de suffisamment de temps pour exploiter pleinement le potentiel du moteur de jeu. Ils peuvent ainsi utiliser certaines caractéristiques du moteur qu'ils n'avaient encore jamais testées, ce qui leur permet notamment d'améliorer sensiblement le rendu graphique des personnages et des niveaux, permettant par exemple de multiplier par quatre le niveau de détail des apparences disponibles pour les personnages, notamment au niveau des visages. Devant l'ampleur de la tâche, Epic Games décident finalement de faire de cette extension un jeu à part entière qu'ils baptisent Unreal Tournament. L'Unreal Engine est programmé en C++ et dans un langage de script, baptisé « UnrealScript », utilisé pour 90 % du code lié au gameplay d'Unreal Tournament.

Le moteur utilise une conception orientée objet, ce qui lui permet d'être extrêmement modulaire. Les interfaces des différentes sous-parties du code étant clairement définies, les développeurs peuvent travailler sur l'une d'entre elles sans que cela n'affecte le reste du jeu. Cela a également permis d'isoler le code lié aux plates-formes dans des bibliothèques logicielles séparées, facilitant le portage du jeu sur d'autres plates-formes. Enfin, cette modularité se retrouve dans le langage de script du moteur de jeu. Tous les éléments du jeu, comme les armes ou les objets, sont définis de manière indépendante. Les programmeurs peuvent donc ajouter ou modifier ces éléments sans que cela n'affecte le code source.

Cette conception orientée objet couplée avec l'éditeur de niveau UnrealEd fait de l'Unreal Engine un système performant pour créer de nouveaux jeux ou des modifications de jeux l'utilisant. Une grande partie du code a en effet été conçue de manière à rendre le jeu attractif pour les programmeurs et les artistes de la communauté des moddeurs. L'Unreal Engine est axé sur la CSG (la géométrie de construction de solides) soustractive à l'inverse du quake engine, basé sur la géométrie additive. En résumé, pour construire un univers dans le quake engine, les éléments sont ajoutés au vide (un peu comme l'on construirait une maison en ajoutant des briques). Dans l'Unreal engine, le monde est plein au départ et le designer creuse dans celui-ci pour créer son univers.

Cela offre la possibilité au designer de l'unreal engine d'utiliser des "portals" pour cloisonner son monde en zones fermées bien distinctes et ainsi offre au moteur la possibilité grâce aux PVS (potential visibility sets) de n'afficher que la partie visible d'un univers. Ainsi, le moteur d'unreal pouvait offrir des intérieurs complexes et des extérieurs riches tout en offrant des performances impressionnantes pour l'époque. Pour le « ciel », Epic adopta le système des Skybox : procédé aujourd'hui courant pour représenter le ciel en 3D temps réel. Comme id Software avec le Quake engine, Epic Games met ensuite l'Unreal Engine à disposition sous licence. Particulièrement populaire dans la communauté des développeurs de mods, celui-ci devient rapidement le principal concurrent du Quake Engine. En plus des titres développés par Epic Games et Digital Extreme, celui-ci a ainsi été utilisé par de nombreux studios de développement pour créer des jeux tels que Deus Ex, Rune ou encore Star Trek: Deep Space Nine: The Fallen Plusieurs versions du moteur Unreal Engine se sont succédées suite aux améliorations constantes et perpétuels, depuis la création de la version UE1 jusqu'aujourd'hui a la version UE5. Mais cette dernière version « Unreal Engine 5 » n'étant pas suffisamment stable notre projet d'étude de fin de mémoire se portera sur l'Unreal Engine 4 « version 4.26 ».

2.2 Unreal Engine 4

En août 2005, alors qu'aucun jeu utilisant Unreal Engine 3 n'était encore sorti, Ken Beaulieu, vice-président d'Epic Games, annonce que l'équipe travaille depuis deux ans sur la version 4, sans donner plus de détails. Ce moteur graphique cible la huitième génération de consoles. La seule personne à travailler sur le code source de l'Unreal Engine 4 est Tim Sweeney, directeur technique et fondateur d'Epic Games. Michael Capps, président d'Epic Games, a indiqué que le moteur devrait être prêt vers 2012.

Durant l'annonce de la PlayStation 4, Sony a présenté des enregistrements de jeux utilisant l'Unreal Engine 4. Le 19 mars 2014, Epic Games annonce que son moteur sera disponible pour tous au prix de 19 $ par mois, ainsi que 5 % de royalties.

Depuis le 2 mars 2015, Unreal Engine 4 est gratuit, les développeurs l'utilisant pour développer des jeux ou vidéos commerciaux devant toutefois payer une redevance à partir d'un certain chiffre d'affaires21. À titre de comparaison, la société exigeait une redevance de 25 % pour Unreal Engine 3 [9]. L'Unreal Engine 4 permet de créer pour PS4, PS5, Xbox One, Xbox Series, Nintendo Switch, iOS, Windows, Steam, Linux, HTML 5, PlayStation VR, Android, Oculus, Gear VR, Steam VR, Magic leap et Google VR. Il propose également le C++, et un nouveau langage de programmation graphique appelé Blueprint.

2.2.1 Le Blueprint

Etymologiquement Le terme blueprint désigne, en anglais, une reproduction d'un plan détaillé, ce que l'on appelle en dessin technique un dessin de définition. Le terme, signifiant littéralement « impression en bleu », provient d'un procédé d'imprimerie, la cyanotypie, puis la diazographie.

Ce terme, en tant qu'anglicisme, est utilisé dans certains domaines pour désigner une représentation spatiale d'un objet, selon un ou plusieurs points de vue définis par les standards du dessin industriel ou du dessin architectural (vue de face, de droite, de gauche, de dessus, de dessous, de derrière). Les Blueprints sont utilisés, entre autres, par les modeleurs tridimensionnels, afin de sculpter à partir d'une référence, pour respecter l'échelle dudit objet.

Un langage de programmation graphique ou visuel est un langage de programmation dans lequel les programmes sont écrits par assemblage d'éléments graphiques. Sa syntaxe concrète est composée de symboles graphiques et de textes, qui sont disposés spatialement pour former des programmes. De nombreux langages visuels se basent sur les notions « de boîtes et de flèches » : les boîtes (ou d'autres d'objets) sont traitées comme des entités, reliées par des flèches ou des lignes qui représentent des relations.

Plus précisément, un langage est défini par une syntaxe abstraite, à laquelle sont associées une ou plusieurs syntaxes concrètes, parmi lesquelles une ou plusieurs peuvent être graphiques.

Généralement ces langages sont associés à un environnement graphique de programmation. Il n'est pas toujours possible de les dissocier. Il faut également faire la distinction entre le langage au sens "normalisé" et son implémentation au sens "logiciel".

Pour le côté simplicité les Blueprint de façon évidente sont beaucoup plus simples pour le débutant parce qu'il n'y a pas de syntaxe de langage à connaitre, il n'y a pas d'erreur possible sur la façon de déclarer les variable, les fonctions etc. C'est beaucoup plus facile de s'approprier quelque chose qui est graphique qui marche avec des nœuds. Si on reste dans la logique de la programmation quand on fait une boucle en Blueprint ça reste exactement la même logique que celle qu'on emploierait pour un programme en C++ mais c'est guidé et beaucoup plus simple

Les erreurs sont beaucoup faciles à corriger dans un Blueprint que dans un C++, déjà ils sont plus difficiles à commettre puisqu'il n'y a pas la possibilité d'utiliser des pointeurs et autre indice de ponctuations spécifiques aux autres langages.

Et de plus on a un débogueur visuel ou on peut voir au fur et à mesure que se déroule l'action et exactement par quel nœud ça passe.

Donc de façon générale les Blueprints sont plus faciles à modifier, à bidouiller, on changera plus facilement une valeur de variable etc. Il n'y a pas ce côté compilation à chaque fois, en plus en cours d'exécution on peut changer des valeurs ce que l'on ne fera pas avec le C++.

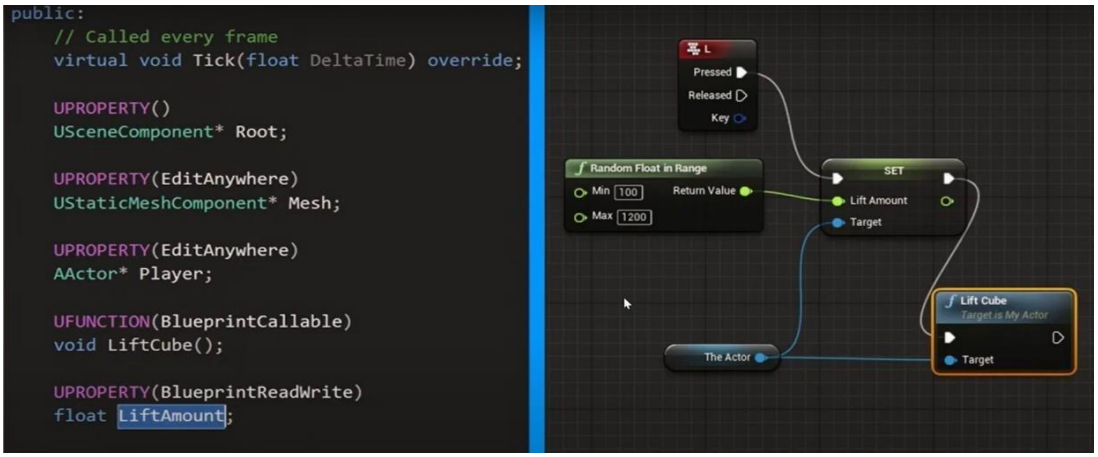

Figure 1 : Programme en C++ Vs Blueprint

❖ **Les Différentes Classes de Blueprints**

Unreal Engine va mettre à votre disposition plusieurs types d'entités qui possèdent plus au moins de fonctionnalités. Certaines pourront être utilisée comme des points de spawn, d'autres comme de simples décors alors que d'autres seront des ennemis ou le joueur lui-même.

L'ensembles des objets qui vont être créer sur Unreal héritent tous de l'une des classes que je vais vous présenter ici tel que [5] :

La classe Actor : C'est l'entité de base, elle est équivalente au GameObject dans Unity. En réalité c'est un nœud vide sur la scène qui ne possède qu'un composant gérant ses transformations

Le Pawn : Il hérite d'Actor, mais peut être contrôlé par un joueur et peut recevoir des événements input.

Character : C'est un Pawn amélioré qui peut marcher et sauter ! C'est ce que nous allons utiliser très prochainement !

Player Character : : C'est un Character avec plus de fonctionnalités (notamment pour la gestion en réseau).

GameMode : c'est lui qui va gérer justement les règles du jeu. Nous en reparlerons de lui plus en profondeur au cours de Mise En Pratique.

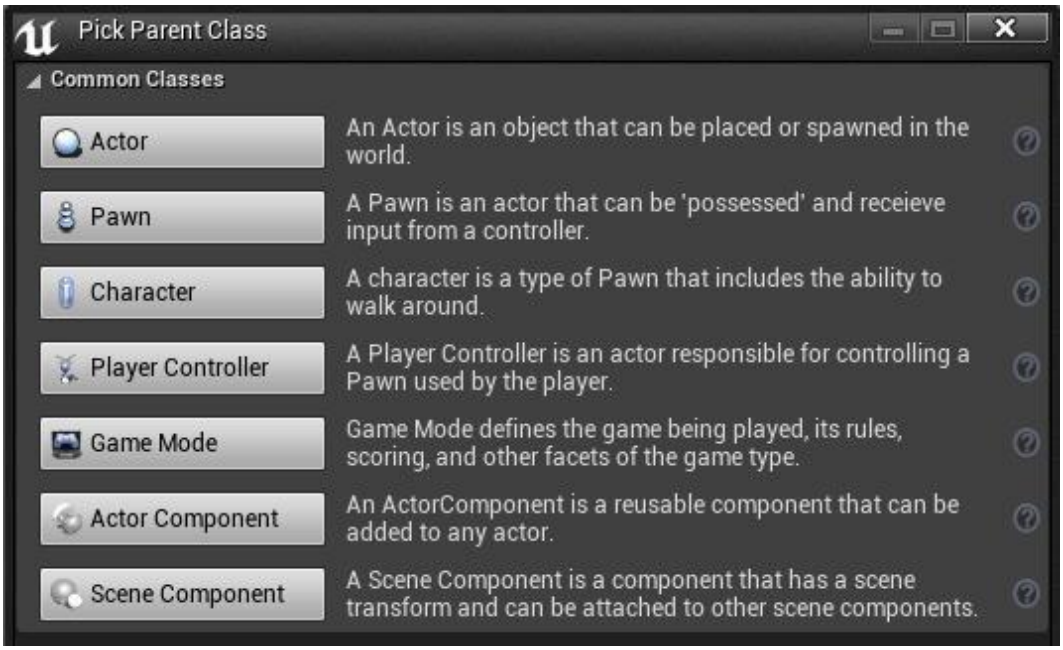

***Figure** 2 : Les classes parents*

2.2.2 Présentation détailler du moteur UE4

Apres le téléchargement et l'installation complète du moteur Unreal Engine lançons le logiciel. On se retrouve alors à la page d'accueil.

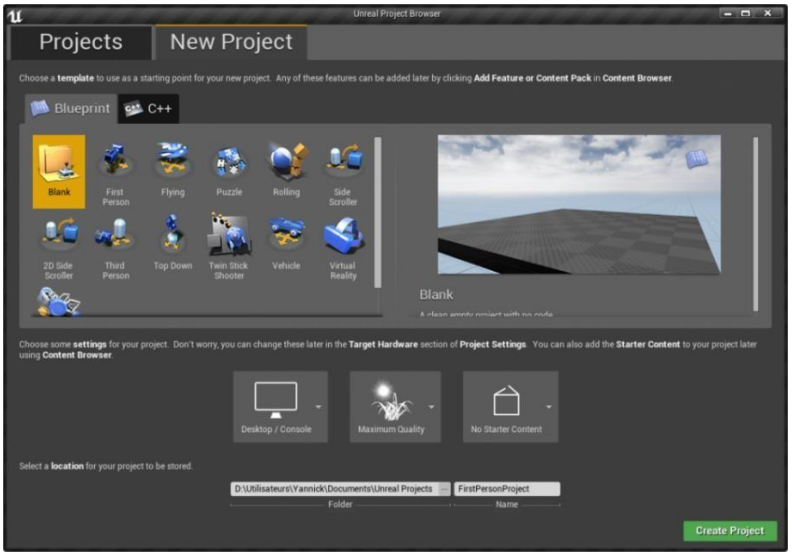

Figure 3 : Unreal Project Browser

Dans cette fenêtre il est question de Choisi le langage de programmation a utilisé dans la création du futur projet, Choisi le lieu de stockage du Projet et le NOM a assigné au projet, Choisi la plateforme sur laquelle tournera le jeu, choix si oui ou non on veut commencer le projet avec ou sans Starter Content.

- **L'éditeur Unreal**

Si vous utilisez déjà Unity, vous verrez qu'il y a beaucoup de similitudes !

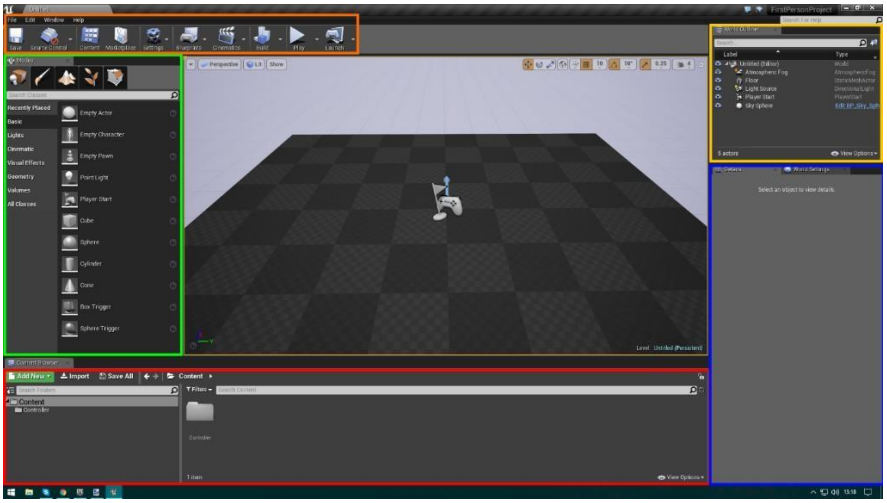

Figure 4 : Editeur Unreal

La zone orangée représente la barre de menu et la barre d'action. Vient ensuite la hiérarchie de votre scène en jaune. Les panneaux dans le carré bleu permettent de configurer les entités de votre scène et de régler les paramètres du monde. Unreal Editor dispose comme tout bon moteur qui se respecte, d'une sélection d'objets et d'outils pour construire votre scène. Lumières, primitives, éditeur de terrain, éditeur de maillage, etc…

Enfin, la zone en rouge représente le Content Browser (le gestionnaire d'assets), c'est là où vous glisserez vos images, modèles et sons. Les assets importés sont automatiquement copiés et convertis au format.uasset, il n'est pas possible de les modifier après cette étape. Vous devez donc conserver vos assets de base sur votre disque. Maintenant que vous êtes un peu plus familiarisé avec l'éditeur, nous allons pouvoir passer à la suite, je veux bien évidemment parler du réglage des entrées utilisateur !

Vous voyez la partie du milieu où il la surface noir ? Cette fenêtre se nomme le Viewport. C'est dans cette fenêtre que nous pourrons contempler l'évolution de notre projet, c'est la zone la plus importante.

❖ Character et Game Mode : Un Duo incontournable !

Unreal Engine va mettre à votre disposition plusieurs types d'entités qui possèdent plus au moins de fonctionnalités. Certaines pourront être utilisée comme des points de spawn, d'autres comme de simples décors alors que d'autres seront des ennemis ou le joueur lui-même.

Dans le Content Browser (le gestionnaire d'assets), vous pouvez constater qu'il y a un dossier Content. Cliquez dessus et créez un dossier à l'intérieur que vous nommerez Controller. De là, dans la zone centrale du Content Browser, faites un clic droit, cela ouvrira un menu contextuel. Il faut choisir Blueprint, puis Character. Nommez le nouveau fichier : CharacterBP.

