

Réalité augmentée 3D avec Hololens pour simulation d'un scénario de crise

Technologies de l'information et de la communication (TIC)
Filière informatique (INFO)
IICT



HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Simon Baehler

<https://github.com/simon-baehler/ARshopShooter>

Ehrensberger Juergen, Responsable
Dr Arash Salarian, Expert

Yverdon-les-bains, HEIG, 2017

Remerciements

Si deux chemins s'offrent à toi
prends le plus difficile.
— David Belle

Un grand merci au professeur Ehrensberger Juergen : pour m'avoir offert la possibilité de faire ce projet et pour le suivi durant tout le long du projet...
A Julien Amacher : Pour la relecture la relecture attentive...

TRAVAIL DE BACHELOR 2016 - 2017

Réalité augmentée 3D HoloLens pour la simulation d'un scénario de crise

Institut IICT

Énoncé

Contexte

Ce TDB fait partie d'un projet plus large qui vise à démontrer l'utilité de visualisation 3D pour la simulation de scénarios de crise et la formation du personnel de la police pour l'intervention. Une telle simulation s'effectuerait dans un espace spécialement équipé de capteurs qui mesurent la position d'une ou plusieurs personnes à l'intérieur de l'espace et détectent leur orientation et posture. A l'aide d'un dispositif de réalité augmentée (par exemple Microsoft HoloLens) les personnes peuvent visualiser l'environnement de manière réaliste tout en tenant compte de la position/orientation du porteur ainsi qu'éventuellement d'autres personnes présentes.

Objectif global

L'objectif de ce projet est la réalisation d'une visualisation interactive 3D d'une intervention par la police. La visualisation utilisera un masque de réalité augmentée Microsoft HoloLens ainsi qu'une plateforme de visualisation 3D temps réel, Unity 3D.

Cahier des charges

Le projet comprend les étapes suivantes:

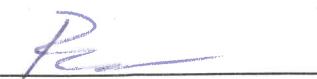
- Etablissement de la planification du projet
- Prise en main du moteur de jeu et de Microsoft HoloLens
- Définition d'un scénario de simulation concret
- Modélisation 3D du scénario y compris des détails de l'intérieur d'un bâtiment
- Réalisation de la simulation 3D d'une intervention, y compris
 - la simulation d'un personnage "tireur" ayant un comportement intelligent,
 - la simulation de plusieurs personnages "civils", avec différents comportements,
 - l'interaction du policier avec le tireur, par exemple par voix.
- Tests et démonstrations

Candidat

Baehler Simon

Date: 2.6.2017

Signature:

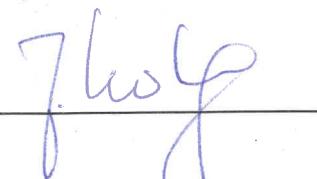


Responsable

Ehrensberger Juergen

Date: 2.6.2017

Signature:



Chef du département TIC

Sanchez Eduardo

Date: Yverdon-les-Bains, le 02.06.2017

Signature:



Confidentialité liée au Travail de Bachelor (TB)

Diplômant: Baehler Simon
Titre du travail de Bachelor: Réalité augmentée 3D HoloLens pour la simulation d'un scénario de crise
Domaine de recherche du TB: Informatique
Professeur responsable du TB: Ehrensberger Juergen

Tous les TB sont déposés à la Bibliothèque de la HEIG-VD qui en gère l'archivage et la consultation. Quel que soit le niveau de confidentialité du TB, le nom du diplômant, le nom du professeur responsable, le titre du TB, le domaine de recherche et le résumé publiable figurent dans tous les documents de présentation des TB ainsi que dans la base de données des TB (<http://tb.heig-vd.ch>). Le professeur responsable veille à ce que le titre du TB, le libellé du domaine de recherche et le résumé publiable soient rédigés conformément au niveau de confidentialité voulu.
Les TB peuvent être soumis à un logiciel anti-plagiat. Dans ce cas, leur contenu sera traité de manière confidentielle.



Le TB n'est pas confidentiel:

Outre les informations mentionnées ci-dessus, les documents de présentation du TB contiennent également les noms des entreprises partenaires, le résumé publiable et une affiche descriptive. Le TB peut être consulté ou emprunté librement à la Bibliothèque par le corps enseignant et les étudiants. Si une personne externe à la HEIG-VD souhaite consulter ou emprunter un TB, elle dépose une demande motivée auprès de la Bibliothèque, laquelle sollicite l'accord du professeur responsable et du doyen du département concerné.



Le TB est confidentiel: les conditions suivantes de diffusion des informations sont à appliquer:

- Oui Non Nous acceptons que **les noms des entreprises partenaires figurent dans les documents publiés ainsi que dans la base de données consultable sur <http://tb.heig-vd.ch>.**
- Oui Non Nous acceptons que, une fois validé par les entreprises partenaires, **l'affiche descriptive du TB figure dans la base de données consultable sur <http://tb.heig-vd.ch>.**
- Oui Non Nous acceptons que le TB soit consultable et empruntable par le corps enseignant, les étudiants ou une personne externe à la HEIG-VD sous condition de l'obtention de l'accord des entreprises partenaires, du professeur responsable du TB et du doyen du département concerné. Le TB porte la mention « **confidentiel, consultable sous condition** ».
- Oui Non Nous demandons qu'aucune consultation ou emprunt du TB ne soit permis hormis par le professeur responsable du TB qui s'engage à ne pas faire usage des informations mises à sa disposition. Le TB porte la mention « **confidentiel, non consultable** ».

Dans tous les cas, un accord de confidentialité doit être signé par l'étudiant, l'expert et toutes les personnes participant à l'évaluation du TB.

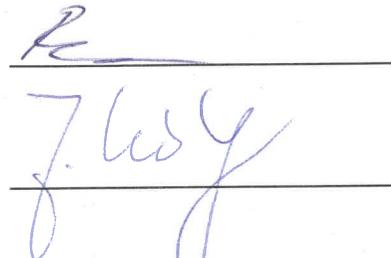
« Confidentialité »

Nous déclarons accepter les conditions de diffusion du Travail de Bachelor indiquées.

Diplômant :

Date: 2.6.2017

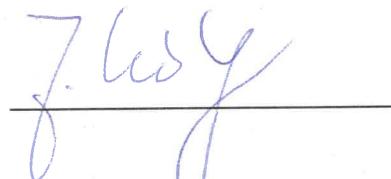
Baehler Simon



Professeur responsable :

Date: 2.6.2017

Ehrensberger Juergen



N.B.: Ce document fait partie intégrante du cahier des charges du TB.

La forme masculine est utilisée comme genre neutre et désigne à la fois les hommes et les femmes.

Table de matières

Remerciements	iii
Cahier des charges	v
Confidentialité liée au Travail de Bachelor	vii
1 Résumé	1
2 Introduction	3
3 Énoncé	5
3.1 Contexte	5
3.2 Projet antérieur	5
3.3 Objectifs	5
3.4 Tâches	5
3.5 Compétences	6
4 Analyse	7
4.1 Choix des technologies	7
4.1.1 Technologies et logiciels	7
4.1.2 Langages	7
4.1.3 PlugIns et Addons	7
4.1.4 Ressources mises à disposition	8
4.1.5 Unity vs Unreal	8
4.1.6 Hololens vs Oculus Rift	9
4.1.7 Rain AI	11
4.1.8 Polarith	14
4.1.9 IL2CPP vs .Net	14
4.1.10 HoloTool Kit	15
4.1.11 VRKinectAmokShooter	15
4.2 Analyse pré-développement	16
4.2.1 Gestuelle vs Vocale	16
4.2.2 Tir	18
4.2.3 Autres	19
5 Apprentissage	21
5.1 Rain AI	21
5.1.1 Introduction	21
5.1.2 Pathfinding	21
5.1.3 Detector	27

5.1.4	Interaction - Memory	32
5.1.5	Custom action	33
5.2	Collision curseur-model 3D	34
6	Scénarios-maquettes	37
6.1	Projet 1 : Rain AI	37
6.1.1	Objectifs	37
6.1.2	Description du projet	37
6.2	Projet 2 : Shooter	38
6.2.1	Objectifs	38
6.2.2	Description du projet	38
6.2.3	Civil	39
6.2.4	Forcené/Shooter	41
6.3	Projet 3	43
6.3.1	Objectifs	43
6.3.2	Description du projet	43
6.3.3	Civil	45
6.3.4	Forcené/Shooter	46
6.4	Projet 3.1 : HUD	49
6.5	Projet 3.2 : IA Avancées	49
6.5.1	Objectifs	49
6.5.2	Adam	49
6.5.3	Michael	50
6.5.4	Civil avancé	52
6.5.5	Shooter avancé	53
6.5.6	Safe Zone	53
7	Réalisation	55
7.1	Classes et diagrammes de classes - RAINAction	55
7.1.1	ReassuringCivilian	55
7.1.2	RandomWayPoint	55
7.1.3	CheckPanicRun	56
7.1.4	CheckPanic	56
7.1.5	PlayerAvoidance	56
7.2	Classe Keywords	57
7.2.1	KeywordHandler	57
7.3	Classes et diagramme de classes - IA	58
7.3.1	HumainAI	58
7.3.2	Civilian	60
7.3.3	ShopShooter	60
7.3.4	Adam	61
7.3.5	Michael	61
7.4	RainAI	63