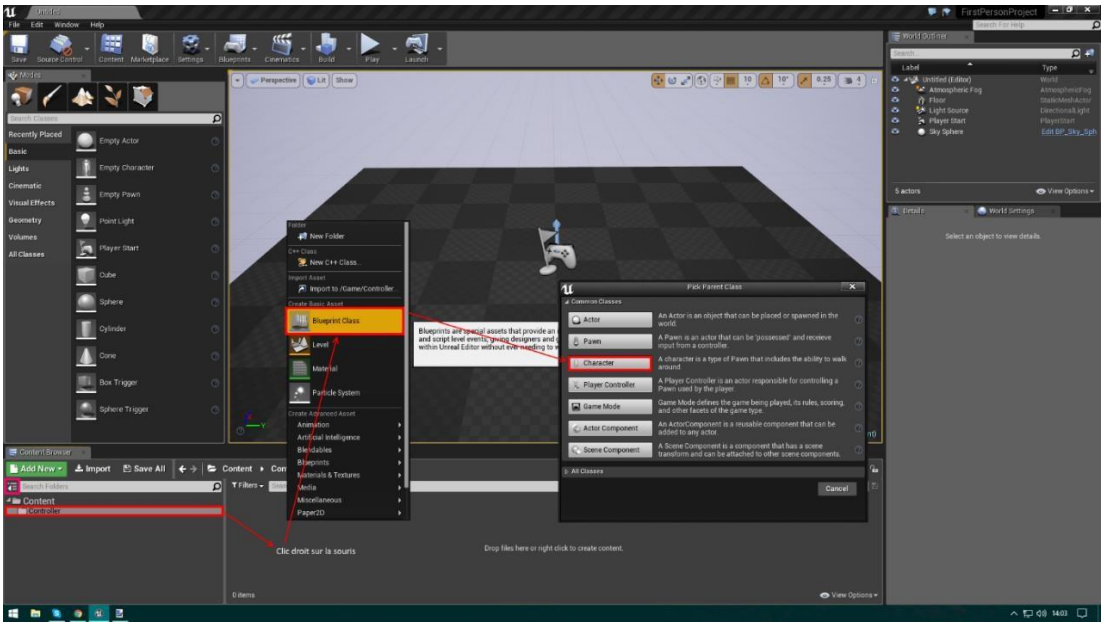

Figure 5 : Création d'un Blueprint

Recommencez la même opération mais cette fois-ci, au lieu de choisir Character, choisissez GameMode et nommez le MyGameMode. Cet objet un peu particulier va définir les règles de votre jeu. Vous allez par exemple pouvoir définir le nombre de joueurs, si le jeu peut être mis en pause ou pas, si c'est une cinématique, le gestionnaire d'interface utilisateur, etc… Un jeu complet aura plusieurs GameMode, un qui définira le fonctionnement de vos scènes cinématiques, un pour les menus et enfin un autre pour les phases de jeu par exemple.

Avant d'aller plus loin, je vous propose de sauvegarder votre Map (l'équivalent de la Scène avec Unity), même si cette dernière est vide. On commence par créer un dossier Maps à côté du dossier Controller. Puis dans le menu File / Save As. Enregistrez la scène dans le dossier Maps sous le nom MyMap (Original non ?).

❖ Game Mode : La pratique

Vous pouvez désormais configurer votre jeu via une pléthore de paramètres. Nous n'en changerons qu'un seul aujourd'hui, celui de la classe du joueur à utiliser par défaut. A droite, dans la catégorie Classes, il faut changer la valeur de Default Player Class, par celle que vous avez créé tout à l'heure, à savoir CharacterBP. Première révélation, un Blueprint est une classe !

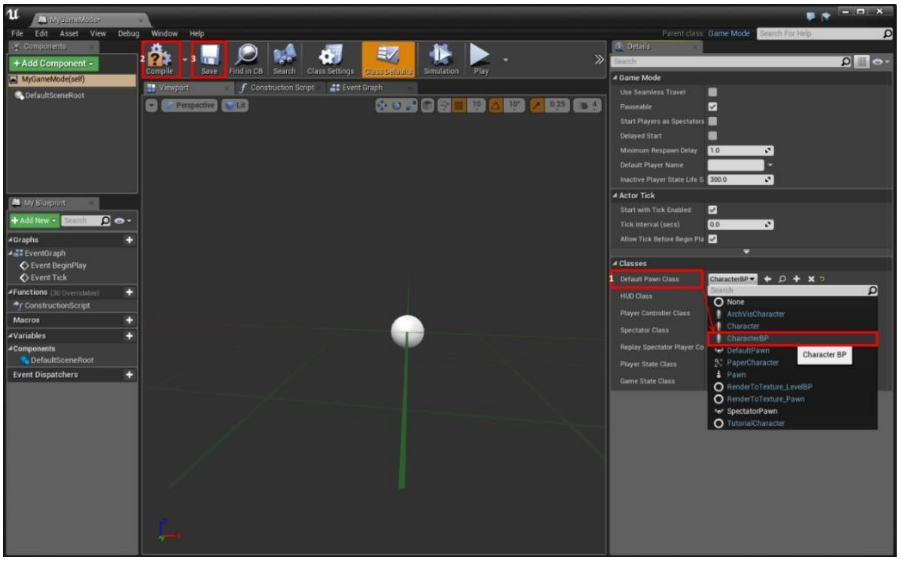

***Figure 6** : Viewport d'un Blueprint*

Pour ce petit exemple rapide nous allons utiliser ce GameMode par défaut (car aujourd'hui nous n'avons qu'une seule scène et que c'est simple de faire comme ça. Et toc !). Rendez-vous dans le menu Edit/ Project Settings. Dans la colonne de gauche, dans la partie Project, il faut choisir Maps & Modes. De là vous pouvez changer la valeur de Default GameMode et choisir celui que vous venez de créer, à savoir MyGameMode. Vous pouvez aussi, si vous le désirez, choisir la scène que vous avez sauvé juste avant en tant que scène principale.

2.3 Premier pas en scripting visuel

Nous avons créé deux Blueprint tout à l'heure, ce sont des classes (comme en C++) avec des variables, des méthodes, mais au lieu d'être représenté sous forme de code, un Blueprint est représenté de manière visuelle. Cela a ses avantages et ses inconvénients. Un bon paquet de fonctionnalités C++ sont accessibles via Blueprint, cela peut vous permettre par exemple de créer un jeu entier (sans gameplay complexe) sans coder une seule ligne de C++ ! Rassurez-vous, vous pouvez aussi créer vos Blueprint en C++, mais ça c'est une autre histoire.

Commencez par ouvrir le fichier CharacterBP (qui pour rappel est le joueur) en double cliquant dessus, une fenêtre s'ouvrira...

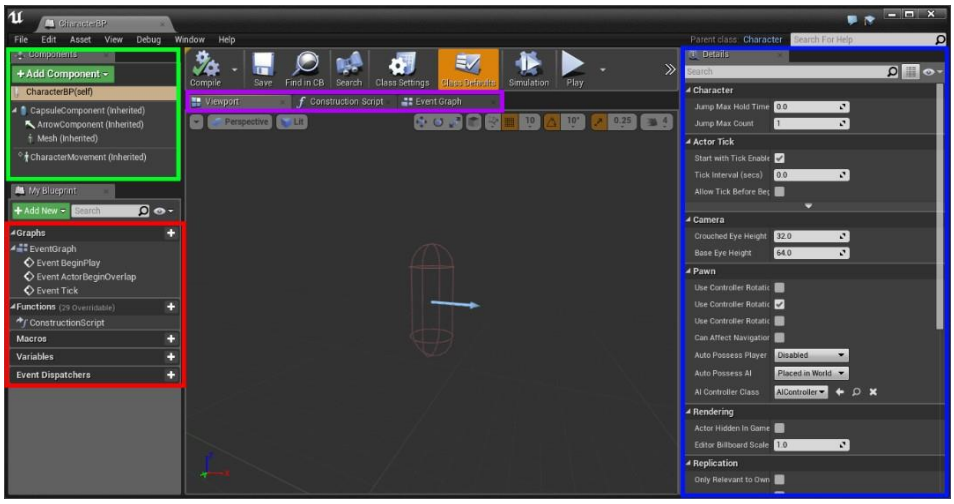

Figure 7 : Éditeur Blueprint complet

Cet éditeur va vous permettre de faire beaucoup de choses, c'est pour ça qu'il est un peu chargé Nous allons donc le découvrir petit à petit. Rappelez-vous juste que cette première vue va vous servir à paramétrer la partie visuelle de votre entité en lui ajoutant par exemple des composants (Rectangle vert). Qui dit Blueprint, dit fonctions, variables… Et bien dans le panneau représenté par le rectangle rouge, vous pouvez définir des variables, des fonctions et des événements. Le panneau en bleu est un inspecteur, tout ce qu'il y a de plus classique. Enfin les onglets vous permettront dans changer de mode (Conception, Construction, Event). Vous remarquerez qu'il n'y a pas de caméra et pourtant si vous essayez le jeu tout de suite, vous verrez qu'une caméra est bien présente. Elle est intégrée automatiquement, il n'est pas obligatoire de la rajouter. Cependant vous pouvez le faire si vous avez des choses spécifiques à faire dessus.

Passons tout de suite au vif du sujet, le scripting visuel, avec l'onglet Event Graph ! Vous pouvez déjà commencer par supprimer les trois blocs présents devant vous (touche Supprimer). Pour poser un nouveau nœud, il suffit de faire un clic droit, de là un menu contextuel s'ouvre et vous permet de rechercher ou sélectionner la fonction requise.

2.3.1 Votre premier Blueprint

- **Avancer, reculer, droite et gauche**

La première chose à faire est de récupérer l'événement MoveForward pour avancer ou reculer le joueur lorsque les touches Z/D sont appuyées, ou l'axe Y du stick droit du Gamepad est pressé. Pour cela faites un clic droit pour afficher le menu contextuel et tapez MoveForward. Vous avez deux options possibles, soit la valeur de l'axe, soit l'événement. Nous prenons l'événement (Axis Events).

A partir de cet événement on peut très facilement dire que si l'axe à une valeur, alors il faut l'utiliser pour faire bouger le joueur. Pour cela on va utiliser un nœud **AddMovementInput,** avec les liaisons suivantes.

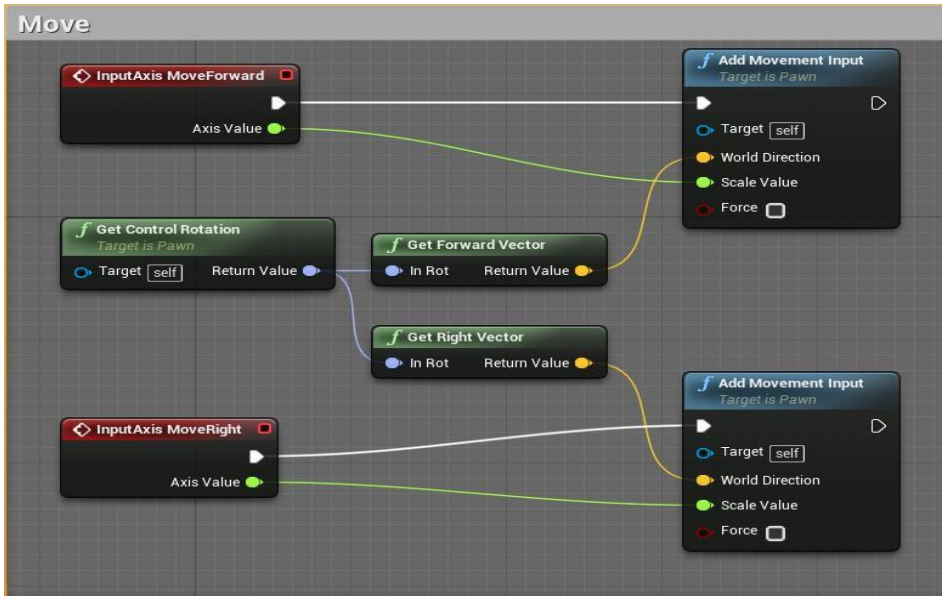

Figure 8 : Deplacement du personnage

Voilà ce que donne le graphe complet pour le mouvement avant / arrière et droite / gauche. Je vous encourage à encadrer vos graphes dans des boites commentaire et de les nommer, cela sera plus facile par la suite pour vous relire. Pour créer un commentaire, il suffit de sélectionner les nœuds que vous voulez, puis faire un clic droit dans la zone de titre d'un nœud et choisir Create Comment From Sélection.

- **Rotation verticale et horizontal**

Cette partie est encore plus simple car nous n'avons qu'à appliquer deux rotations ! Vous allez utiliser les événements Turn pour la rotation horizontal (Yaw en Anglais) et LookUp pour la rotation vertical (Pitch en Anglais). Pour appliquer une rotation on utilise un nœud AddController [Yaw|Pitch|Roll]Input, avec uniquement un paramètre Val !

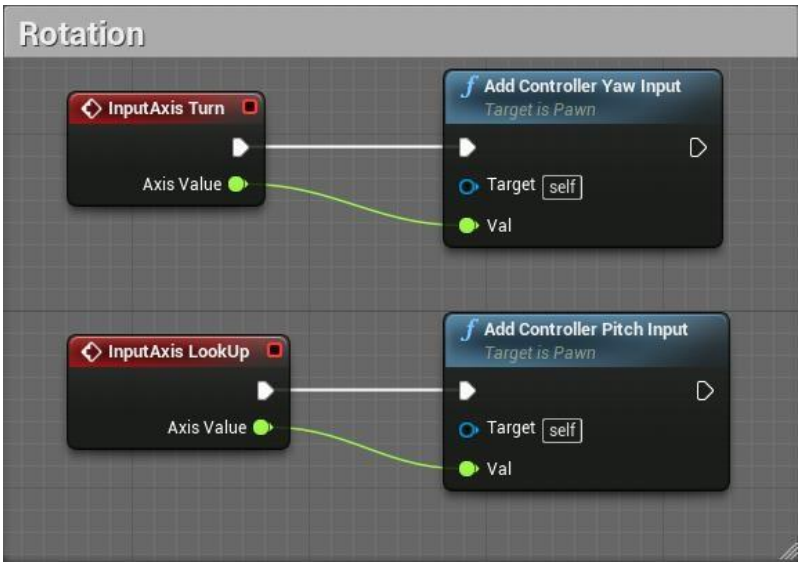

Figure 9 : Rotation du Personnage

❖ **Le saut**

Vous vous rappelez sans doute que nous utilisons un Blueprint dérivant de la classe Character, ce dernier possédant tout un tas de fonctions relatives au joueur. Et bien figurez-vous qu'il y a une fonction Jump et StopJump.

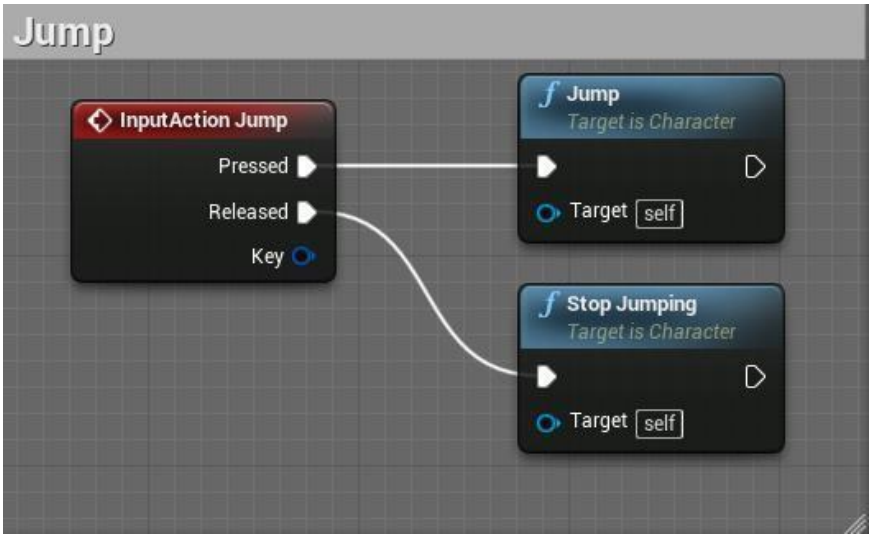

Figure 10 : Logique de Jump

Facile non ? Lorsque le bouton Jump est pressé, on active la commande de saut et quand il est relâché et bien... On arrête tout. Compilez et sauvegardez votre travail avant de tester pour valider le fonctionnement. La touche espace permet désormais de sauter !

Cet exemple vous a permis de créer votre propre contrôleur à la première personne qui gère automatiquement les collisions et peut aussi sauter. Vous pouvez essayer de l'améliorer en ajoutant par exemple une fonction pour courir (attention AddMovementInput ne prend qu'une valeur entre -1.0 et 1.0). Si vous avez un casque de réalité virtuelle, je vous recommande de regarder le petit triangle à côté du bouton play, il y a un mode VR Preview qui vous permettra

d’essayer votre jeu en VR sans rien faire (si vous utilisez OSVR, il faut activer le plugin dans le gestionnaire de plugins).

Ainsi ce termine notre présentation complète et notre petit exemple à l’appui sur le mode de fonctionnement du moteur graphique Unreal Engine 4. Sur lequel sera développer notre futur projet de mémoire de fin d’étude « Modélisation et implémentation d’une Intelligence Artificielle dans un moteur graphique ».

2.4 Conclusion

Unreal Engine à révolutionner le monde des moteurs graphiques, grâce à son système de programmation visuel (Blueprint) il devient plus facile de faire de montage graphique avec le moins d’erreur possible, et même en cas d’erreurs il est plus facile a corriger vu que le programme est écrit en script visuel. Facilitant du même coup un gain de temps et d’espace.

Le but visé par un moteur de jeu est de permettre à une équipe de développement de se concentrer sur le contenu et le déroulement du jeu plutôt que la résolution de problèmes informatiques.

3 CHAPITRE 3 : Fonctionnement d'une Intelligence Artificielle De jeux vidéo

3.1 Introduction

Depuis les débuts de l'Intelligence Artificielle, une partie s'est toujours intéressée au développement des jeux. En effet, parmi les premières IA développées figurait le jeu de Nim (1952). Il y avait également des programmes écrits pour jouer au jeu de dames ou encore aux échecs. À l'époque, il était surtout question de développer des méthodes et des algorithmes.

Aux premières apparitions des jeux vidéo, vers les années 60, la recherche en IA était en plein essor. Les réalisations comme Pong, Spacewar et Gotcha présentaient exclusivement une compétition entre joueurs. Autrement dit, l'IA n'avait pas encore de rôle dans le domaine.

Mais l'intégration de l'Intelligence Artificielle n'a pas mis beaucoup de temps. En effet, dans les années 70, les jeux ont commencé à proposer un mode solo. En 1974, des jeux d'arcade tels que Qwak, Speed Race ou encore Pursuit sont apparus.

L'IA des jeux vidéo a surtout connu un succès pendant l'âge d'or des jeux d'arcade. Cela correspond notamment à des niveaux de difficultés croissants comme dans Space Invaders ou à des mouvements plus complexes et plus variés des ennemis comme dans Galaxian. En d'autres termes, cette période a vu de nombreux jeux apporter des améliorations grâce à l'Intelligence Artificielle (Pac-Man, First Queen, Dragon Quest IV, Secret of Mana…).

Des jeux vidéo sportifs ont également tiré profit de la technologie pour maximiser la précision ou produire des stratégies de gestion ou d'entraînement définies par le joueur. D'autre part, Creatures et Black & White utilisaient l'IA ascendante, tandis que Façade (2005) utilisait principalement des dialogues interactifs pour le jeu.

Par ailleurs, les plus gros défis des outils IA étaient le développement de nouveaux accessoires de jeu, les informations incomplètes, la recherche de chemin, la planification économique ou encore les décisions en temps réel.

En termes de jeux vidéo, l'IA désigne des algorithmes et des techniques informatiques, robotiques, infographiques ainsi que des techniques de contrôle. À une époque, l'IA de jeu était surtout utilisée pour améliorer les niveaux de difficulté par le biais des adversaires. Plus

récemment, elle se trouve également dans le pathfinding et dans les arbres de décisions afin de guider les actions des PNJ [6].

Nous avons eu un aperçu de la manière dont l'IA a été utilisée pour améliorer les jeux vidéo au fil du temps. Mais pour mieux comprendre son fonctionnement, nous allons entrer davantage dans les détails.

3.2 État des lieux et historique

Intelligence Artificielle et jeux de plateaux comme les échecs ou le Go ont toujours été liés depuis l'émergence de la discipline. En effet, ceux-ci fournissent un terrain propice afin d'évaluer les différentes méthodes et algorithmes d'IA développés. Quand les premiers jeux vidéo ont fait leur apparition, la recherche en IA battait son plein. C'est donc logiquement que l'intérêt a été porté à ce nouveau support. Cependant et pendant longtemps, la recherche a surtout permis l'amélioration des jeux vidéo et non l'inverse.