7.4.1	Civilian	63
7.4.2	ShopShooter	64
7.4.3	Adam	66
7.4.4	Michael	68
7.5	Code	69
7.5.1	Création d'entités et de senseurs pour <i>Rain AI</i> via C#	69
7.5.2	Changement de scène	69
7.6	Problèmes rencontrés	70
7.6.1	Nouvelle technologie	70
7.6.2	Bonnes pratiques	71
7.6.3	Pathfinding sur AR et position de l'IA	73
7.6.4	Traversée des hologrammes par le joueur	74
7.6.5	Reconnaissance vocale	75
7.6.6	Apprentissage	76
7.6.7	RainAI et .Net	77
7.6.8	Holotool kit de Microsoft	78
7.6.9	RainAI sur Hololens et .Net 4.6	80
7.6.10	Non-unicité des ressources	81
7.6.11	Collision (Detection) Avoidance	81
8	Tests	83
8.1	Live compilation	83
8.1.1	Test sans forcené/shooter	83
8.1.2	Test avec forcené/shooter	84
8.1.3	Ramener le forcené/shooter dans la safe Zone	84
8.1.4	Ramener le forcené/shooter et les civils dans la safe Zone	85
8.2	Sur Hololens	86
8.2.1	Navigation	86
8.2.2	Commande gestuelle	86
8.2.3	Commande Vocale	86
8.2.4	Fin du jeu	86
9	Conclusion	87
9.1	Etat du projet	87
9.2	Utilité du Hololens	87
9.3	Paradoxe des jeux pédagogiques et de la simulation	87
9.4	Réponse aux questions de l'introduction	88
9.5	Les plus	89
9.5.1	Peu de grands problèmes	89
9.5.2	Une communauté active et à l'écoute	89
9.5.3	Unity - Game design	89
9.6	Erreurs commises - Les moins	89
9.6.1	Apprentissage	89

9.6.2 Premier jeu et jeu != logiciel classique	90
9.6.3 Incapacité à définir le projet minimum et de s'y tenir	91
9.7 Travaux futurs	92
9.8 Mots de la fin	93
10 Planification	95
11 Webographie	101
11.1 Microsoft	101
11.2 Youtube	101
11.3 Autre	102
12 Liste des symboles et abréviations utilisés	105
12.1 Symboles	105
12.2 Abréviations	105
13 Références	107
14 Annexe	115
14.1 Code	115
14.2 Manuel d'utilisation	121
14.2.1 Menu	121
14.2.2 Jeu	121
14.3 Manuel d'installation	125
14.3.1 Installation des logiciels	125
14.3.2 Installation de l'émulateur	125
14.3.3 Déploiement de l'application	126
15 Journal de travail	129

Résumé

Ce travail de Bachelor s'intéresse à la formation de policiers dans le cadre de scénarios de crise. Il possède deux objectifs bien distincts. Le premier objectif est la réalisation d'un outil/d'une application de formation pour la gestion de scénarios de crise pour la police. Le second objectif est un travail de recherche visant à explorer le *Hololens* en exploitant ses capacités techniques et à évaluer si oui ou non celui-ci est un outil viable à la formation de policer pour des scénario de crise, mais également dans un sens plus large, celui de l'évaluation de la viabilité de la réalité augmentée/réalité virtuelle dans le cadre de la formation de policer.

Cette application est présentée sous la forme d'une simulation en vue à la première personne dans un espace virtuel en trois dimensions. Le but de cette simulation est d'avoir une immersion et un rapprochement maximal de la réalité. La simulation prend place dans un supermarché, où plusieurs civils se déplacent en marchant et prennent peur à la vue du forcené/shooter. Le travail du policer n'est pas uniquement d'arrêter le forcené/shooter mais également de gérer l'évacuation des civils. L'interaction entre le policer et les intelligences artificielles peut s'effectuer de manière gestuelle et/ou vocale.

Au cours de ce projet l'outil utilisé a été *Unity*, un moteur de jeux, ainsi qu'un *PlugIn* du nom de *Rain AI* pour la gestion de intelligences artificielles. Pour la question du langage, seul le C# a été utilisé.

Introduction

L'apparition de nouvelles technologies comme les casques de réalité augmentée / réalité virtuelle a attiré les technophiles mais également les plus néophytes, y voyant déjà dans ces "gadgets" l'apparition d'un Saint Graal, ou le premier pas vers l'armure d'*Iron Man*, où l'humain est entièrement connecté à la machine.

Mais où en sommes-nous réellement aujourd'hui ? Actuellement, quelles sont les possibilités d'une telle technologie dans le cadre de la formation ? Nous aurons certes un parti pris axé sur la formation très pratique, celle de la formation de policiers dans le cadre de scénario de crise, mais rien ne nous empêche de nous poser la question de manière plus large, comme par exemple, l'utilisation de la réalité augmentée / réalité virtuelle dans le cadre de formations plus théoriques telles que l'anatomie ou l'astronomie.

La réponse à ces questions s'appuie essentiellement sur l'analyse des expériences personnelles qui ont eu lieu tout au long du projet, et qui devraient donc permettre d'évaluer l'utilité du *Hololens* ou, de manière plus générale, l'utilité de la réalité virtuelle / réalité augmentée. L'exploitation de ces expériences devait permettre de répondre à une série d'interrogations inhérentes au sujet telles que : L'immersion dans la simulation est-elle satisfaisante ? Est-il possible de réaliser une simulation proche de la réalité ?

Hypothétiquement parlant nous pouvons répondre à la question seulement grâce à son oxymore qui en fait partie intégrante : réalité augmentée / réalité virtuelle. D'un autre bord les casques de réalité augmentée / réalité virtuelle permettraient l'immersion quasi-totale dans un nouvel environnement, inconnu, et donc toucher des doigts des scénarios rares mais possibles.

Énoncé

3.1 Contexte

Ce TDB fait partie d'un projet plus large qui vise à démontrer l'utilité de visualisations 3D pour la simulation de scénarios de crise et la formation du personnel de la police pour l'intervention. Les personnes sont équipées de masques de réalité augmentée (HoloLens) qui visualisent l'environnement de manière réaliste puisque tenant compte de la position/orientation du porteur ainsi que des autres personnes éventuellement présentes.

L'idée principale de ce projet est de permettre au porteur du *Hololens* de pouvoir interagir de manière gestuelle et/ou vocale avec les différents éléments projetés dans les lunettes comme par exemple des civils et un forceⁿ/shooter au sein d'un scénario défini.

3.2 Projet antérieur

Un premier projet du même acabit a déjà été développé en utilisant d'autre technologies (*Oculus Rift* et *Kinect*). Ces technologies étant limitées sur certain points, comme par exemple les déplacements simulés par un levé de jambes et le fait que le casque devait être connecté de manière filaire, ont conduit à la fin au projet.

3.3 Objectifs

L'objectif de ce projet est la réalisation d'une visualisation interactive 3D d'une intervention par la police. Le scénario concret sera défini au début du projet. La visualisation utilise des masques HoloLens ainsi qu'une plate-forme 3D temps réel (moteur de jeu vidéo) comme *Unity 3D* ou *Unreal*.

Le projet est réalisé en collaboration avec le CCIR (Conférence des Chefs d'Instruction Romands).

3.4 Tâches

- Définition d'un scénario concret.

- Prise en main du moteur de jeu
- Modélisation 3D du scénario, y compris des détails de l'intérieur d'un bâtiment
- Réalisation de la simulation et de l'interaction avec l'environnement
- Tests et démonstrations

3.5 Compétences

- Expérience de développement de jeu vidéo
- Modélisation 3D
- Expérience avec les moteurs de jeu vidéo comme *Unity* 3D ou *Unreal*
- Développement logiciel

Analyse

4.1 Choix des technologies

Ce chapitre traite des logiciels et des technologies qui ont été utilisés pour la réalisation de ce travail de Bachelor. Il justifie également, par la suite, le choix de ces logiciels et de ces technologies.

4.1.1 Technologies et logiciels

- *Unity* 3D 5.6.1f1
- IL2CPP
- Visual Studio 2017 Entreprise
- Rider de JetBrains
- HoloLens
- Photoshop CC 2017
- Paint
- Visio 2016

4.1.2 Langages

- C#

4.1.3 PlugIns et Addons

- Rain AI
- HoloToolKit
- Polarith (découverte et test uniquement)

4.1.4 Ressources mises à disposition

- VRKinectAmokShooter

4.1.5 Unity vs Unreal

Unity

Unity est un moteur de jeux multi-plateforme, gérant autant la 2D que la 3D. Il permet de développer des jeux pour les plateformes suivantes :

- Windows (desktop), Windows Phone
- macOS et iOS
- Xbox 360 et One
- Wii et Wii U
- PlayStation 3, 4 et Vita
- Android
- WebGL

Asset et market

Unity est intéressant pour la mise à disposition d'un *Asset Store* permettant de se procurer des ressources, assurant un gain de temps important en nous acquittant des tâches qui ne sont pas de notre domaine de compétences, comme par exemple la création d'objets en 3D, de textures et d'animations pour nos objets 3D.

Développement

Pour ce qui est des langages de développement, *Unity* utilise le C# et le JavaScript indépendamment de la plateforme de déploiement finale, cela est du au fait que le langage est interprété par *Unity*. Un point intéressant avec *Unity* est son outil de *Live Compilation* qui nous évite de déployer notre travail à chaque modification. Nous pouvons donc jouir d'un gain de temps non-négligeable.