Le but principal de l'IA dans un jeu vidéo est de rendre le monde dépeint par le jeu le plus cohérent et le plus humain possible afin d'améliorer le plaisir et l'immersion du joueur. Cet objectif de recréer un monde au plus proche du réel peut passer par beaucoup d'approches différentes comme améliorer le comportement des personnages non joueurs (PNJs) ou encore générer du contenu de façon automatique [7]. À l'instar des jeux de plateau, les premiers jeux vidéo présentaient souvent une confrontation entre deux joueurs (comme Pong en 1972). Puisque deux joueurs s'affrontaient, aucune IA n'était donc nécessaire dans ce type de jeu. Mais rapidement, les jeux se complexifient et de nombreux éléments commencent à présenter des comportements indépendants des actions du joueur rendant le jeu plus interactif, comme les fantômes dans Pac Man.

Il devient donc rapidement nécessaire d'implémenter de l'IA dans les jeux vidéo pour être capable de générer ce genre de comportements. Depuis, IA et jeux vidéo sont irrémédiablement liés, l'IA ayant pris une place de plus en plus importante dans les systèmes de jeu.

Désormais, quel que soit le type de jeux vidéo (jeux de course, jeux de stratégie, jeux de rôle, jeux de plateforme, etc.…), il y a de grandes chances pour que de nombreux éléments du jeu soient gérés par des algorithmes d'IA. Cependant cette forte imbrication entre IA et système de jeu ne s'est pas toujours faite sans accroc et, même aujourd'hui, certaines intégrations sont déficientes.

3.3 Fonctionnement d’un Intelligence Artificielle dans un jeu vidéo

L’utilisation de l’IA dans les jeux vidéo n’a pas vraiment d’effets visibles pour les utilisateurs. Elle peut correspondre à une exportation de données ou à une procédure de génération de contenu. En gros, l’Intelligence Artificielle de jeu est surtout un calcul automatisé qui fournit un ensemble de réponses prédéterminé dont les entrées sont limitées [2].

Le 24 janvier 2019, Google DeepMind présente sur son blog AlphaStar, une Intelligence Artificielle dédiée au jeu de stratégie en temps réel StarCraft II qui a affronté deux joueurs humains lors d'un match retransmis en direct sur Internet. Durant cet évènement, AlphaStar bat deux joueurs professionnels, dont Grzegorz « MaNa » Komincz, de l'équipe Team Liquid, l'un des meilleurs joueurs professionnels au monde. Le développement de cette Intelligence Artificielle a été permis par un partenariat entre Google DeepMind et Blizzard Entertainment, l'éditeur du jeu.

L'une des caractéristiques de cette Intelligence Artificielle est qu'elle propose une version implémentant un brouillard de guerre. C'est-à-dire que, contrairement aux personnages contrôlés par le jeu, l'Intelligence Artificielle n'a accès qu'aux informations auxquelles aurait accès un joueur humain. Par ailleurs, le nombre d'actions par minute d'AlphaStar était inférieur au nombre d'actions par minute de ses adversaires. Ce n'est donc pas la rapidité de jeu de l'IA, mais l'efficacité de sa stratégie qui aurait permis à AlphaStar de gagner, bien que cette question soit sujette à controverses

En 2018, une équipe de chercheurs de Google DeepMind affirme sur son blog avoir conçu un programme d'Intelligence Artificielle capable de battre les champions humains du jeu vidéo de tir à la première personne Quake III, en utilisant les bots (les robots) intégrés au jeu. Le mode choisit était le CTF (Capture the flag, « capture du drapeau » en français) : dans ce mode, le but du jeu est de récupérer le drapeau de la base ennemie pour le ramener dans sa propre base, tout en défendant son propre drapeau des assauts ennemis.

N'ayant reçu aucune information avant de commencer à jouer, ces robots (nommés « agents coopératifs complexes » par Deepmind) ont joué des milliers de parties entre eux, apprenant de leurs erreurs pour perfectionner leurs tactiques (« comment voir, agir, coopérer et concourir dans des environnements invisibles, le tout à partir d’un seul signal de renforcement par match » disant aux agents s'ils avaient gagné ou non). À chaque nouvelle partie, une nouvelle map (zone de jeu) était

générée automatiquement (de manière procédurale), permettant de compliquer la tâche aux bots. Les trente bots se sont affrontés sur un demi-million de parties (450 000), afin de maîtriser l'environnement et les différentes tactiques et stratégies inhérentes au jeu. Selon les chercheurs : « Grâce à de nouveaux développements dans l'apprentissage par renforcement, nos agents ont atteint des performances de niveau humain dans Quake III Arena CTF, un environnement multi-agents complexe et l'un des jeux multi-joueurs cultes en 3D à la première personne. Ces agents démontrent leur capacité à faire équipe avec d'autres agents artificiels et des joueurs humains ». Pour Deepmind, « les agents n'ont jamais été informés des règles du jeu, mais ont appris ses concepts fondamentaux et [ont] développé efficacement une intuition pour le CTF ».

Au lieu de former un seul agent, les chercheurs ont entraîné « une population d'agents » qui apprenaient en jouant les uns avec les autres, fournissant ainsi « une diversité de coéquipiers et d'adversaires ». Chaque agent dans la population possède « son propre signal de récompense interne, ce qui permet aux agents de générer leurs propres objectifs internes, comme la capture d'un drapeau ». « Un processus d'optimisation à deux niveaux optimise les récompenses internes des agents directement pour gagner », en se servant de l'apprentissage par renforcement. Selon les chercheurs, « les agents opèrent à deux échelles de temps, rapide et lent, ce qui améliore leur capacité à utiliser la mémoire et à générer des séquences d'actions cohérentes ». Selon un graphique de progression dévoilé par Deepmind, les agents dépassaient déjà le niveau des joueurs humains moyens après 150 000 parties.

Par la suite, un tournoi est organisé entre des binômes de machines contre des binômes humains (40 joueurs humains), ainsi que des duos mixtes humains/machines entre eux. Selon DeepMind, les équipes de bots atteignirent un taux de victoire probable de 74 %. En comparaison, les très bons joueurs humains n'ont atteint que 52 % de taux de victoire. Les équipes composées d'agents uniquement sont restées invaincues lors de leurs confrontations avec équipes composées exclusivement d'humains. Les duos d'agents avaient 95 % de chances de gagner contre des équipes humains/agent artificiel. Par ailleurs, les chercheurs ont observé que la probabilité de victoire par les bots baissait si le nombre de membres dans l'équipe augmentait. Cela s'explique par le fait que l'Intelligence Artificielle apprend à jouer au jeu en solo, mais ne comprend pas encore complètement la notion de coopération, souvent capitale dans un jeu en équipe. Ce paramètre était l'un des objets de cette expérience, visant à améliorer la conscience collective des IA, ce qui est un point déterminant dans le développement d'une espèce.

3.3.1 La Conscience collective

Dans le jeu vidéo l'ensemble des personnages et systèmes partagent des fonctionnalités communes, ces derniers sont interconnectés que ce soit à travers une variable ou une fonction. Ceux si fonctionnent en collaboration afin de rendre l'expérience de jeux plus fluide. Malgré que chaque IA procède ça propre conscience ils sont tous gouvernés par une conscience collective appeler Game mode. C'est lui qui gère toutes les règles au saint du jeu, La majorité des systèmes d'intelligences artificielles au saint d'un jeu ne peuvent effectuer que la tache pour laquelle elles ont été programmer car un système d'Intelligence Artificielle chargé de faire apparaitre un projectile au bout du canon d'une arme est incapable d'affiché une interface widget. Le Game mode est chargé donc ici de les faire fonctionner en toute synchronisation.

Dans le domaine du jeu vidéo on parle de conscience de type fourmilière, où le Game Mode désigne la reine.

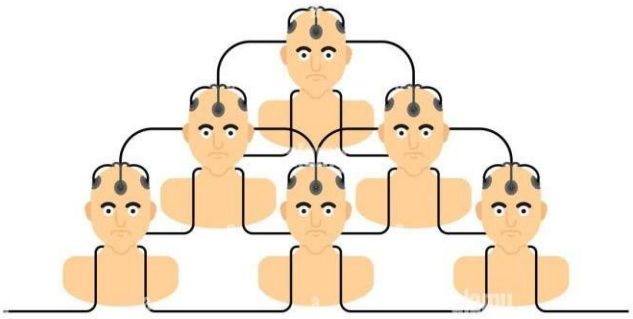

***Figure 11** : Conscience collective d'IA*

3.3.2 IA et jeux vidéo, faux frères ?

Au commencement du domaine, il y a longtemps eu un fossé entre la recherche en IA classique et l'IA effectivement implémentée par les développeurs dans les jeux vidéo. En effet, il existait entre les deux des différences majeures dans les connaissances, les problématiques rencontrées mais aussi les façons de résoudre ces problématiques. La principale raison derrière ces différences provient sans doute du fait que ces deux approches de l'IA n'ont pas vraiment les mêmes objectifs.

La recherche en IA classique aspire en général à améliorer ou créer de nouveaux algorithmes pour faire avancer l'état de l'art. En revanche, le développement d'une IA dans un jeu vidéo a pour objectif de créer un système cohérent qui s'intègre le mieux possible dans le design du jeu afin d'être amusant pour le joueur. Ainsi, une IA très performante qui n'est pas bien intégrée au gameplay peut davantage desservir le jeu qu'elle ne va l'améliorer.

Développer une IA pour un jeu vidéo requiert donc souvent de trouver des solutions d'ingénierie à des problèmes peu ou pas du tout adressés par la recherche classique en IA. Par exemple, un algorithme d'IA dans un jeu est très fortement contraint en matière de puissance de calcul, de mémoire et de temps d'exécution. En effet, le jeu doit tourner par le biais d'une console ou d'un ordinateur "ordinaire" et il ne doit pas être ralenti par l'IA. C'est pourquoi certaines solutions de l'état de l'art en IA gourmandes en ressources n'ont pu être implémentées dans les jeux vidéo que plusieurs années après leur utilisation en IA classique.

Les solutions possibles pour résoudre un problème donné sont aussi une distinction entre les deux domaines. Par exemple, il est souvent possible de contourner une problématique difficile d'IA en modifiant légèrement la conception du jeu. De plus, dans un jeu vidéo, une IA peut se permettre de tricher (en ayant accès à des informations qu'elle n'est pas censée avoir, ou en ayant plus de possibilités que le joueur par exemple) afin de compenser son manque de performance. La triche d'une IA en soi n'est pas un problème tant qu'elle permet d'améliorer l'expérience du joueur. Cependant, en général, il est difficile de bien dissimuler le fait que l'IA triche et cela peut entraîner de la frustration pour le joueur.

On peut noter que la plupart des grands concepts de jeux vidéo (Jeux de course, de plateforme, de stratégie, de tir, etc…) ont été créés autour des années 80 et 90. À cette époque, le développement d'une IA complexe dans un jeu était quelque chose de difficilement réalisable au vu des ressources et méthodes existantes. Les concepteurs de jeux de l'époque ont donc souvent dû composer avec le manque d'IA en faisant des choix adaptés de Game Design. Certains jeux ont même été pensés pour ne pas avoir besoin d'IA. Ces conceptions basiques ont été héritées par les différentes générations de jeux vidéo qui, pour beaucoup, restent assez fidèles aux canons du genre qui fonctionnent. Ceci peut donc aussi expliquer pourquoi les systèmes d'IA dans les jeux vidéo sont en général assez basiques quand on les compare aux approches de la recherche classique.

Ainsi, les méthodes que nous allons présenter pourraient surprendre par leur relative simplicité. En effet, lorsque l'on entend parler d'IA aujourd'hui, cela fait référence au Deep

Learning et donc aux réseaux de neurones très complexes et abstraits. Cependant, le Deep Learning n'est qu'un sous domaine de l'IA, et les méthodes symboliques ont été les approches communément utilisées pendant longtemps. On fait référence aujourd'hui à ces approches avec le nom GOFAI (« Good Old-Fashioned Artificial Intelligence » ou "bonne vieille IA démodée" en français). Comme son surnom l'indique, l'IA symbolique n'est clairement plus l'approche la plus populaire mais elle permet cependant de résoudre de nombreux problèmes, notamment dans le cadre du jeu vidéo.

3.3.3 L'IA pour les jeux vidéo

Le critère principal de succès d'une IA dans un jeu classique est probablement son niveau d'intégration et d'imbrication dans le design du jeu. Par exemple, des PNJs ayant un comportement injustifiable peuvent briser l'expérience de jeu prévue et donc l'immersion du joueur.

L'un des exemples les plus classiques dans les vieux jeux est le cas où les Personnages Non Joueurs (PNJs) se retrouvent coincés dans le décor. À l'inverse, une IA qui participe pleinement à l'expérience de jeu aura forcément un impact positif sur le ressenti du joueur, en parti parce que les joueurs sont plus ou moins habitués aux comportements parfois incohérents des IA.

Suivant le type de jeu, une IA peut avoir des tâches très variées à résoudre. Nous n'allons donc pas nous pencher sur l'ensemble des cas d'utilisation possibles d'une IA dans un jeu mais plutôt sur les principaux. Il s'agira donc notamment de la gestion du comportement des PNJs (alliés ou ennemis) dans les jeux de tir ou les jeux de rôle, ainsi que de la gestion plus haut niveau d'un agent qui doit jouer à un jeu de stratégie.

3.3.4 Les personnages non joueurs (PNJ)

Il existe un grand nombre d'approches permettant de développer des comportements réalistes dans un jeu vidéo. Cependant, le côté réaliste ne suffit pas à rendre un jeu amusant. En effet, il ne faut pas oublier que la finalité d'un ennemi dans un jeu reste en général de se faire éliminer par le joueur. Un jeu avec des adversaires trop réalistes ne sera pas forcément agréable à jouer. Par exemple, des PNJs qui passent leur temps à fuir risquent d'agacer le joueur plus qu'autre chose. Il y a donc un compromis à trouver dans le comportement des PNJs qu'il n'est pas forcément facile à balancer.

Dans cette partie, nous allons faire un tour d'horizon des solutions principales pour essayer de résoudre cette problématique. Mais pour commencer, faisons le point sur quelques notions importantes.

- Les agents

Probablement la notion première lorsque l'on parle d'Intelligence Artificielle pour les jeux vidéo, un agent est une entité qui peut obtenir des informations sur son environnement et prendre des décisions en fonction de ces informations dans le but d'atteindre un objectif. Dans un jeu vidéo, un agent représente souvent un PNJ, mais il peut aussi représenter un système plus complet comme un adversaire dans un jeu de stratégie. Il peut donc y avoir plusieurs niveaux d'agents qui communiquent entre eux dans un même jeu.

- Les états

Un état est une configuration unique de l'environnement dans lequel un agent se trouve. L'état peut changer quand un agent (ou le joueur) effectue une action. Pour un agent, l'ensemble des états possibles s'appelle l'espace des états (state space en anglais). C'est une notion importante puisque l'idée de base de la plupart des méthodes d'IA est de parcourir ou d'explorer l'espace des états pour trouver la meilleure façon d'agir en fonction de la situation présente. Les caractéristiques de l'espace des états impactent donc la nature des méthodes d'IA utilisables.

Si l'on considère par exemple un agent qui définit le comportement d'un PNJ, l'espace des états peut être défini de façon assez simple : il s'agit de l'ensemble des situations dans lequel le PNJ peut se trouver. On peut ainsi mettre en place un état dans lequel le PNJ ne voit pas le joueur, un état dans lequel il voit le joueur, un état lorsque le joueur tire sur le PNJ et un état quand le PNJ est blessé. En considérant ainsi l'ensemble des situations possibles pour le PNJ, on peut déterminer les actions qu'il peut entreprendre ainsi que les conséquences de ces actions en termes de changement d'état (on parcourt l'espace des états facilement). On peut donc intégralement définir le comportement de l'agent et choisir les actions les plus pertinentes à chaque instant en fonction de ce qui se passe.

A l'inverse, dans le cas d'un agent qui joue à un jeu comme les échecs, l'espace des états sera l'ensemble des configurations possibles des pièces sur le plateau de jeu. Cela représente un nombre de situations extrêmement grand (10^47 états pour les échecs). Dans ces conditions, Il n'est plus possible d'explorer toutes les possibilités. Par conséquent, on ne peut plus déterminer les conséquences des actions de l'agent et donc de juger de la pertinence de ses actions. On est donc forcé d'oublier les approches manuelles et de s'orienter sur d'autres méthodes plus complexes (que l'on abordera plus tard).

❖ La recherche de chemin

Avant de pouvoir modéliser le comportement d'un PNJ en explorant l'espace des états, il faut définir comment celui-ci peut interagir avec son environnement et notamment, comment il peut se déplacer dans le monde du jeu. La recherche de chemin (pathfinding), qui est le fait de trouver le plus court chemin entre un point A et un point B, est donc souvent l'une des briques de base d'un système d'IA complexe. Si cette brique est défaillante, tout le système en pâtira. En effet, des PNJs ayant un pathfinding défaillant risquent de se coincer dans le décor et donc d'impacter négativement l'immersion du joueur.

L'algorithme le plus utilisé pour le pathfinding est l'algorithme A*. Il s'agit d'une version plus rapide d'un algorithme plus ancien, l'algorithme de Dijkstra. Tous deux sont ce que l'on nomme des algorithmes de parcours de graphe. Pour utiliser ces algorithmes dans un jeu vidéo, le terrain de jeu (qui prend en compte les éventuels obstacles et parties cachées) est modélisé sur une grille en 2 dimensions (une grille est une forme de graphe). L'algorithme va s'exécuter dans cette grille 2D.

Le principe de Dijkstra est relativement simple, il va parcourir dans l'ordre chaque case de la grille en fonction de sa distance avec la case de départ jusqu'à trouver la case de destination.

Là où Dijkstra trouve toujours la solution optimale, A* peut ne trouver qu'une solution approchée. Cependant, il le fera beaucoup plus rapidement, ce qui est un point critique dans un jeu vidéo. En plus d'avoir l'information de distance entre chaque case avec la case de départ, l'algorithme calcule approximativement la direction générale dans laquelle il doit aller pour se rapprocher de la destination. Il va ainsi chercher à parcourir le graphe en se rapprochant toujours de l'objectif, ce qui permet de trouver une solution bien plus rapidement.

Dans des cas plus complexes où l'algorithme rencontre un obstacle, il repartira du début pour trouver un autre chemin qui se rapproche de la destination mais en évitant l'obstacle.

Pour les jeux plus récents en trois dimensions, le monde n'est pas représenté par une grille mais par un équivalent en 3D que l'on appelle un maillage de navigation. Un maillage de navigation est une simplification du monde qui ne considère que les zones dans lesquels il est possible de se déplacer afin de faciliter le travail de l'algorithme de pathfinding. Le principe de recherche de chemin reste le même.

Ces algorithmes de parcours de graphe peuvent sembler simples et l'on peut se demander s'il s'agit vraiment d'IA. Il faut savoir qu'une bonne partie de l'IA peut se réduire finalement à

des algorithmes de recherche. Cette sensation qu'un problème résolu par IA n'était finalement pas si compliqué s'appelle l'effet IA.

Mais pour ce qui est de la recherche de chemin, ce n'est pas un problème si simple, et des chercheurs s'y intéressent toujours. Par exemple en 2012, une alternative à l'algorithme A* a été développée : la méthode Jump Point Search (JPS). Elle permet, lors de la recherche dans certaines conditions, de sauter plusieurs cases d'un coup et ainsi de gagner du temps par rapport à A* qui ne peut chercher son chemin que case par case.