Unreal

Au même titre que *Unity*, *Unreal* est également un moteur de jeu développé par *Epic Games, Inc.*, et tout comme *Unity*, il est multi-plateforme et est équivalent d'un point de vue de portabilité. Ci-après les plateformes compatibles :

- Windows, OS X, Linux

- Playstation 4
- Xbox One
- SteamOS
- IOS, Android
- HTML5

Asset et market

Tout comme *Unity*, il possède également un *asset store* permettant de se procurer des ressources.

Développement

Unreal utilise le C++. Contrairement à *Unity* où le code est interprété, ici le code est compilé. Un point très intéressant avec *Unreal* est son outil nommée *Blueprint* (non ce n'est pas l'écran bleu de *Windows*), il s'agit d'un outil de développement graphique qui nous permet de réaliser un jeu, ou du moins un prototype de jeu sans avoir nécessairement de bonnes connaissances en programmation.

Choix final : Unity

Le choix de *Unity*, à la place de *Unreal Engine*, a été motivé par le fait qu'un projet similaire a déjà été réalisé par deux personnes au sein de l'école. Ainsi, une ligne directrice peut être conservée et une partie du travail déjà effectué peut être réutilisé. De plus, cet ancien projet peut aussi servir de formation et de source d'informations pour ce nouveau projet. La dernière chose, et non des moindres, est que *Unity* et *Microsoft* sont partenaires¹ pour le projet *Hololens*, ce qui apporte une meilleure compatibilité. Une autre avantage résultant de ce partenariat entre *Unity* et *Microsoft* est que les formations mises à disposition par *Microsoft* pour le développement sur *Hololens* sont sous *Unity*. De plus, *Microsoft*, a également mis à disposition sur leur repository *GitHub* un *PlugIn Unity* qui sert de *Toolkit*. Un grand nombre d'exemples et d'outils y sont disponibles, le travail est donc grandement facilité.

4.1.6 Hololens vs Oculus Rift

Oculus Rift

L'*Oculus Rift* est un casque de réalité virtuelle compatible avec les systèmes d'exploitation *Windows*, *Linux*, *OS X* et *Android*. A l'opposé de l'écosystème du *Hololens* se trouve l'*Oculus Rift*. Ce dernier possède un écran opaque, un système audio intégré et des entrées HDMI et USB. Pour les commandes, deux gamepads, aussi appelés *Oculus Touch*, sont utilisés. L'*Oculus*

¹<http://www.windowscentral.com/unity-3d-details-its-partnership-hololens>

Rift est un casque filaire, c'est-à-dire qu'il doit être connecté à un ordinateur via un câble, et contrairement au *Hololens*, ce n'est pas un ordinateur à part entière.

Figure 4.1 – Oculus DK2



Source : <https://www3.oculus.com/en-us/dk2/>

Hololens

Le *Hololens* est un casque de réalité augmentée, à l'inverse de *Occulus Rift* qui est un casque de réalité virtuelle, permettant de générer et d'interagir avec des hologrammes dans l'environnement qui nous entoure. Il ne possède pas d'écran réel, les hologrammes sont projetés en transparence sur la vitre de la lunette. Il faut savoir que le *Hololens* est un ordinateur complet et pas un périphérique relié à un ordinateur. C'est une technologie plus récente et les kits de développement ne sont disponibles que depuis octobre 2015. Il possède une version adaptée de *Windows*, de capteurs de mouvements pour les commandes gestuelles et un microphone (pour les commandes vocales) ainsi que trois processeurs, respectivement un CPU, un GPU et un HPU (*Holographic Processing Unit*)

Figure 4.2 – Hololens



Source : http://www.etr.fr/devices_images/980-hololens_face.png

Choix final : Hololens

Le choix de partir sur le *Hololens* a été effectué avant la mise en route du travail de Bachelor. Les raisons ayant motivé ce choix sont les suivantes. Premièrement, par le fait que nous nous étions heurtés aux limitations techniques de *Occulus Rift*, principalement le fait que les déplacements étaient non-naturels (simulation du déplacement en levant les jambes mais en restant sur place). Un autre problème est qu'il faut posséder une réalisation 3D du bâtiment où se déroule le scénario, ce qui d'un point de vue mémoire et puissance graphique peut s'avérer gourmand. A noter que ce point sera finalement toujours présent dans la version du *Hololens* à cause du problème de *Pathfinding* (plus de détails dans le chapitre dédié aux problèmes rencontrés). Et finalement nous ne sommes pas limités par la capacité physique de notre machine car le *Hololens* possède un ordinateur.

4.1.7 Rain AI

C'est un *PlugIn Unity* pour gérer les intelligences artificielles (que nous abrégerons IA dans ce document). Il a l'avantage d'être gratuit et sans licence. Il offre également un grande variété de services comme le *Pathfinding*, la détection de bruit et de présence et possède un système équivalent aux variables de programmation, appelé *Memory*. Nous utiliserons ce *PlugIn* pour les hologrammes de personnes de notre jeu (civils et forcené/shooter), et ce afin de leur donner la possibilité de réagir à ce qui se passe dans les jeux (actions de l'utilisateur ou actions des autres hologrammes).

Le choix de prendre *Rain AI* plutôt qu'un autre *PlugIn* d'intelligence artificielle a été motivé par le fait que celui-ci est également utilisé dans le projet sous *Occulus Rift*. Ainsi, nous concevons une continuité sur la technologie utilisée. De plus, il offre des services très complets et est prisé par des grandes compagnies de l'industrie du jeu vidéo. Il dispose aussi d'une large communauté, facilitant la recherche de tutoriels de formation. Une présentation plus

approfondie de *Rain AI* se trouve plus bas. Une section lui est dédiée dans le chapitre consacré à l'apprentissage.

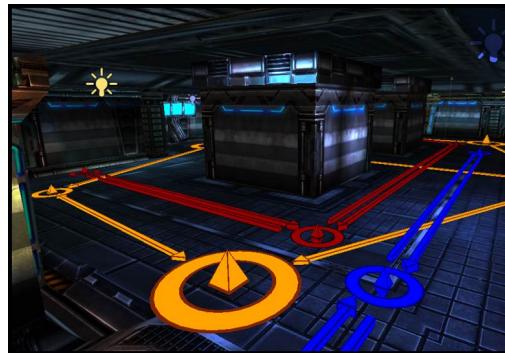
Mouvement

La première image ci-dessous représente les *Navigation Meshs* (zones bleues). Il s'agit de la surface où notre IA est capable de passer. La seconde image représente des routes de passage que notre IA pourra emprunter. Dans le cas où deux points adjacents sont séparés par un obstacle, *Rain AI* est capable de le contourner pour relier les deux points adjacents. Une documentation plus détaillée du *Pathfinding* est donnée dans le chapitre d'apprentissage.

Figure 4.3 – Navmesh



Figure 4.4 – Route

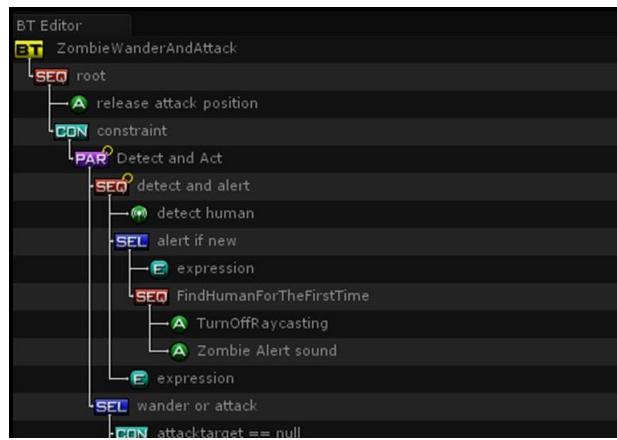


Source : <http://legacy.rivaltheory.com/rain/features/>

Arbre de décision

Une autre force de *Rain AI* est qu'il offre la possibilité d'implémenter des IA sans poser une seule ligne de code, et cela via ce qui s'appelle un *Behavior Tree* (voir illustration ci-dessous). Une documentation plus détaillée du *Behavior Tree* est donnée dans le chapitre d'apprentissage ainsi que dans le chapitre de réalisation où sont montrés les différents *Behavior Tree* du projet.

Figure 4.5 – arbre de décision



Source : <http://legacy.rivaltheory.com/rain/features/>

Perception

Un point extrêmement intéressant avec *Rain AI* est qu'il offre la possibilité à une IA d'avoir un senseur de détection, visuel ou auditif. Cette possibilité sera fort utile pour l'intelligence artificielle du forcené/shooter et des civils. Par exemple, si le civil aperçoit le tireur, cela peut déclencher une action, telle que la fuite vers la sortie ou bien la paralysie tant que le tireur est à moins de X mètres. Une explication plus détaillée est donnée dans le chapitre d'apprentissage.

Figure 4.6 – Perception



Source : <http://legacy.rivaltheory.com/rain/features/>

4.1.8 Polarith

Polarith est un autre *PlugIn* intéressant pour IA qui a été ajouté à la fin du projet mais n'a malheureusement pu être que très peu exploré. La raison à cela est due au fait qu'une version gratuite n'est sortie que très récemment, mi-juin 2017. *Polarith* est intéressant car il permet d'offrir à nos IA la capacité d'éviter d'autres IA. Il est très intéressant pour rendre un mouvement de foule plus réaliste.