Les implémentations de pathfinding dans les jeux vidéo sont aujourd'hui souvent des variations de A* ou de JPS, adaptées aux problématiques spécifiques de chaque jeu.

Le pathfinding est une composante importante d'un système comportemental des PNJs. Il y en a d'autres comme par exemple la gestion de la ligne de vue (Les PNJs ne doivent pas voir dans leur dos, ni à travers les murs) ou encore les interactions avec l'environnement qui sont nécessaires avant de pouvoir implémenter un comportement cohérent. Mais puisqu'elles ne sont, en général, pas gérées par de l'IA, nous n'allons pas nous attarder davantage dessus. Il ne faut cependant pas oublier que le système final nécessite toutes ces briques pour fonctionner. Une fois que notre PNJ est capable de se déplacer et d'interagir, on peut développer son comportement.

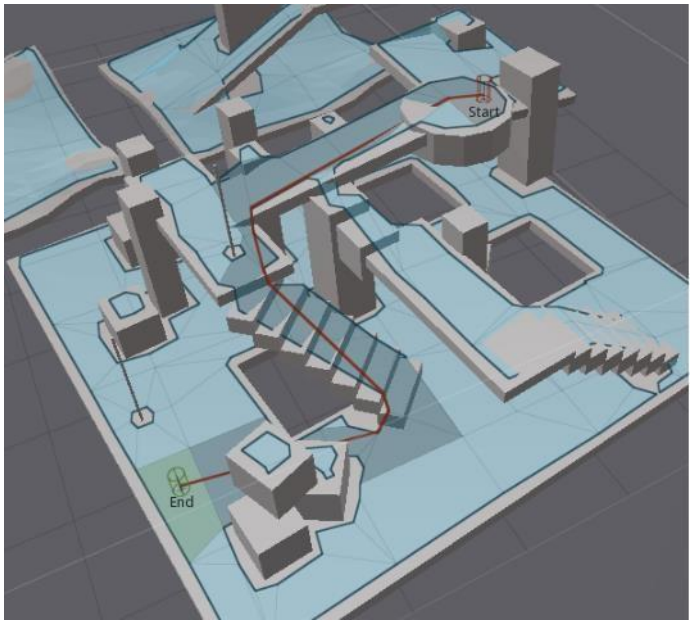

***Figure 12** : Recherche de chemin dans un maillage de navigation 3D*

3.3.5 La création de comportements Ad-hoc

La création de comportements Ad-hoc (Ad hoc behaviour authoring en anglais) est probablement la classe de méthode la plus courante pour implémenter de l'IA dans un jeu vidéo. Le terme même d'IA dans les jeux vidéo fait encore aujourd'hui surtout référence à cette approche. Les méthodes Ad Hoc sont des systèmes experts, c'est-à-dire des systèmes où l'on doit définir manuellement un ensemble de règles qui vont servir à définir le comportement de l'IA. Ce sont donc des méthodes manuelles de parcours de l'espace des états. Leur application est donc limitée aux agents confrontés à un nombre de situations possibles assez restreint, ce qui est en général le cas pour les PNJs. Contrairement aux algorithmes de recherche comme A*, les méthodes ad-hoc ne peuvent peut-être pas réellement être définies comme de l'IA classique, mais elles sont considérées comme telles par l'industrie du jeu vidéo depuis son commencement.

- **Les automates finis (Finite State Machine)**

Un automate fini permet de manuellement définir les objectifs et les stimuli d'un agent afin de dicter son comportement. Pour chaque objectif, on définit un ou des états et chaque état dicte une ou des actions pour le PNJ. En fonction d'événements extérieurs ou de stimuli, l'objectif du PNJ peut changer et donc l'automate peut transitionner d'un état à un autre.

Un automate fini est donc simplement un moyen de représenter l'espace des états d'un agent ainsi que l'ensemble des transitions entre ses différents états. Si les états et les transitions sont bien définis, l'agent présentera un comportement pertinent dans toutes les situations. Les automates finis sont en général représentés sous la forme d'un diagramme de flux comme l'exemple ci-dessous.

L'avantage principal des automates finis est qu'ils permettent de modéliser un comportement pertinent assez simplement et rapidement puisqu'il suffit de définir manuellement les règles qui définissent les différents états et transitions. Ils sont aussi pratiques puisqu'ils permettent de voir visuellement comment peut se comporter un PNJ.

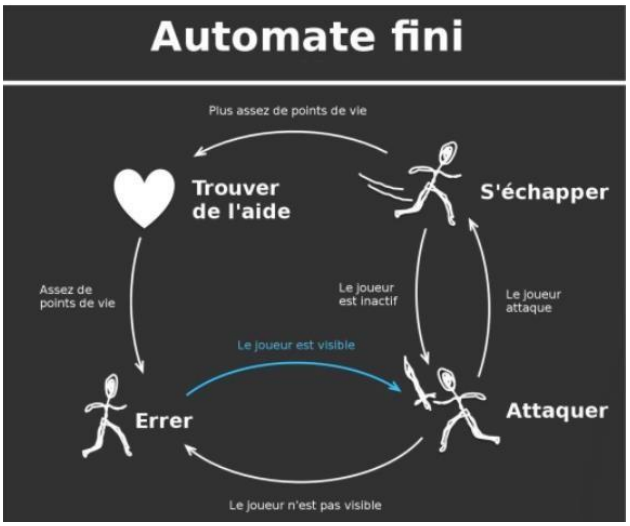

***Figure 13** : Finite State Machine*

Cependant, un automate fini ne présente aucune adaptabilité, puisque une fois l'automate défini et implémenté, le comportement de l'IA ne pourra pas évoluer ou s'adapter. Pour un même objectif, le PNJ effectuera toujours les mêmes actions. Il présentera donc souvent un comportement relativement simple et prévisible (même s'il est toujours possible de présenter de l'imprévisibilité en ajoutant un peu d'aléatoire dans les transitions par exemple).

Un automate fini permet de manuellement définir les objectifs et les stimuli d'un agent afin de dicter son comportement. Pour chaque objectif, on définit un ou des états et chaque état dicte une ou des actions pour le PNJ. En fonction d'événements extérieurs ou de stimuli, l'objectif du PNJ peut changer et donc l'automate peut transitionner d'un état à un autre.

Un automate fini est donc simplement un moyen de représenter l'espace des états d'un agent ainsi que l'ensemble des transitions entre ses différents états. Si les états et les transitions sont bien définis, l'agent présentera un comportement pertinent dans toutes les situations. Les automates finis sont en général représentés sous la forme d'un diagramme de flux comme l'exemple ci-dessous.

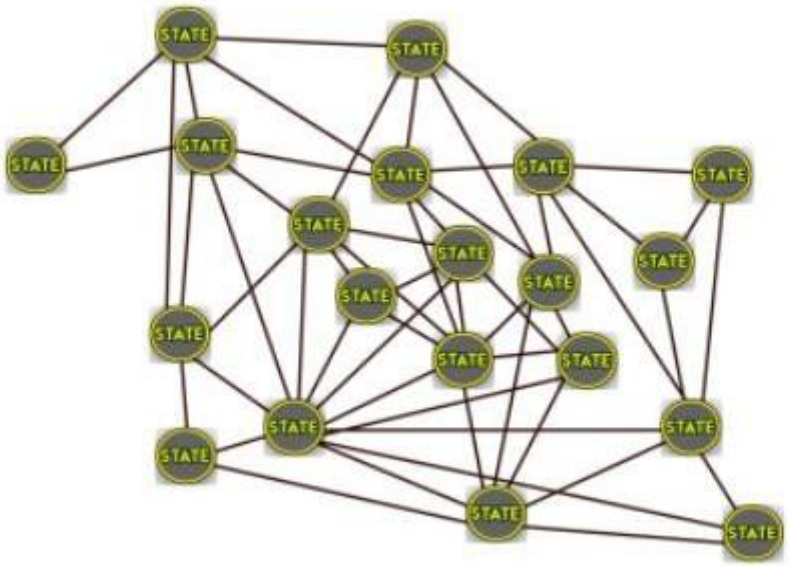

Figure 14 : Etat d'un automate fini

L'avantage principal des automates finis est qu'ils permettent de modéliser un comportement pertinent assez simplement et rapidement puisqu'il suffit de définir manuellement les règles qui définissent les différents états et transitions. Ils sont aussi pratiques puisqu'ils permettent de voir visuellement comment peut se comporter un PNJ.

Les automates finis, par leur conception simple, permettent de compléter, de modifier ou de debugger un système d'IA relativement facilement. Il est ainsi possible de travailler sous forme itérative afin de pouvoir rapidement vérifier les comportements implémentés.

Cependant, lorsque l'on veut mettre en place un comportement plus complexe avec un automate fini, cela devient vite compliqué. Il faut définir manuellement un grand nombre d'états et de transitions et la taille de l'automate peut très vite impacter la facilité de débuggage et de modification.

Une évolution des automates finis essaie d'adresser ce problème : les automates finis hiérarchiques. Cette approche groupe des ensembles d'états similaires afin de limiter le nombre de transitions entre les états possibles. L'objectif est de garder la complexité de ce qu'on souhaite modéliser, tout en simplifiant le travail de création et de maintenance du modèle en structurant les comportements.

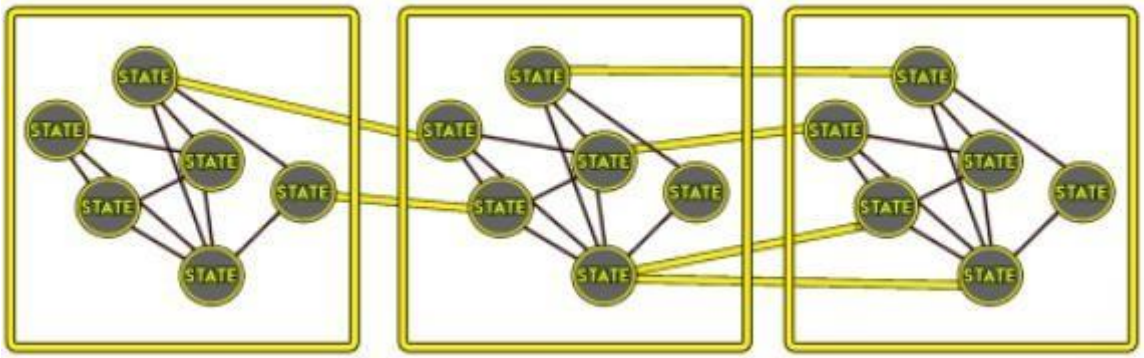

Figure 15 : Un automate fini hiérarchique

Les automates finis ont été la méthode la plus populaire pour construire des comportements simples dans les jeux vidéo jusqu'au milieu des années 2000. Ils sont encore utilisés sur certains jeux récents comme la série des Batman : Arkham ou DOOM (qui utilise des automates finis hiérarchiques pour modéliser le comportement des ennemis). Cependant, ils ne sont plus les algorithmes les plus fréquemment utilisés sur le marché.

- **Les arbres de comportements (Behaviour Tree)**

Les arbres de comportements sont assez proches des automates finis, à la différence qu'ils se basent directement sur les actions (et non les états), et qu'ils représentent les différentes transitions entre actions sous la forme d'un arbre. La structure de représentation sous forme d'arbre permet d'implémenter beaucoup plus simplement des comportements complexes. C'est aussi plus simple à maintenir, car on obtient une meilleure idée des possibilités d'action de l'agent en parcourant l'arbre qu'avec un diagramme de flux.

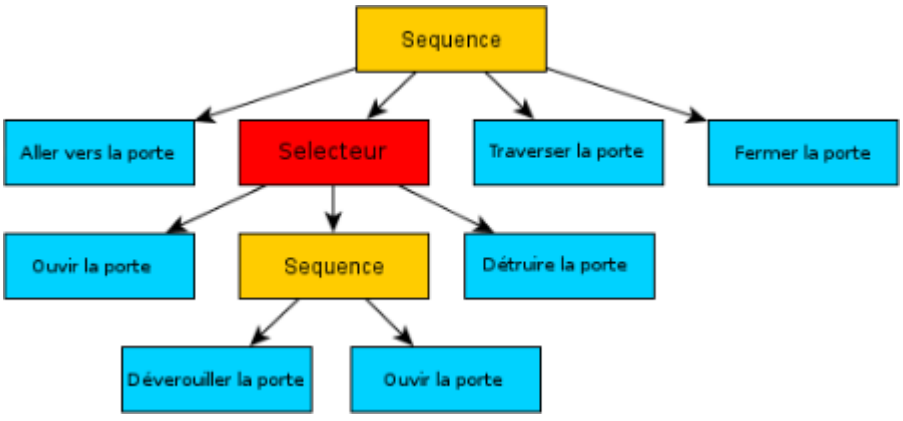

Figure 16 : Un exemple d'arbre de comportement

Un arbre de comportement est constitué de différents types de nœuds :

- Les nœuds feuilles (ci-dessus en bleu) qui n'ont aucun descendant et qui représentent des actions
- Les nœuds sélecteurs (en rouge) qui ont plusieurs descendants et qui permettent de choisir le plus pertinent en fonction du contexte
- Les nœuds séquences (en jaune) qui exécutent tous leurs nœuds descendants dans l'ordre et permettent donc de faire des séquences d'actions

Un arbre de comportement est exécuté de haut en bas et de gauche à droite On commence du nœud racine (le tout premier nœud) jusqu'à la feuille la plus à droite. Une fois arrivé à la feuille et l'action exécutée, on retourne au nœud racine et on parcourt jusqu'à la feuille suivante.

❖ **Les approches basées sur l'utilité**

Avec cette approche, plutôt que de transitionner d'un état à un autre en fonction d'évènements extérieurs, un agent va constamment évaluer les différentes actions possibles qu'il peut effectuer à un instant donné et choisir l'action qui a la plus forte utilité pour lui dans les conditions présentes. Pour chaque action on définit en amont une courbe d'utilité en fonction des conditions. On peut par exemple définir une courbe d'utilité sur le fait de tirer sur le joueur en fonction de la distance avec celui-ci. Plus le joueur est proche plus il semble utile de tirer dessus pour éviter de se faire éliminer. En choisissant différents types de courbes (linéaires, logarithmiques, exponentielles, etc..) pour les différentes actions, on peut prioriser quelle action effectuer dans quelles conditions sans avoir à détailler manuellement tous les cas possibles.

Cette approche permet donc de développer des comportements complexes et beaucoup plus modulaires sans avoir à définir un très grand nombre d'états. La liste des actions possibles peut facilement être complétée ou modifiée avec ce type d'approche, ce qui n'était pas le cas avec des automates finis ou des arbres de comportements. Cependant, il est quand même nécessaire de définir toutes les courbes d'utilité et de bien les paramétrer pour obtenir les comportements voulus.

Cette approche, grâce à ses avantages, est de plus en plus utilisée dans les jeux vidéo. On peut citer notamment Red Dead Redemption (2010) et Killzone 2 (2009).

Cette approche, utilisée pour la première fois en 2005 pour le jeu F.E.A.R, est basée sur l'utilisation d'automates finis ainsi qu'une recherche par A* dans les états de ces automates.

Dans le cas d'un automate fini classique, la logique qui détermine quand et comment transitionner d'un état à un autre doit être spécifiée manuellement ce qui peut être problématique pour des automates avec beaucoup d'états comme on l'a vu précédemment. Avec l'approche GOAP, cette logique est déterminée par le système de planification plutôt que manuellement. Cela permet à l'agent de prendre ses propres décisions pour passer d'un état à un autre.

Avec cette approche, l'espace des états est représenté par un automate fini. Contrairement à l'approche classique où l'on définit un automate avec beaucoup d'états qui correspondent tous à une action spécifique, on va définir un petit automate avec seulement quelques états assez abstraits. Chaque état abstrait correspond donc à de grandes catégories d'actions qui seront dépendantes du contexte. Par exemple, les automates finis des PNJs dans F.E.A.R présentent seulement 3 états très abstraits. Un des états permet de gérer les déplacements et les deux autres permettent de gérer les animations. Le comportement complexes des PNJs est donc simplement une succession de déplacements et d'animations.

L'intérêt de l'approche GOAP est que, suivant le contexte dans lequel un objectif est défini, plusieurs plans d'actions différents peuvent être trouvés pour résoudre un même problème. Ce plan d'action est construit à travers les états de l'automate fini qui dictent les mouvements et animations du PNJ.

Quoi qu'il arrive, le plan (la séquence d'actions) trouvé par l'algorithme A* sera toujours une réflexion de l'objectif de l'agent à l'instant présent. Si les objectifs des PNJs sont toujours bien définis, ils présenteront un comportement cohérent, complexe et varié voire parfois surprenant. En effet, ces plans peuvent être plus "long terme" que ceux définis manuellement par un simple automate fini ou un arbre de comportement. C'est pourquoi on peut apercevoir des ennemis se déplacer pour contourner le joueur dans F.E.A.R par exemple.

Cette approche permet d'obtenir des comportements très crédibles dans F.E.A.R, mais elle n'est cependant pas forcément adaptée à tous les types de jeux. En effet, pour des jeux très ouverts et complexes où le nombre d'actions possibles est important la méthode peut présenter des limitations. Dans F.E.A.R, les plans trouvés par l'algorithme de recherche restent relativement courts (en général les séquences ne font pas plus de 4 actions), mais comme les actions et les états sont définis de façon abstraite et haut niveau, une séquence de 4 actions est suffisante pour changer de pièce, contourner le joueur, déplacer des objets, se mettre à couvert etc. Si un jeu nécessite de trouver des plans avec des séquences d'actions beaucoup plus longues, il est fréquent que l'algorithme soit trop lourd en termes de ressources et que les plans trouvés divergent trop du comportement visé par les concepteurs. En effet, ce genre d'approche apporte moins de contrôle sur le résultat final qu'un arbre de comportement ad hoc par exemple.

C'est le problème qu'ont rencontré les développeurs du jeu Transformers : War for Cybertron en 2010. Ils ont donc décidé de changer d'approche et se sont tournés vers une autre méthode de planning : l'approche Hierarchical Task Network planning (HTN). Pour ceux qui sont intéressés, leurs problématiques et l'implémentation de l'approche sont expliqués dans cet article d'AI and Games.

Cette approche dans F.E.A.R a été à l'époque de sa sortie une petite révolution dans le monde de l'IA dans les jeux vidéo et elle a depuis été modifiée et améliorée suivant les besoins (avec l'approche HTN par exemple). Par exemple, on retrouve cette approche dans S.T.A.L.K.E.R, Just Cause 2, Tomb Raider, Middle Earth : Shadow of Mordor ou encore Deus Ex : Human Revolution.

- **Les jeux de combat**

L'IA joue également un rôle dans les jeux vidéo d'action qui comportent des niveaux de combat. Le premier cas d'utilisation de l'Intelligence Artificielle concerne sa capacité à chasser. Pour ce faire, elle recherche des marqueurs réalistes (sons ou empreintes de pas) du personnage. Cela contraint le joueur à utiliser la meilleure manière d'approcher l'ennemi.

Par ailleurs, l'IA a également développé un instinct de survie, toujours en identifiant des éléments dans l'environnement pour déterminer s'ils lui sont bénéfiques ou néfastes. Ainsi, si l'IA sent qu'il court un danger, il peut se mettre à l'abri avant de régénérer sa force (recharger une arme, lancer une grenade, etc.). De même, elle peut être programmée pour réagir de manière spécifique à un seuil de santé prédéfini.