Un petit projet de test a été réalisé, mixant *Rain AI* et *Polarith* :

- <https://github.com/simon-baehler/collisionAvoidance>

4.1.9 IL2CPP vs .Net

IL2CPP

IL2CPP est un *scripting backend* développé par *Unity*. Lors de la compilation, *Unity* convertit nos *Common Intermediate Language*² et les assemble en C++ avant de créer la librairie.

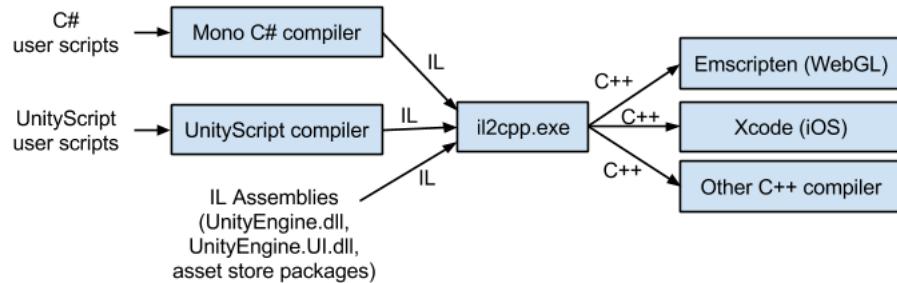
1. Le code de l'API de *Unity* est compilé en .Net DLL
2. Unused Bytecode Stripper trouve et supprime toutes les classes inutiles

²Dans l'environnement de programmation Microsoft, le Common Intermediate Language (CIL) est le langage de programmation de plus bas niveau qui peut être lu par un humain. Le code de plus haut niveau dans l'environnement .NET est compilé en code CIL qui est assemblé dans un code dit bytecode. CIL est un code assembleur orienté objet et pile. Il est exécuté par une machine virtuelle.

Le CIL était initialement connu sous le nom de Microsoft Intermediate Language ou MSIL durant les bêta du langage .NET. Après la standardisation du C et de la CLI, le bytecode fut officiellement référencé sous le nom de CIL. Les utilisateurs précoce de la technologie continuent néanmoins à se référer au terme MSIL. Source : https://fr.wikipedia.org/wiki/Common_Intermediate_Language

3. Conversion en code C++
4. Compilation du C++
5. Link du code à un exécutable ou une DLL selon la plate-forme

Figure 4.7 – Fonctionnement de IL2CPP



Source : <https://tinyurl.com/ybaba7jr>

Un grand intérêt d'IL2CPP est qu'il apporte une grande portabilité et une performance de haut-rang à *Unity*.

Le choix d'utiliser IL2CPP à la place de .Net découle directement du choix précédent. *Rain AI* n'étant pas compatible avec .Net pour des raisons inconnues, il faut donc trouver un moyen auxiliaire. IL2CPP est ce moyen. IL2CPP est un *scripting backend* développé par *Unity*. Ce changement fait d'une pierre trois coups car, en plus de résoudre notre problème de compatibilité entre *Rain AI* et IL2CPP, nous avons apparemment un gain en performance et également un gain en portabilité.

Plus d'information sur *IL2CPP* sont disponibles sur le site de *Unity*³

4.1.10 HoloTool Kit

Holoolkit est un *PlugIn* offert par *Micosoft* comportant une collection de scripts et d'exemples pour une réalisation facilitée de nos projets. Il est disponible sur le répertoire *GitHub* de *Microsoft*⁴

4.1.11 VRKinectAmokShooter

Il s'agit d'un projet antérieur réalisé par deux anciens étudiants de la HEIG. L'idée du projet est la même; nous avons toujours un shooter/forcené avec des civils qui se déplacent dans le

³<https://docs.unity3d.com/Manual/IL2CPP.html>

⁴<https://github.com/Microsoft/HoloToolkit-Unity/>

bâtimen. Contrairement à ce projet, VRKinectAmokShooter a été réalisé sur l'*Occulus Rift*. Le projet n'a pas eu de suites pour les raisons citées précédemment dans les défaut de l'*Occulus Rift*, à savoir : déplacement pas naturel et casque filaire.

4.2 Analyse pré-développement

Cette section contient les différentes analyses et réflexions qui ont été faites avant leurs implémentations au sein du projet principal.

4.2.1 Gestuelle vs Vocale

Le *Hololens* offre deux méthodes pour interagir avec notre environnement : la première est la gestuelle dont une liste est donnée plus bas avec un code d'exemple, lui donné en annexe, la seconde fonctionne via la reconnaissance vocale.

Gestuelle

Dans cette partie nous analyserons plus en détail la viabilité des commandes gestuelles du *Hololens* ainsi que les possibilités qu'elles nous offrent. L'utilité de ce type de commande se trouve principalement hors du jeu, comme par exemple au début, dans le menu, avant de lancer le jeu.