❖ La recherche arborescente Monte-Carlo (MCTS)

La recherche arborescente Monte-Carlo est une méthode qui permet de créer des obstacles supplémentaires pour le joueur afin d'améliorer le gameplay. En termes simples le MCTS est un diagramme en arbre sur lequel l'IA sélectionne un chemin pour aboutir au prochain obstacle auquel le joueur doit faire face.

La méthode MCTS permet de rechercher efficacement des solutions dans un espace impossible à parcourir entièrement. La méthode se base sur des notions d'apprentissage par renforcement, particulièrement le dilemme exploration-exploitation. L'exploitation consiste à parcourir les branches de l'arbre que l'on pense être les plus pertinentes pour trouver la meilleure solution. L'exploration à l'inverse consiste à parcourir de nouvelles branches que l'on ne connaît pas ou que l'on avait préalablement considérés comme non pertinentes.

Il peut sembler logique de ne vouloir faire que de l'exploitation, c'est-à-dire toujours se focaliser sur les solutions que l'on pense être les meilleures. Cependant la méthode MCTS est utilisée dans les cas où il n'est pas possible de rechercher l'ensemble des solutions puisque l'arbre est trop grand (sinon on utilise un algorithme comme le minimax). Dans ce cas-là il n'est pas possible de savoir si la solution que l'on considère est vraiment la meilleure. Peut-être qu'une autre partie de l'arbre non parcourue contient de meilleures solutions. De plus, une branche non pertinente à un moment donné, peut le devenir plus tard, c'est pourquoi si l'on fait seulement de l'exploitation on risque de rater des solutions intéressantes. La méthode MCTS permet de trouver un compromis entre exploration et exploitation afin de trouver rapidement des solutions satisfaisantes.

❖ Les réseaux antagonistes génératifs (GAN)

Certains développeurs utilisent également les GAN pour générer de nouveaux contenus dans les jeux vidéo. En s'entraînant sur des niveaux de jeux créés par les humains, ces réseaux peuvent créer des personnages, un gameplay et des graphismes [3]. Cette méthode a par exemple été utilisée dans DOM (1993) et Super Mario.

3.4 Conclusion

Comme on a pu le voir les méthodes d'implémentations d'IA dans les jeux vidéo ont évolué sans cesse au cours du temps pour devenir de plus en plus complexes et poussées. On peut ainsi se demander pourquoi lorsque l'on joue, on a la sensation que l'IA stagne et qu'elle ne s'améliore pas vraiment. La réponse est très probablement que les jeux deviennent de plus en plus grands et compliqués et donc que les tâches que les intelligences artificielles doivent résoudre pour être cohérentes sont de plus en plus complexes.

4 CHAPITRE 4 : Application Dans Le Développement D’un Jeu De Type Topdown Shooter

4.1 INTRODUCTION

Dans ce chapitre nous mettrons en pratique tous ce qui a été exposer dans les chapitres précédents. Et pour cela nous allons créer un jeu de type Topdown shooter, les jeux "*Top down*" sont ceux dont la caméra est vu du dessus. Nous combinerons ce dernier à la fonctionnalité Twin Stick Shooter, un Twin stick shooter est un jeu dont les contrôles sont attribués à chacun des deux *sticks analogiques* : un pour avancer (le gauche en général), l'autre pour viser et tirer (en général, le droit). Avec ces contrôles particuliers, les twin stick shooters sont souvent adaptés avec une caméra vue du dessus.

Nous mettrons en lumière les relations entre les différentes classes de blueprint et les connections avec le système d’intelligence mère (le Game Mode).

4.2 Mise en pratique

Notre jeu sera Développer sur le moteur Unreal Engine 4, Version 4.26.2

Installer sur Un PC de marque HP Elite Book

RAM 8 giga

Processeur 6ieme génération

Core i5

4 giga de carte graphique

Espace nécessaire sur le Disque 30giga octet

4.2.1 Création de la map

Notre Map a été créée par assemblage d'éléments de forme géométriques(box, pavé..) Et habiller par des textures simulant des matériaux de la vraie vie.

Nous avons aussi ajouté un Navigation Mech ; afin de délimiter l'espace sur laquelle les IA pourront se déplacer, Ne vous inquiété pas, cette zone peut être caché en appuyant sur a touche P du clavier, et est invisible IN GAME.

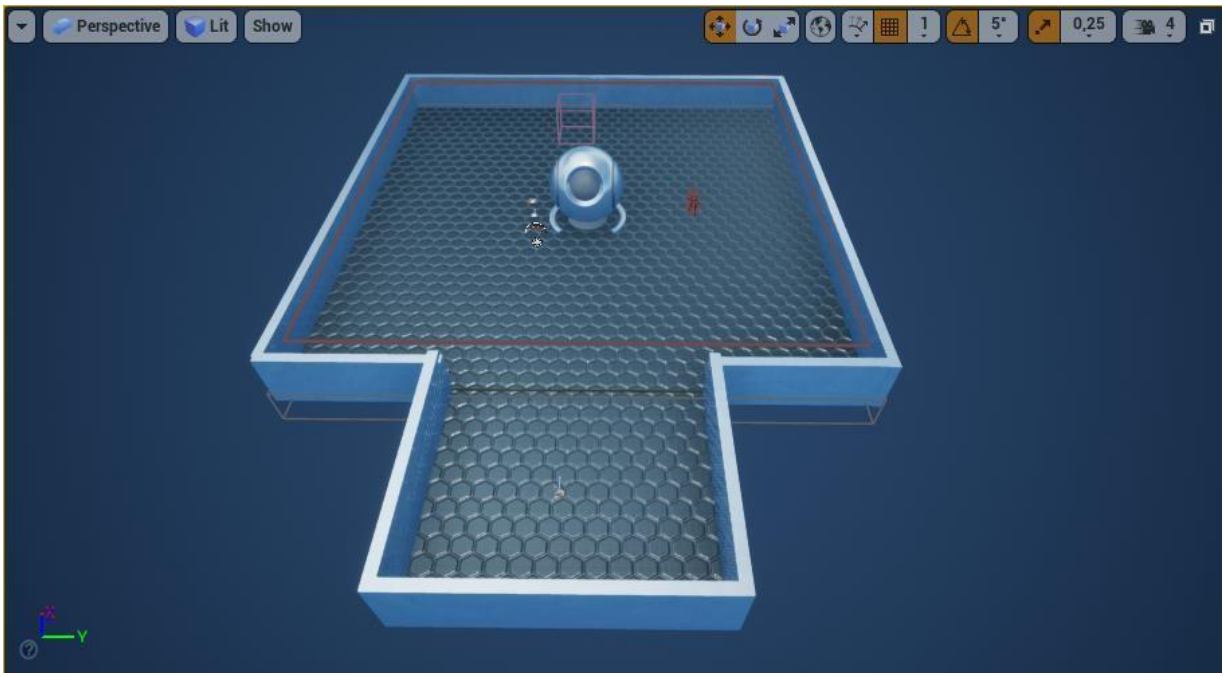

Figure 17 : La Map

4.2.2 Création du Blueprint du personnage

Les personnages de notre jeu partagent des fonctionnalités communes.

Dans cette partie nous allons créer notre propre classe au travers de notre tout premier blueprint, la classe que nous allons créer va hériter de la classe Character qu'on a présenté dans le chapitre précédent de notre mémoire.

Elle aura ainsi les mêmes (components) qu'un personnage de base, comme par exemple une capsule pour gérer les collisions ou même la gestion de mouvement avec les « Character Movement Component », et en plus de ses composants de base on va y ajouter les fonctionnalités qui seront propres à notre personnage. Donc les fonctionnalités qui vont être

propres à notre classe personnage seront relativement simples puisqu'elles vont permettre de gérer les points de vie de ce dernier , et ces fonctions vont non seulement retirer des points de vie au personnage mais aussi envoyer un signal si jamais les points de vie atteignent zéro et que le personnage est censé mourir, et la force de notre implémentation c'est que lorsque nous allons créer le blueprint du héros et de ses ennemis, nous allons faire en sorte qu'il aient eux aussi accès aux fonctionnalités liées à la gestions des points de vie. La logique permettant de vie est la même peu importe que ce soit le héros ou l'ennemi donc plutôt que de dupliqué le code lier à la gestion des points de vie d'un blueprint à un autre autant mieux coder une seul fois la logique dans une seule classe mère pour utiliser les avantages proposés par l'héritage en programmation.

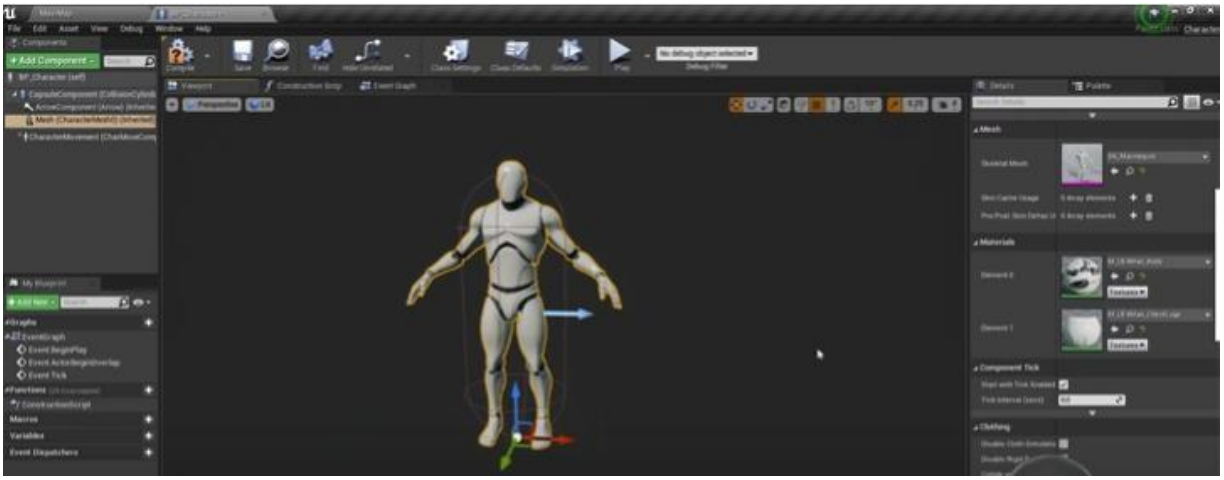

***Figure 18** : La Classe Mère Bp_character*

Character Movement quant à lui est un component qui permet de faciliter les déplacements du personnage. Nous avons par la suite codé la logique permettant de gérer les points de vie au niveau de l'Event Graph.

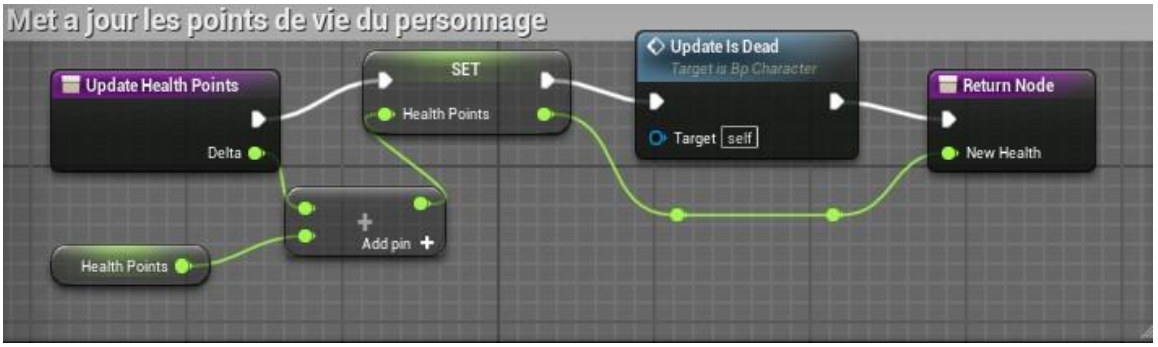

***Figure 19** : Mettre à jours les point de vie du personnage*

Maintenant qu'on sait mettre ajour les points de vie du personnage, il nous reste à tester, qu'à chaque fois qu'il perdra des points de vie on doit vérifier s'il lui reste encore des points de vie parce que sinon il faudra envoyer le signal que le personnage est mort. Et à quel moment est-ce qu'il meurt ? il meurt lorsque les points de vie sont inférieurs ou égal à zéro.

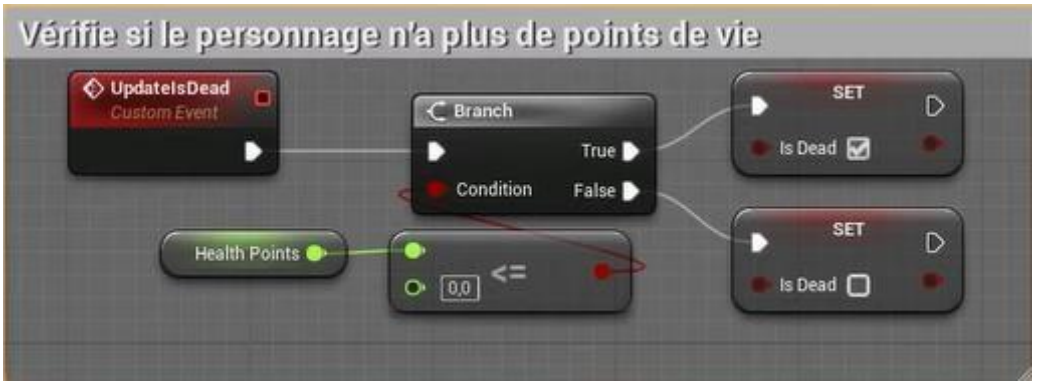

***Figure 20** : vérifier si le personnage n'a plus de point de vie*

Parce qu'à chaque qu'on va mettre à jour on va vouloir faire appel à la fonction Update Is Dead pour mettre à jour également l'état du personnage (est-ce qu'il est mort ou pas ?). On va donc faire appel à l'évènement créer plus tôt. Compiler et sauvegarder, on a donc terminé la création de notre classe mère (Bp_Character).

4.2.3 Création du Blueprint du héros

Nous allons paramétrer le blueprint du héros de notre jeu. Dans la partie précédente on a créé notre blueprint mère le (Bp_Character) qui permet de représenter un personnage de notre jeu qu'il s'agisse du héros ou de ses ennemis, on a implémenté toute la logique et on a ajouté aussi le mech du personnage. On va utiliser la puissance de l'héritage en programmation pour créer notre héros et nos ennemis à partir de la classe mère, qui eux auront toutes les fonctionnalités du Bp_character, chacun d'eux pourra par exemple gérer ses points de vie, il aura une variable IsDead, Health Points… il aura exactement la même représentation graphique. Contrairement aux ennemis, le héros a une caméra qui va le suivre constamment, de manier à pas perdre l'action, c'est-à-dire que lorsque le joueur va bouger la caméra va rester au-dessus du héros. Nous avons ajouté des sockets pour pouvoir rattaché des armes à notre personnage.

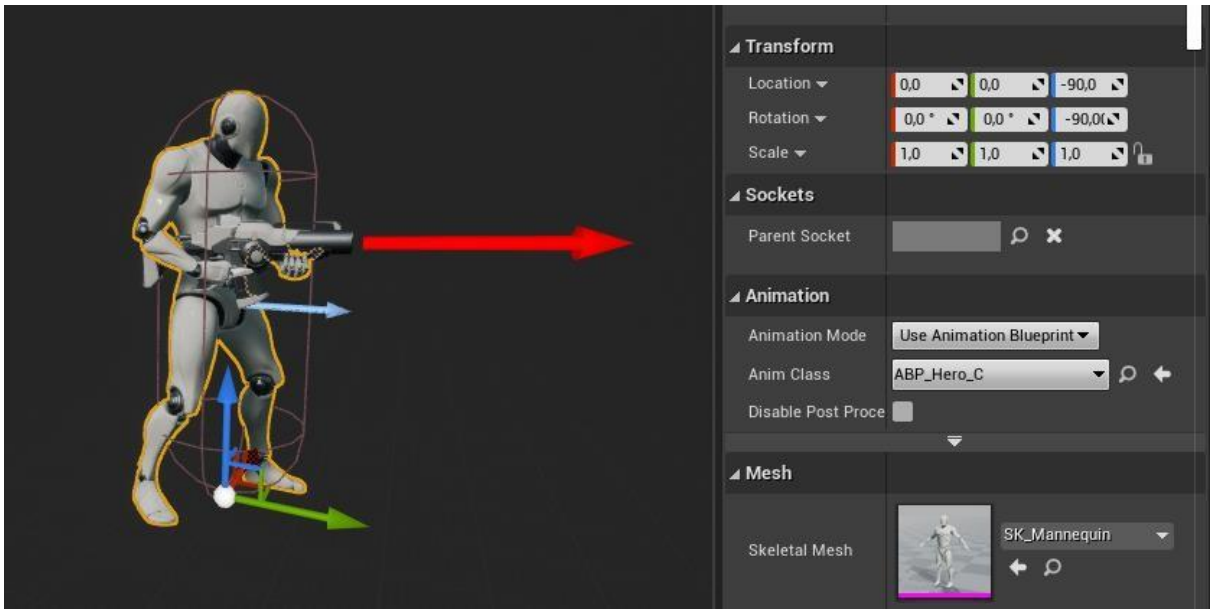

Figure 21 : Bp_Hero

❖ Création du character movement du héros

Le seul souci c'est que si jamais nous lançons le jeu actuellement le héros apparaitre bien dans le jeu mais si j'appuie sur les touches du clavier, il ne se passe rien donc cette fois si nous allons gérer les inputs, nous allons voir comment lorsqu'on va incliner le stick ou appuyer sur l'une des touches du clavier, le personnage va se déplacer, tout cela forme ce qu'on appelle « Character Movement ».

Pour gérer les inputs, nous allons ouvrir le Bp_Hero, ensuite se rendre dans l'Event Graph et c'est l'a que nous allons coder toute la logique des mouvements tout simplement. Donc là faire un clic droit, faire appel à un Evènement nommé « Axis Event : Move Up », alors qu'est-ce que cet évènement ? c'est lui qui va définir les déplacements vers le haut ou vers le bas, cet évènement va être appelé à chaque frame. C'est-à-dire qu'à chaque image, chaque rafraichissement de l'image cet évènement va être appeler et il va renvoyer un « Axis Value ». Par la suite faisons appelle à un autre évènement qui se nomme « Add Movement Input » se nœud-là ne s'applique que si l'on l'appel sur un 'Pawn'. Le Add Movement Input va ajouter des mouvements au personnage (Vitesse de déplacement, direction…). Maintenant qu'on a fait le Move Up, il ne nous reste plus qu'à faire appel à un autre nœud « Move Right » nous allons faire appel une nouvelle fois au Add Movement Input et les branché ensemble.

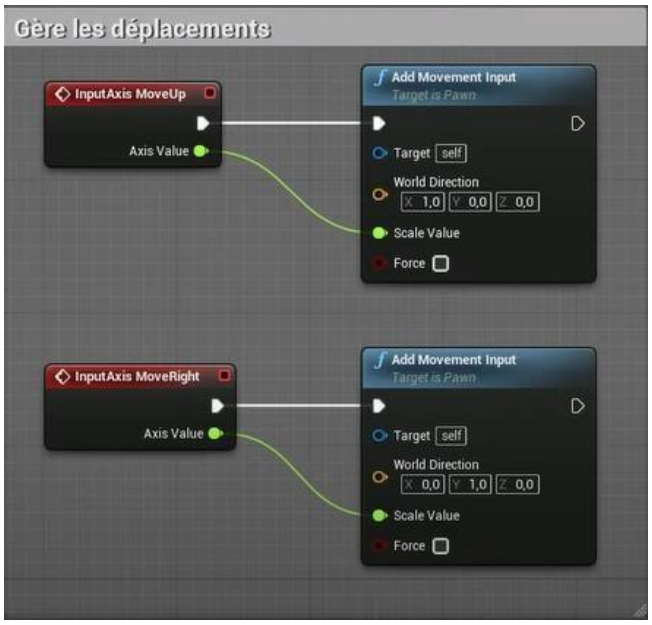

Figure 22 : Les Deplacements du Héros

Et donc là maintenant que nous pouvons déplacer le jouer dans le monde, nous allons faire en sorte de pouvoir l'orienter en fonction des inputs du player, on va dans l'Event Graph, clic droit j'appelle l'évènement « Input Axis Look Up », tirer ensuite un câble depuis le look up, créer un Branch, il faut maintenant coder une logique.

Nous allons faire appel au « Make Vector » pour ainsi définir le degré d'inclinaison du stick pour permettre d'orienter le personnage, une fois le degré d'inclinaison initialisé à 25% on va faire appel au « Set Control Rotation » pour lui demander d'appliqué une rotation à son Pawn.

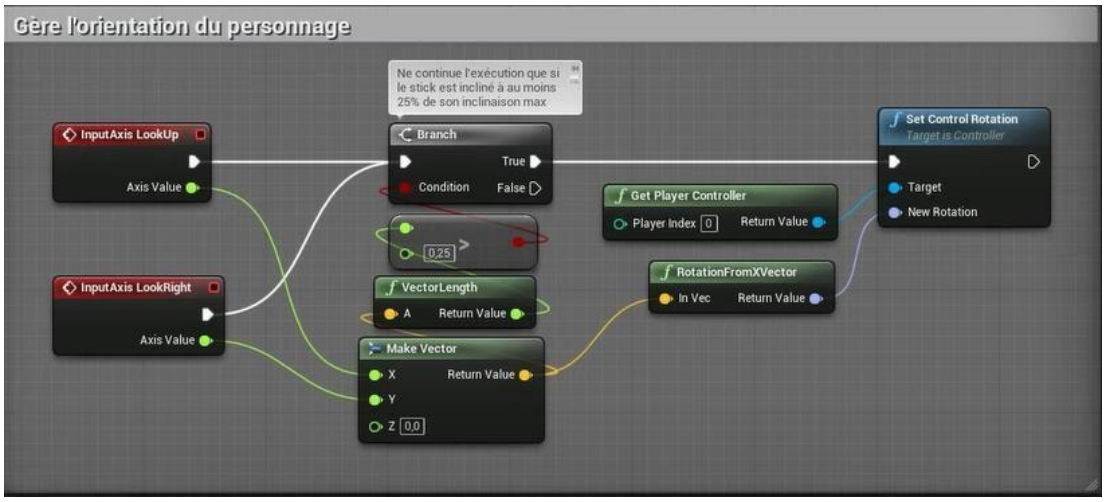

Figure 23 : Gérer l'orientation du personnage

Nous revenons dans le MainMap vous verrez qu'on peut enfin effectuer les rotations avec notre personnage. On peut passer à la suite du projet (la création des ennemis).

4.2.4 Création du Blueprint des ennemis

Tout comme le héros nous allons utiliser la puissance de l'héritage pour faire en sorte que l'ennemi hérite des fonctionnalités d'un personnage de base.

Etant donné que l'ennemi est une classe enfant de la classe Bp_Character, l'ennemi peut gérer ses points de vie et ainsi définir s'il est mort ou pas. Afin de ne pas nous perdre et confondre l'ennemi au héros on va devoir changer la couleur de l'ennemi, la couleur de l'ennemi correspond à l'habillage.

Comme on a expliqué dans la partie sur la création de la map, l'habillage est fait à partir de ce qu'on appelle des Materials, qui permettent de simuler la texture des matériaux de la vraie vie. Donc on va sélectionner le mech de notre ennemie et effectuer les mêmes manipulations expliquer plus tôt dans la création de notre projet.

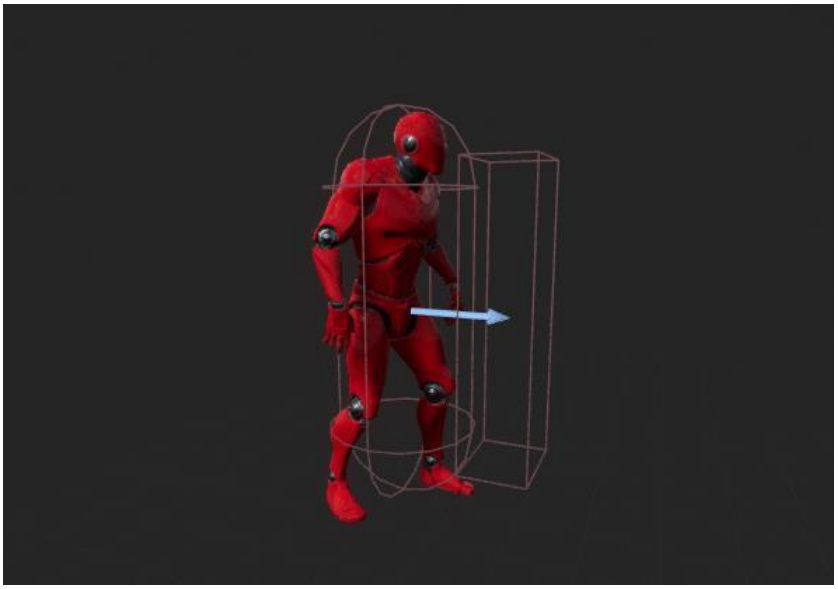

Figure 24 *: Bp_Enemy*

❖ Création d'un AI Controller : Système d'Intelligence Artificielle

Nous avons une classe de type « AI Controller », dans les chapitres précédents nous avons expliqué que le AI Controller est comme un cerveau que nous allons brancher sur le Bp_Enemy de manière à simuler une intelligence.

on va commencer à coder une petite logique, on se rend dans l'Event Graph : clic droit, Custum Event nous allons le nommer « Track Player », cette événement la quand on va faire appel à lui il va demander au Pawn qui possédé par cet AI Controller (dans notre cas ça sera l'ennemi), de récupérer la position du player et d'aller vers lui, et comment va-t-on faire ça ? on va utiliser la fonction « AI Move To », cette dernière demandera quel est le Pawn à qui on va appliquer l'opération 'Move', il suffira de faire un clic droit de faire appel au « Get Controller Pawn » et de le connecter au nœud (AI Move To).

Ensuite il va nous demander quel acteur on veut tracker, on devra donc faire référence à notre player en faisant appel au « Get Player Pawn » que nous connectons au pin d'exécution Target Actor du nœud AI Move To.

Pour que l'ennemi n'oublie jamais sa mission nous allons ajouter à notre système d'Intelligence Artificielle un Set Timer by Event et activer « le looping », ainsi pour lui rappeler toutes les secondes qu'il doit continuer la Track. Pour faire en sorte que cela se lance dès le début du jeu, connectons tout ça au nœud Event Begin Play.

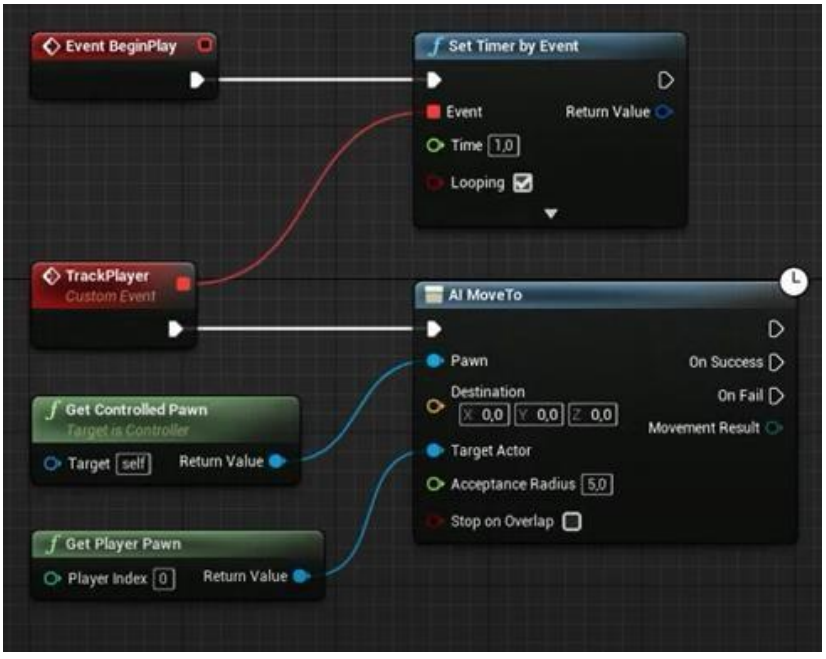

Figure 25 : AIC_Enemy

4.2.5 Création du projectile de l'arme du héros

Pour notre projectile on va créer un blueprint de type Actor, pourquoi Actor et non pas Pawn par exemple ? c'est tout simplement parce que le projectile ne peut pas être contrôlé, c'est juste un objet qui va être placer dans le monde et va suivre sa trajectoire, dès qu'il va toucher quelque chose il va se désintégrer, si cette chose est un ennemi on va lui faire des dégâts et ainsi de suite. Nous appellerons se nouveau Blueprint « Bp_Projectile ».

On Lui ajouté ensuite une sphère collision afin de pouvoir faire des dégâts au ennemis a chaque fois qu'ils seront toucher.

Pour lui implémenté les propriétés d'un projectile telque donner une vitesse a la sortie du canon nous lui avons ajouté un component de type (projectile Movement).

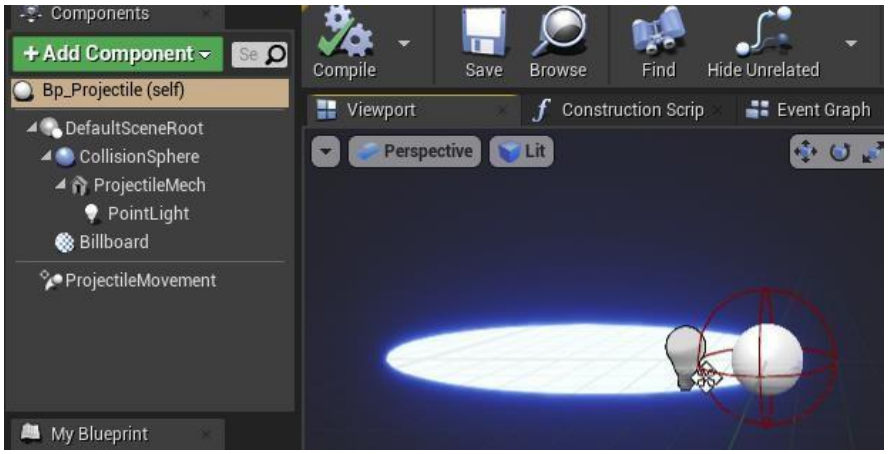

Figure 26 : Bp_Projectile

4.2.6 Création du Blueprint de l'arme du héros

Le Gun , créer a partir d'un blueprint de classe Actor, nous avons ajouté un skeleton Gun Mech a son capsule component.

Et pour terminer nous avons ensuite ajouté un Arrow components afin définir l'emplacement du canon de l'arme.

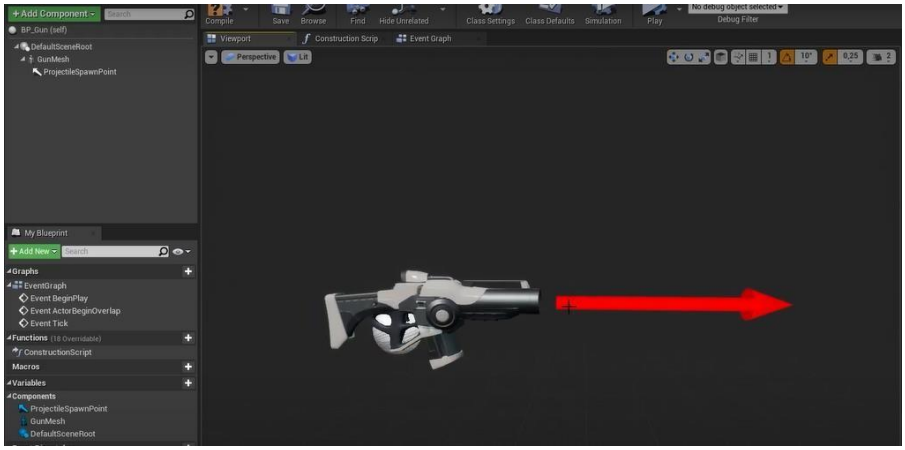

***Figure 27** : Bp_Gun*

L'évènement Fire Projectile va nous permettre de faire apparaitre un projectile au niveau du projectile Spawn Point. Le Start Firing va faire appeler à la fonction Fire Projectile de manière répéter, c'est lui qui va gérer la cadence de tire. Le stop Firing va permettre d'arrêter le mécanisme que nous aurons lancer (Tire).

Nous allons commencer par implémenter l'évènement Fire Projectile, pour cela on tire un câble depuis son pin d'exécution et faire appelle au nœud « Spawn Actor From Class », ce nœud va permettre de faire apparaitre un acteur dans le monde, il faudra alors lui renseigner la classe de l'acteur qu'on veut faire apparaitre, dans notre cas c'est le (Bp_Projectile), on renseigne ensuite sa position à travers le pin Spawn Transforme.

***Figure 28** : Faire apparaitre un projectile au bout du canon*

Alors là nous avons finir d'implémentation de la logique qui permet de faire apparaitre un projectile sur le bout du canon. Cette fois-ci nous allons nous attarder sur le start Firing. Sur le pin d'exécution de ce dernier tirer un câble et faire appel à un « Do once », c'est comme un passage qui va laisser passer le pin d'exécution une seule fois, ensuite faire appel au « Set Timer By Function Name », on va renseigner le nom de l'évènement à l'intérieur du Function Name, activer le Timer et le « looping ». On va paramétrer ensuite le nombre de projectile que l'on veut tirer à la seconde.

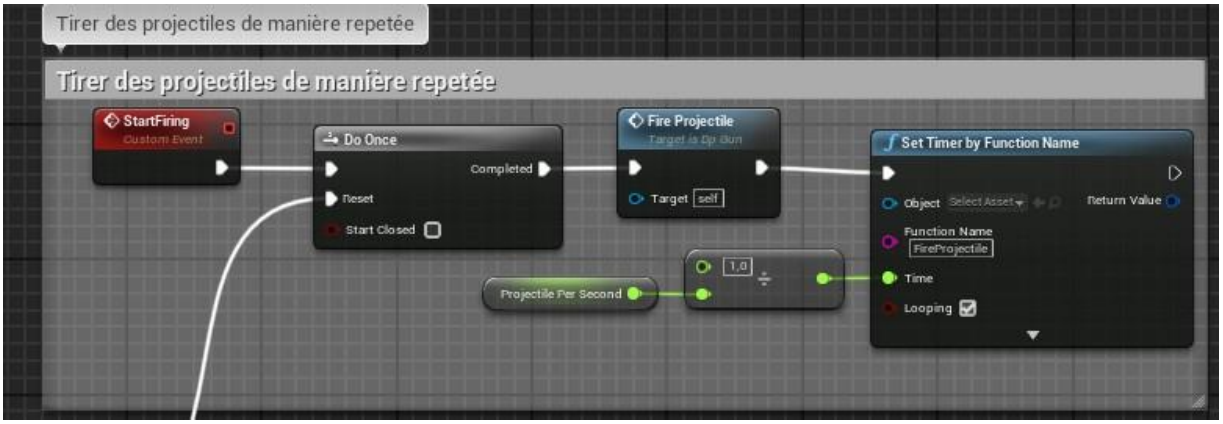

Figure 29 : Tire des projectiles

4.2.7 Infliger des dégâts aux ennemis

De retour dans notre projet Unreal, si vous vous rappelez bien dans la classe Bp_Character, nous avons implémenté les fonction Update Health Points qui permet de gérer les points de vie d'un personnage de notre jeu et Update Is Dead qui permet de gérer la mort d'un personnage. Nous ce qu'on veut c'est pouvoir faire appel à ces fonctions à chaque fois qu'un projectile va toucher l'ennemi, et pour cela on va utiliser ce qu'on appelle les « Blueprint Interfaces », une interface permet à deux blueprint de communiquer entre eux sans forcément savoir de quelle classe est l'autre blueprint. On va juste crée une interface, et cette interface va juste contenir des sortes de promesses, on va renseigner dans cette interface qu'il y a une fonction qui va s'appeler (Faire des dégâts), ensuite cette interface là on va la brancher dans l'ennemi et dans le décor par exemple, et c'est eux qui vont devoir implémenté la fonction « faire des dégâts », résultat lorsque le projectile va toucher quelque chose on ne va plus chercher à comprendre s'il s'agit d'un ennemi ou d'un décor, on va juste chercher à comprendre est-ce que cet objet-là que nous avons touché possède l'interface si c'est le cas nous pourrions lui faire des dégâts.

Maintenant que l'interface a été créée, nous allons pouvoir la brancher sur un ennemi pour qu'il puisse recevoir des dégâts. Ouvrir le Bp_Enemy, et la si vous allez dans class setting vous avez (class option, Blueprint option, et interfaces). Ce qui nous intéresse c'est cette partie-là, dans cette partie on a « Implemented interface » qui permet de savoir qu'elles sont les interfaces qui ont été branchées sur ce blueprint, pour l'instant on n'en a pas et donc on va ajouter la nôtre, cliquer sur « Add » et taper BPI et sélectionner « BPI_Damageable »

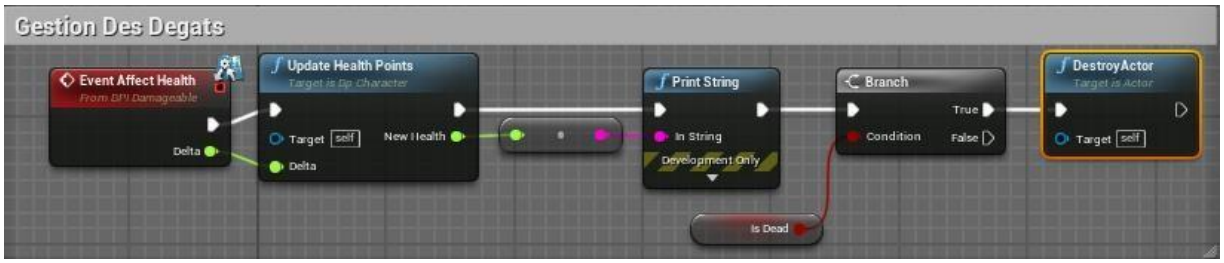

Figure 30 : Gestion des Dégâts

Nous allons passer à la partie qui va permettre de faire appel à cette logique directement depuis le projectile lorsqu'il va toucher un ennemi. Ouvrir le Bp_Projectile, Et comment est-ce qu'on va vérifier s'il touche un ennemi ? on va faire appel à la fonction « Event Actor Begin Overlap », qui est proposé par défaut dans l'Event Graph, le Begin Overlap va être appeler à chaque fois que notre projectile va toucher un autre acteur,

C'est la collision sphère qui va définir si on entre en contact avec un ennemi, un mur ou autre chose. On va référencer l'objet toucher par le projectile grâce au nœud « Doe simplement Interface », une fois que c'est fait appeler à la fonction Affect Health. A ce stade là l'ennemi qui est passé dans la fonction Actor Begin Overlap, c'est juste un acteur qui implémente l'interface BPI_Damageable donc il peut recevoir des dégâts. On va le détruire avec le Destroy Actor.

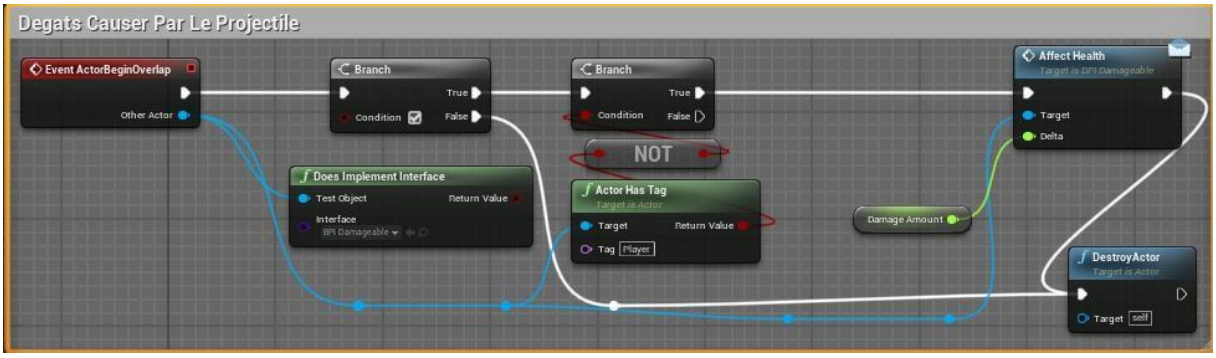

Figure 31 : Dégâts causés par le Projectile

4.2.8 Infliger des dégâts au héros

Dans le point précédent nous avons fait en sorte de pouvoir faire des dégâts à l'ennemi, cependant si on lance le jeu on a un petit souci, si l'ennemi touche le player le héros ne reçoit pas de dégâts, donc nous allons mettre en place ce système-là qui va permettre de faire en sorte que le player reçoive des dégâts lorsqu'il est touché par l'ennemi. Et qu'il puisse aussi mourir lorsqu'il n'a plus de point de vie.

Donc là maintenant que le héros peut recevoir des dégâts nous allons mettre en place la logique dans l'ennemi qui va permettre d'infliger des dégâts au héros. Venir dans le Bp_Enemy, une fois dans le Viewport nous allons lui ajouter un nouveau component « Box collision », comme son nom l'indique c'est une boite de collision. Ce que l'on va faire c'est

que lorsque les ennemis vont se rapprocher du héros et à chaque fois qu'il va se retrouver dans cette boite de collision on va lui faire des dégâts.

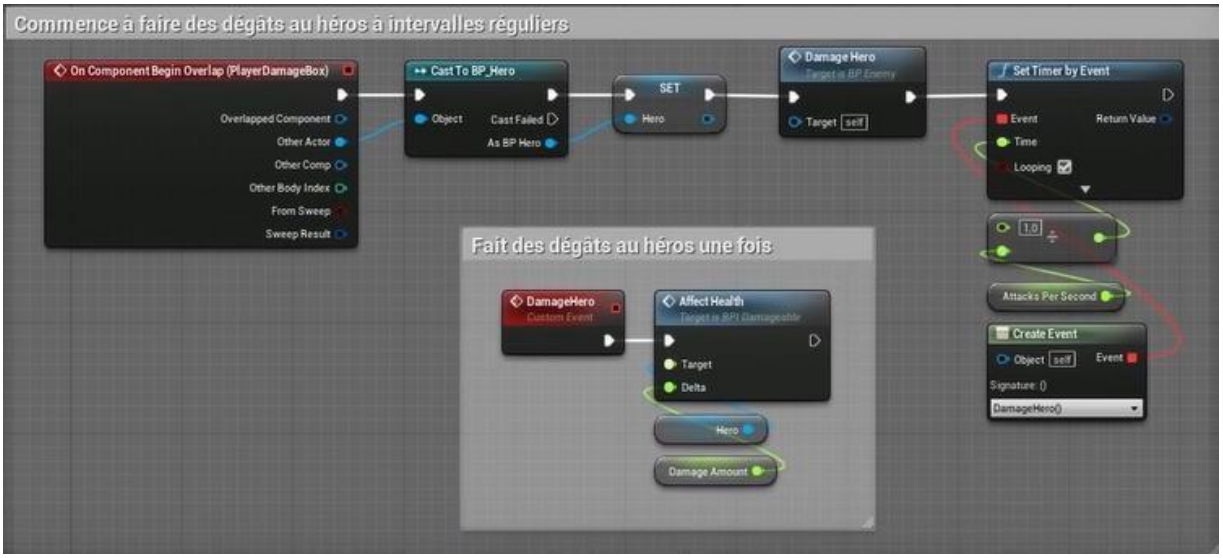

Figure 32 : Infliger des dégâts au Héros

4.2.9 Mort du héros et système de réapparition

Précédemment nous avons mis en place la possibilité de faire des dégâts depuis l'ennemi vers le héros. Cependant lorsque le héros meurt, on a la caméra qui se détache du héros et dans la majorité des jeux c'est à cet instant que l'interface du Game Over est lancer puisqu'on ne peut plus jouer à partir de ce moment-là. Dans cette partie nous allons utiliser le principe du Respawn couramment utilisé sur les projectiles d'une arme et appliquer cette logique pour permettre de faire réapparaitre le héros dans la zone de départ lorsqu'il va perdre la totalité de ses points de vie, aussi le joueur pourra continuer de jouer et essayer d'obtenir un meilleur score.

Nous, ce qu'on veut mettre en place c'est une règle qui dit qu'à chaque fois que le player va avoir ses points de vie qui vont tomber à zéro, plutôt que de le détruire avec un (Destroy Actor), on veut le faire réapparaitre dans la zonz Safe, et si vous vous rappelé bien nous avons parlé du GameMode qui lui est responsable des règles du jeu, nous avons créé un Bp_GameMode plus tôt, donc nous allons l'ouvrir et nous allons implémenter la logique qui va permettre de faire réapparaitre le player. Une fois dans l'Event Graph faire un clic droit : Custom Event, pour créer un évènement nous allons l'appeler « Respawn Player », ensuite faire appel à une fonction qui se nomme « spawn Actor From Class » cette fonction-là, si vous vous rappelez nous l'avons utilisé plus tôt lorsque nous avons implémenté la logique du Gun pour faire apparaitre le projectile sur le canon de l'arme. Nous avons la variable « Spawn transform » qui va devoir être peuplé avec la position de départ du joueur de manier à faire réapparaitre le joueur à cette endroit-là à chaque fois qu'il meurt. Une fois tous les nœuds connectées, compiler, sauvegarder et on ferme le GameMode

Figure 33 : Respawn du Player

Alors là maintenant nous allons implémenter dans le Bp_Hero une logique qui va permettre d'envoyer sa position de départ au GameMode, il va exécuter le code pin par pin. Nous codons ensuite la logique qui permet d'envoyer la position au GameMode à partir du second pin d'exécution

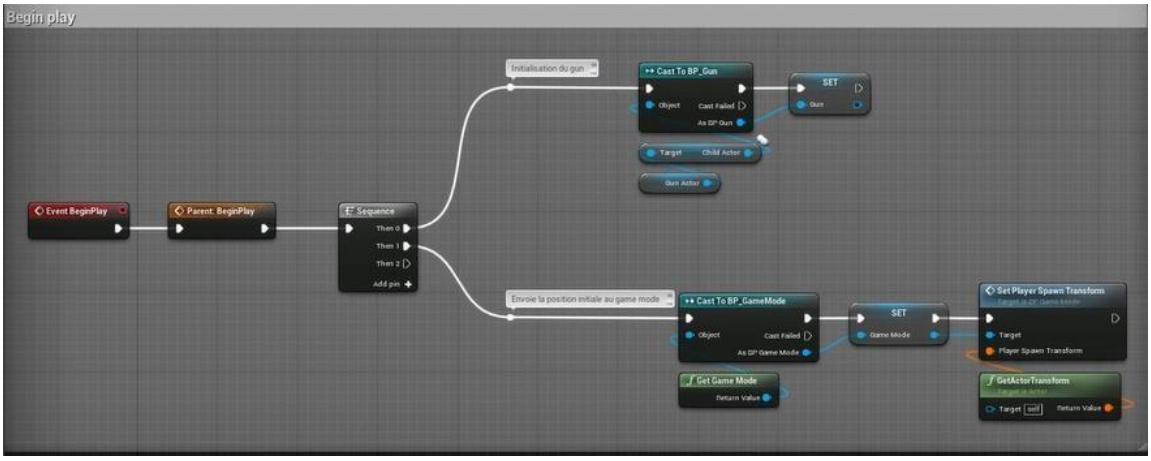

Figure 34 : Le Begin Play

Nous avons fini d'implémenter toute la logique qui va permettre de gérer le Respawn du player il ne reste plus qu'à faire appel à toute cette logique dans le « Affect Health ». Faire appel au Get GameMode, et au Respawn Player que nous connectons au (Branch) juste avant le Destroy Actor. In ne me reste plus qu'à compiler et sauvegardé

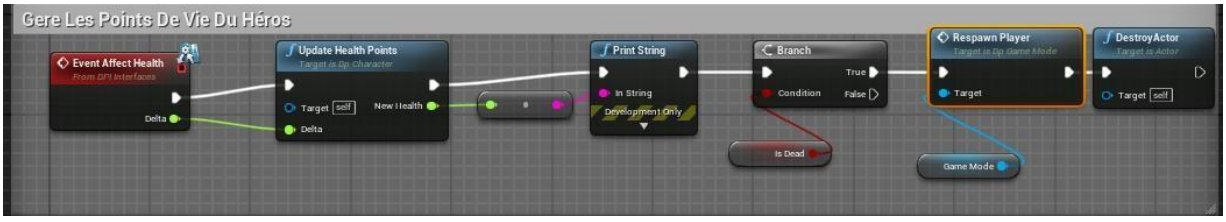

Figure 35 : Gere les points de vie du Héros

4.2.10 Lieux d'apparition aléatoire des ennemis sur la map

Nous allons nous attarder cette fois-ci sur le système d'Intelligence Artificielle charger de l'apparaitre les ennemis de façons aléatoire sur la Map et ce de manière régulière.

On va placer une sorte de volume au-dessus de la zone de jeu, et ce volume-là va permettre de tirer aléatoirement des positions à l'intérieur du volume pour ensuite faire apparaitre des ennemis qui vont tomber directement dans la zone de jeu.

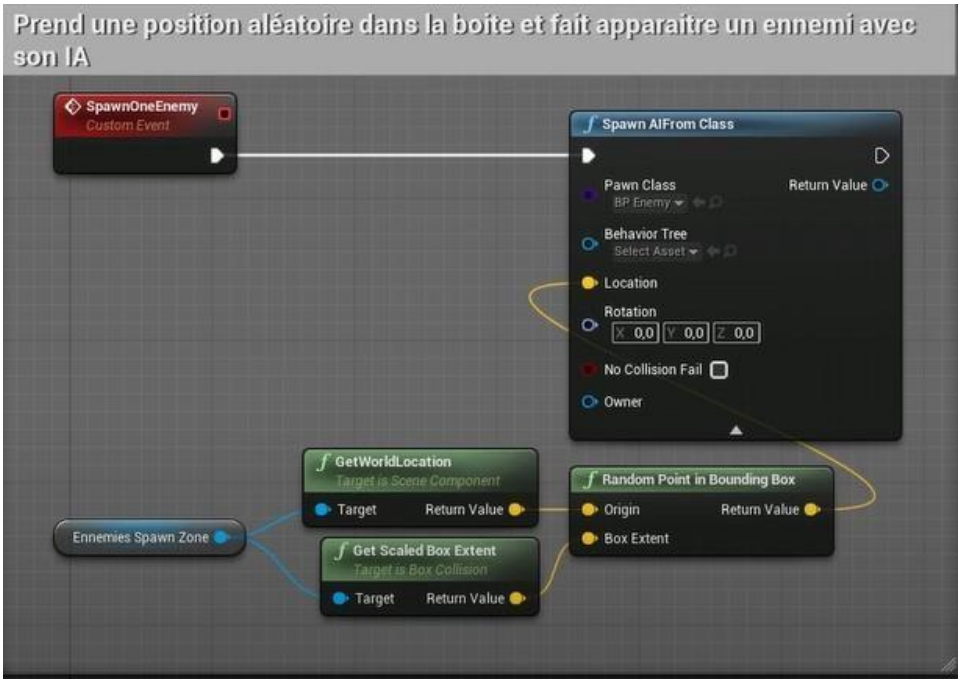

Figure 36 : Spawn Enemy

Maintenant que nous avons implémenté la logique qui permet de faire apparaitre un ennemi, nous allons mettre en place une logique qui va permettre de faire apparaitre plusieurs ennemis de manière répétée. Pour cela on va utiliser un Set Timer. Du coup plutôt que de lancer un Timer directement dans le spawn, nous aimerons bien qu'on puisse contrôler le nombre d'ennemis qu'il y a sur la map. Pour faire ça on va utiliser le Bp_GameMode puisque c'est lui qui va gérer justement les règles. On va créer une interface qui va nous permettre de communiquer directement avec le GameMode.

Dans le content browser faire un clic droit : Blueprint> Blueprint interfaces nous allons l'appeler « BPI_GameModeMP », pour rappel le blueprint interfaces permet de faire communiqué des blueprint entre eux sans forcément connaitre leur classe, l'ouvrir, on va faire apparaitre une première fonction qui va s'appeler « Register Spawner », ensuite lui ajouter un input, et dans nous allons ajouter un objet de type Bp_Spawner, compiler et sauvegarder.

Rendons dans le Bp_GameMode ajouter l'interface, créer un (Event Register Spawner) qui va être appelé à chaque fois, nous faisons ensuite appeler à un Set Timer By Event. Il va permettre de faire trois ennemis à la seconde.

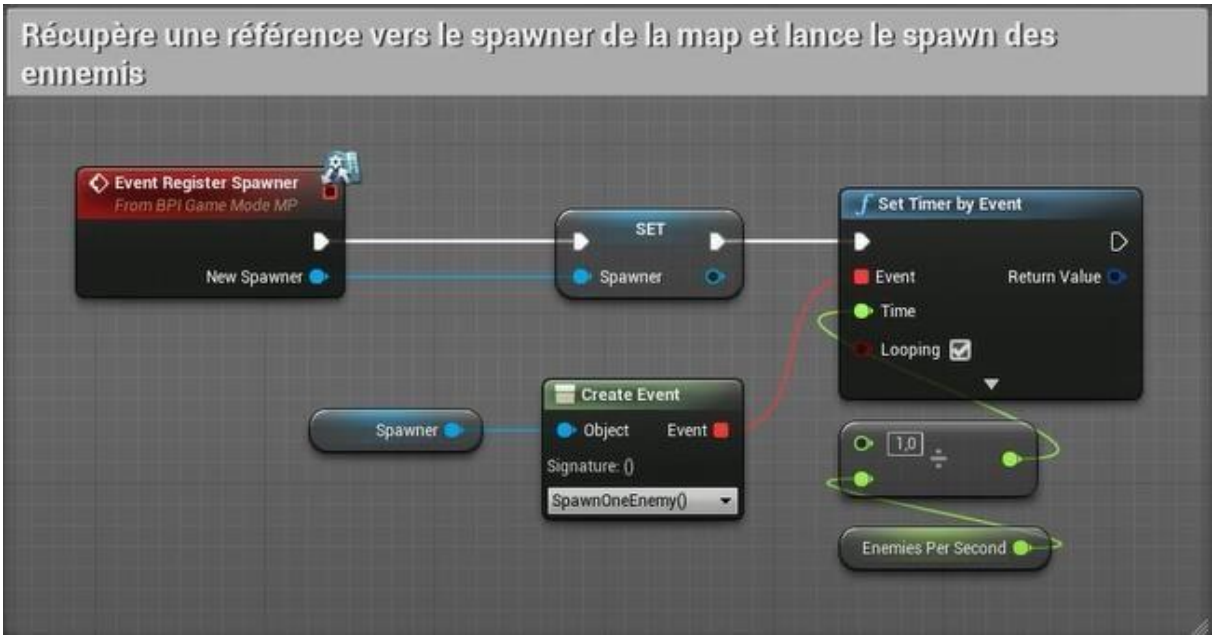

Figure 37 : Récupère une référence vers le Spawner de la map et lance le spawn des ennemis

4.3 Animation du héros

4.3.1 Création d'un Blend space

Le Blend Space est un espace semblable à un time line qui contient une grille avec plusieurs points qui vont permettre de répertorier différentes animations du personnage auquel il est assigné, donc il faudra choisi le personnage à qui est destiné ce Blend Space dans notre cas c'est le Sk_mannequin. Nous allons renommer notre Blend Space en « BS_HeroJog ».

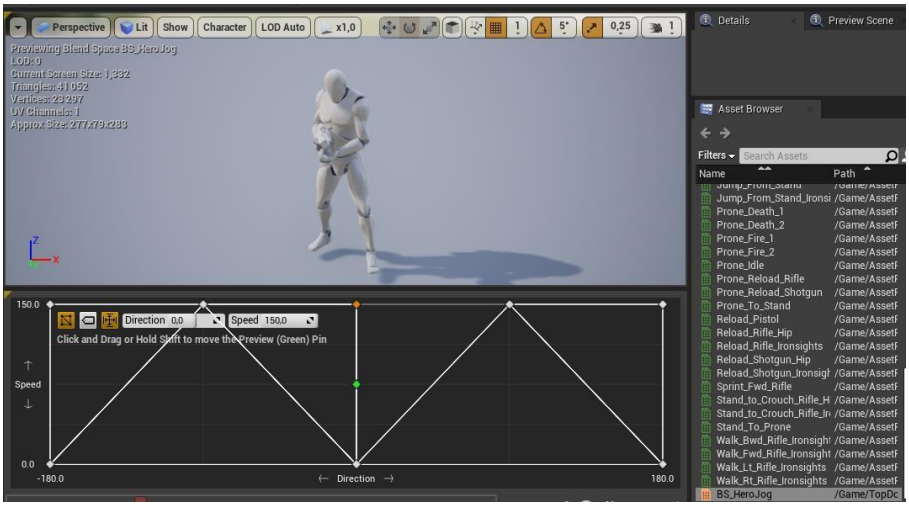

Figure 38 : Animation Blend Space

4.3.2 Création d'un animation Blueprint

Ici nous voulons crée un nouveau type d'Asset à savoir les « Animations Blueprint », un (animation blueprint) est un blueprint donc le seul but sera de définir quelle animation du héros jouer en fonction de ce qu'il sera entrain de faire. Donc dans le content nous allons créer notre animation blueprint, et pour cela clic droit : Animation Blueprint, nous allons choisir le bon squelette auquel l'assigner (SK-Mannequin), cliquer sur ok et nous allons l'appeler (ABP_Hero), l'animation blueprint du Hero enfin crée.

Les deux endroits où on va le plus travailler c'est l'animation Graph et l'Event Graph. C'est dans ce dernier qu'on va récupérer toute la logique chez le player, l'Event Graph va être appeler à chaque Frame et on va pouvoir par exemple récupérer la vitesse du player, la direction…

On va stocker tous ces informations dans des variables qu'on va ensuite utiliser dans l'Anim Graph. Et une fois dans l'animation graph on va pouvoir prendre des animations dans la fenêtre de droite on va les placées et crée une certaine logique et à la fin on va brancher notre logique dans le nœud (Output pose).

Sur la fenêtre de gauche nous avons le mech sur lequel est appliqué l'animation blueprint. Un (animation blueprint) est unique à chaque (Skeleton), donc l'animation qu'on va créer ici ne fonctionne que sur les objets qui ont le même skeleton.

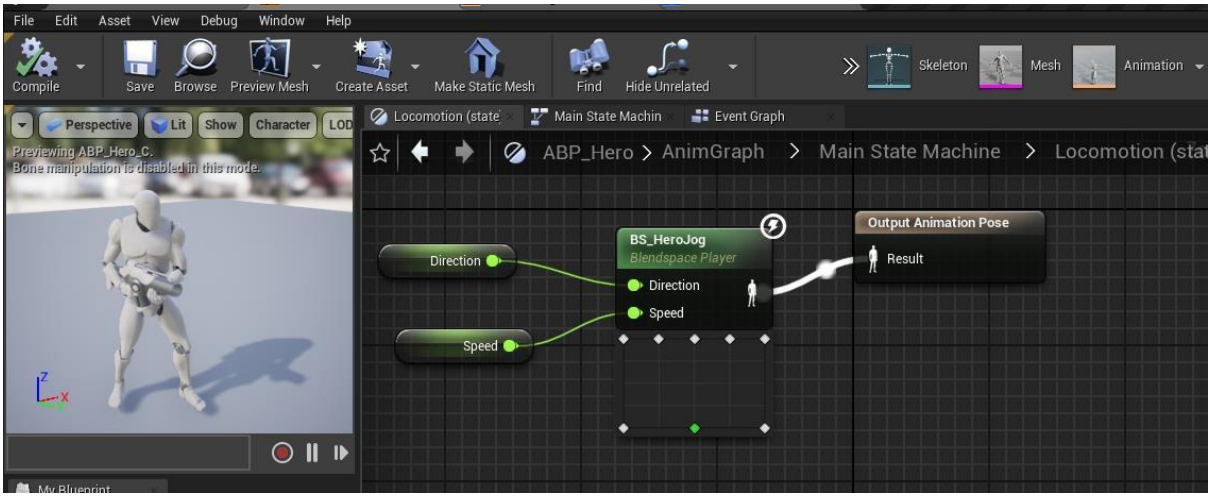

Figure 39 : Animation Blueprint (ABP_Hero)

4.3.3 Guide audio

Dans cette partir nous allons nous occuper de tout ce qui est lier au son, nous allons voir comment jouer des bruitages, de la musique de fond, comment faire de la spécialisation etc…

Nous commençons par rassemblés tous nos fichier sons qui nous seront nécessaires et que l'on aimerait emporter dans notre jeu. Attention n'importé que des sons libres et sans droit d'auteur au risque de s'exposé aux poursuites judiciaires. Une fois que nos sons collectés, on colle le dossier audio à l'intérieur du « Content » de notre jeu sur lequel nous somme entrain de travailler. Pour la création de la musique qui sera jouer IN Game

Unreal Engine propose une fonctionnalité qui permet de transformer des fichiers sons importer en fichier de type (Sound Cue). Nous avons pu ensuite spécialiser notre cue ainsi obtenu sur tout l'étendu notre Level.

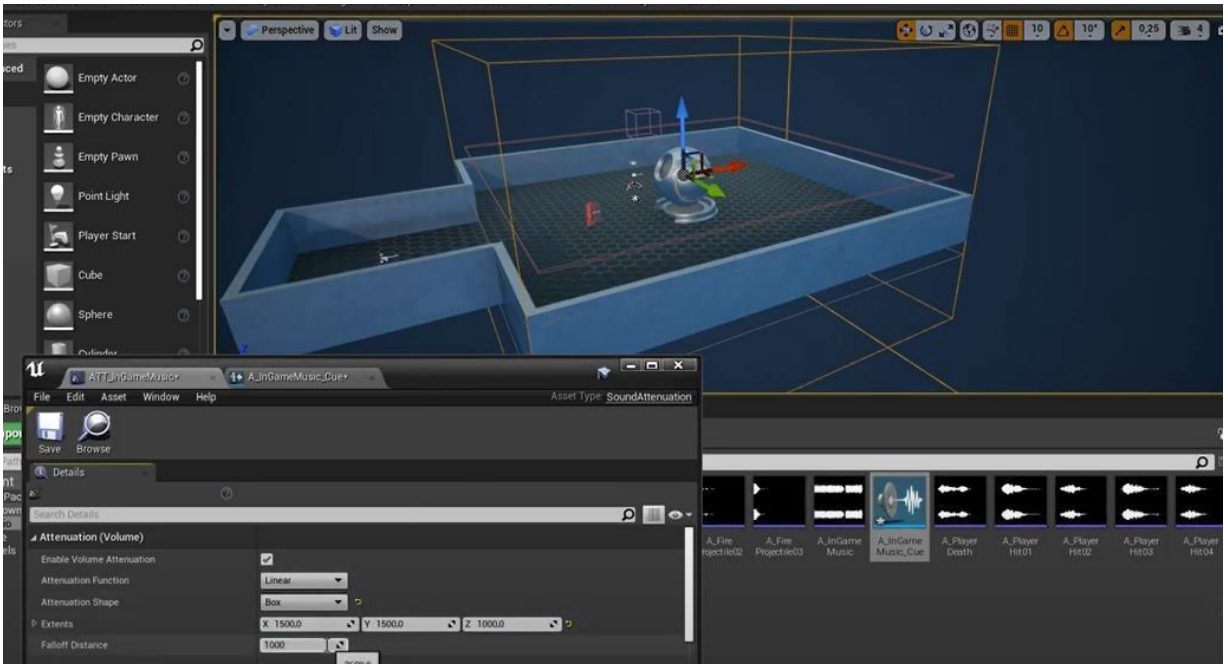

Figure 40 : Spatialisation d'un Sound Cue

4.3.4 Création de l'interface utilisateur

Nous allons voir cette fois-ci comment créer notre première interface utilisateur qui va nous permettre d'afficher les points de vie à gauche et le score du joueur à droite. On va créer des interfaces utilisateur en utilisant UMG. Pour commencer nous allons aller dans le dossier Top Down Shooter, ici faire un nouveau dossier que nous allons l'appeler UI (User Interface),

Cet éditeur est un éditeur visuel qui permet de construire des éléments d'interfaces utilisateur tels que le menu, les HUD qu'on a en jeu, ou tout un tas d'autres éléments graphiques que vous aimeriez présenter aux joueurs. L'éditeur UMG a enfaite deux modes : le mode désigner qui permet en gros de placer les objets sur la fenêtre Viewport, c'est là que nous allons agencer principalement les éléments graphiques. Que nous voulons présenter à l'écran et nous avons aussi un mode graph, qui va permettre de créer la logique qui fera réagir les éléments visuels en fonction de ce qui se passe dans votre jeu.

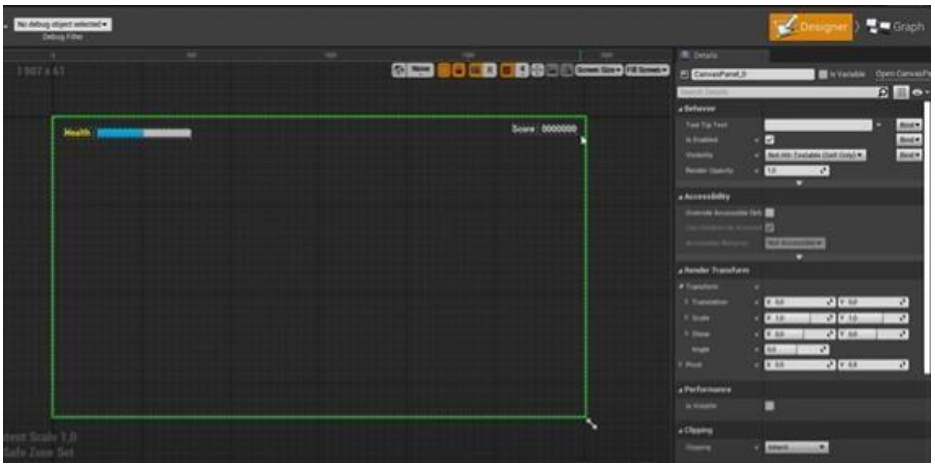

Figure 41 : User Interface

Notre jeu ainsi terminé il ne reste plus qu'à créer un fichier exécutable pour pouvoir le partager avec nos amis.

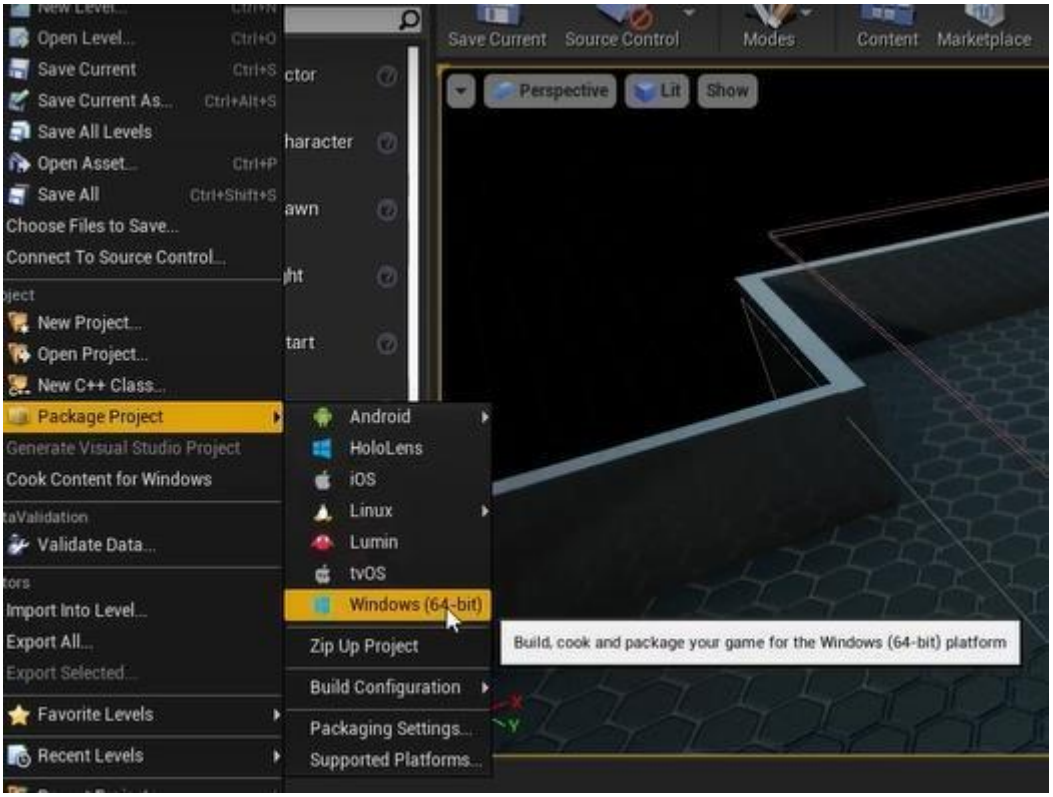

Figure 42 : Création d'un executable

4.4 Conclusion

En conclusion l'Intelligence Artificielle de jeu vidéo fonctionne comme un système de type conscience fourmilière, où chaque individu possède sa propre conscience et tous sont reliés par une conscience collective d'aide à la prise de décision appeler GameMode.

Le GameMode est le chef d'orchestre dans le jeu vidéo, c'est lui qui est chargé de gérer toutes les règles au cœur d'un jeu, c'est le système principal chargé de commander toutes les intelligences artificielles, c'est lui qui décide qui a gagné ou perdu, qui est chargé du décompte du score… sans lui il n'y a pas de jeu, nos IA ne serait que des boites de conserves.

Conclusion et perspectives :

Nous avons découvert que dans un jeu vidéo tous les Blueprints malgré leur indépendance à première vue étaient lier les uns aux autres soit pas une variable soit par une fonction, nous avons utilisé la force de l'héritage en programmation à fin d'éviter de coder plusieurs les fonctionnalités communes entre nos personnages.

En réalisant ce mémoire et grâce aux recherches effectuées ainsi qu'aux réflexions que nous nous sommes faites, nous avons pu gagner en maturité sur notre façon d'aborder les choses. Ce travail étant avant tout un travail de recherches, nous avons dû procéder différemment de ce que nous avions appris à faire par nous-mêmes dans le passé.

Nous nous sommes ainsi documentés sur des textes de différentes natures (articles, livres, documents de recherches), nous avons questionné notre entourage pour confronter nos idées et nos croyances afin de développer un argumentaire et nous avons également cherché des interlocuteurs qui portent un intérêt pour ce domaine afin qu'ils puissent nous conseiller, nous aiguiller ou même nous corriger si le besoin se faisait sentir.

Nous avons découvert comment était fait le système qui permettais de faire apparaitre des projectiles sur le canon du Gun, et pour rendre notre jeu plus complexe nous avons réadapté ce système de Respawn à notre héros afin qu'il puisse réapparaitre dans sa zone Safe après sa mort sans toutefois lancer le widget de Game Over.

Mais cette méthode posait par ensuite plusieurs problèmes, comme le fais que le héros n'avait plus le temps de jouer son animation de mort, le son de mort et de réapparition finissaient par être superposés, notre héros était tout suite détruit après sa mort et réapparaissait du même coup car la fonction « Spawn Actor From Class » était directement lancé. Pour résoudre ses problèmes nous avons utilisé la fonction « Delay » disponible sur Unreal Engine, ainsi on permettait au héros d'avoir le temps de jouer son animation de mort, le son de mort avant que la fonction Destroy Actor n'entre en jeu pour détruit notre héros complétant ainsi le cycle précédent le Respawn de notre joueur.

Nous avons aussi remarqué que le système d'intelligence artificielle avait besoin d'un Navigation Mech sur la map pour pouvoir se mouvoir, ce dernier permettant ainsi de délimiter la zone de déplacement des IA.

Dans la majorité des jeux vidéo triple-A (ps : Dans l'industrie du jeu vidéo, AAA (prononcé « triple A ») ou Triple-A est un terme de classification utilisé pour les jeux vidéo dotés des budgets de développement et de promotion les plus élevés les joueurs et la critique s'attendent à ce qu'un titre considéré comme étant de type AAA soit un jeu de grande qualité ou figure parmi les jeux les plus vendus de l'année.) le système d'Intelligence Artificielle est doté d'organe de sens appelé Pawn Sensing, subdivisé en 2

On See Pawn : pour la vue, et On Hear noise pour l'ouïe. Qui sont des éléments qui sont lancer à chaque que l'ennemi verra ou entendra les sons émis par le héros pendant ses déplacements comme ça une fois le héros repéré la traque pouvait débuter. Majoritairement utilisé dans les jeux d'infiltrations telles que Splinter cell, Farcry …

Dans notre jeu à nous, nous n'avons pas doté nos IA de Pawn Sensing car notre jeu n'est pas un jeu de ce type premièrement, et deuxièmes notre map n'est pas assez grande pour pouvoir se cacher donc au lancement de notre jeu le héros a déjà été repéré par ses ennemis, rendant ainsi l'option Pawn Sensing superflue et inutile. Nous nous sommes juste contentés d'utiliser un « AI Move To » pour renseigner à l'ennemi qu'il devait pourchasser le héros dès le lancement du jeu, et pour qu'il n'oublie jamais quelle était sa mission nous avons combiné cela à un « Set Time By Event » qui était chargé de lui rappeler à chaque seconde l'ordre de sa mission (la traque perpétuelle du héros jusqu'à son extermination).

Nous avons aussi établi qu'afin d'éviter de surcharger le système d'Intelligence Artificielle principale (Le Game Mode) avec des informations superflues, ont jugé mieux de créer un blueprint interfaces (BPI_GameModeMP) qui serait chargé de récolter les informations de secondes zone et de rapporté un résumé au Game Mode par message privé.

En ce qui concerne l'avenir de l'IA dans les jeux vidéo, il est clair que l'IA a largement contribué à améliorer l'univers des jeux vidéo. Elle a commencé par permettre aux gamers de jouer en mode solo et a fini par créer des personnages presque aussi intelligents que les humains.

D'autre part, l'utilisation des différents outils d'IA a permis de faire évoluer tous les aspects, à savoir les personnages, le gameplay, les graphismes, etc. En d'autres termes, l'évolution de la technologie a influencé l'utilisation de l'Intelligence Artificielle dans le monde des jeux vidéo.

Ainsi se clôture notre travail sur la thématique « Application de Intelligence Artificielle dans le développement du jeu vidéo ».

RESUME

L’objectif du présent mémoire est de montrer l’application de l’Intelligence Artificielle dans le développement du jeu vidéo. A travers l’illustration d’un jeu de type Top Down Shooter et de nos recherches, nous avons pu mettre en lumière que l’Intelligence Artificielle dans un jeu vidéo ne représentait pas juste un personnage, mais plutôt un réseau, on parlait ainsi d’un système d’intelligences artificiels. Un système de type conscience fourmilière ou chaque individu procède sa propre conscience et tous son relier par une conscience collective d’aide à la prise de décision. Chaque IA ne peut accomplir uniquement la tâche pour laquelle elle a été conçu, une intelligence artificielle crée uniquement pour ouvrir une porte sera incapable d’ouvrir une fenêtre peu importe la simplicité de la tâche (c’est le principe du Finite State Machine).

ABSTRACT

The objective of this thesis is to show the application of Artificial Intelligence in video game development. Through the illustration of a Top Down Shooter type game and our research, we were able to highlight that Artificial Intelligence in a video game was not just a character, but rather a network, we were talking about a system of artificial intelligences. A system of ant-like consciousness where each individual has its own consciousness and all are linked by a collective consciousness to help them make decisions. Each AI can only accomplish the task for which it was designed, an artificial intelligence created only to open a door will be unable to open a window no matter how simple the task (this is the Finite State Machine principle).

Bibliographie

[1] Brian Schwab, *The Psychology of Game AI*, Cengage Learning, 2015

[2] Ian Millington et John Funge, Artificial Intelligence for Games, Burlington, MA, Morgan Kaufmann, 2009

[3] Olivier Lejade et Mathieu Triclot, La Fabrique des jeux vidéo: Au cœur du gameplay, Paris, La Martinière, octobre 2013

[4] Kate Crawford, Atlas of AI: Power, Politics, and the Planetary Costs of Artificial Intelligence, Yale University Press, 2021

[5] Gossellin De Bénicourt Grégory, Les cahiers d'Unreal Engine 4 T1: Modélisation, Blueprints, Matériaux et Paysages, Graulhet, Éditions Graziel, 2015.

[6] Austin Grossman, Postmortems from Game Developer, Taylor & Francis, 2003.

[7] Jason Gregory, Game Engine Architecture, CRC Press, 2009.

[9] Jason Busby, Zak Parrish et Jeff Wilson, Mastering Unreal Technology: Volume I: Introduction to Level Design with Unreal Engine 3, Pearson Education, 2009

RENSEIGNEMENTS SUR L'AUTEUR

Nom : MEFIRE FAGOUOP

Prénoms : Junior Cardin

Téléphone : +261 34 60 540 48

E-mail : Cardinmefire@yahoo.fr

TITRE DU MEMOIRE :

INTELLIGENCE ARTIFICIELLE : APPLICATION DANS LE DEVELOPPEMENT DU JEU VIDEO

Nombre de page : 68

Nombre de figure : 42

Mots clés : Blueprint, Game Mode, PNJ