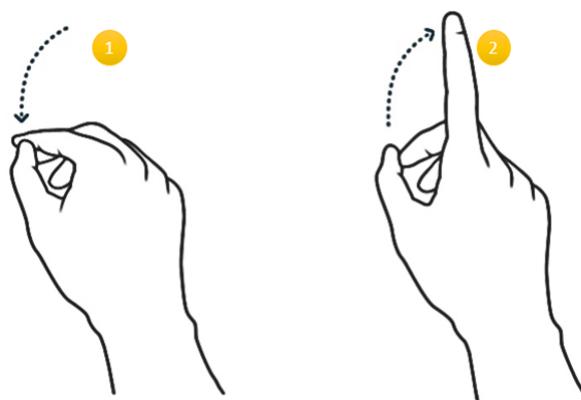
- Select : Commande de base que l'on peut associer à un clic de souris. Il suffit simplement de positionner sa main dans le champ de vision du *Hololens*, revers de main face à nous, et d'effectuer un mouvement de pincement avec notre pouce et notre index, puis relâcher. Aussi appelé *Air Tap ou Tap Movement*
- Bloom : Utilisé pour ouvrir le menu, ce mouvement s'apparente à lancer une balle de jonglage.
- Tap : Idem que Select, mais attend que l'utilisateur relâche sa main.
- Hold : Mouvement de sélection sans relâchement
- Manipulation : Mouvement de sélection sans relâchement, suivi d'un mouvement absolu de main dans n'importe quelle dimension.
- Navigation : Mouvement de sélection sans relâchement, suivi d'un mouvement relatif de main dans n'importe quelle dimension.

Sur les figures 4.8 et 4.9 nous avons les deux gestes de base possibles sur le *Hololens*. Dans le listing donné en annexe *GazeGestureManager.cs* et *SphereCommands.cs*, nous avons un code d'exemple implémentant le *Air Tap ou Tap Movement*. Il permet de déclencher une action

d'un objet quand celui-ci est ciblé et que nous effectuons le *Air Tap*. Dans l'exemple donné, nous rendons tout simplement l'objet sensible à la gravité. Le code donné en exemple ici est pris du tutoriel de base sur le *Hololens* de Microsoft⁵

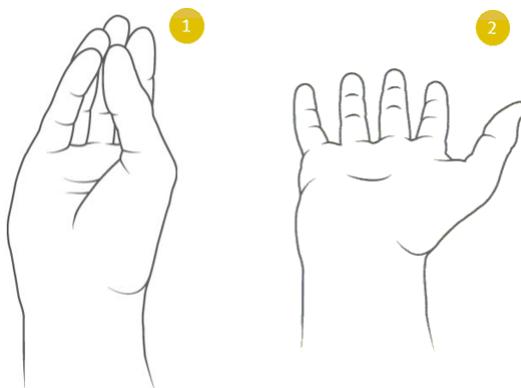
Nous pouvons rapidement constater une problématique : la possibilité de mouvement est limitée et, de plus, ces mouvements ne sont pas vraiment ceux utilisés par un policier sur le terrain, à moins que nous puissions appréhender une personne en lui pinçant l'oreille. En revanche, la gestuelle n'est pas perdue pour autant, puisqu'elle trouve son utilité dans les taches plus statiques comme par exemple le clic de bouton sur le menu (exemple donné plus haut)

Figure 4.8 – Air tap



Source : <https://abhijitjana.files.wordpress.com/2016/05/image51.png>

Figure 4.9 – Bloom



Source : https://kbdevstorage1.blob.core.windows.net/asset-blobs/12462_en_3

⁵https://developer.microsoft.com/en-us/windows/mixed-reality/holograms_101e

Vocale

Après avoir exploré les possibilités gestuelles du *Hololens* et constaté leurs faibles viabilités pour un projet telle que celui-ci, nous allons explorer le deuxième type de commande possible qui comprend les commandes vocales. Nous pouvons utiliser ces types de commande pour des tâches où plusieurs éléments (*gameObjects*) entrent en jeu - comme par exemple demander à une foule d'évacuer un bâtiment et de se diriger vers la sortie, soit des tâches où un focus d'un objet n'est pas nécessaire.

Voici une liste des bonnes pratiques d'utilisation des commandes vocales :

- Commandes concises qui peuvent être facilement mémorisées par l'utilisateur
- Commandes avec plusieurs syllabes. Favoriser des commandes comme "ouvrir la boite" ou "jouer la vidéo"
- Utiliser des mots simples et courants

Voici une liste des mauvaises pratiques d'utilisation des commandes vocales :

- Commandes trop courtes avec une seule syllabe
- Utiliser les commandes qui existent déjà dans le système. Éviter par exemple les commandes comme "select menu" ou "close menu". Les commandes qu'il ne faut pas utiliser sont les suivantes :
 - Go Back
 - Scroll Tool
 - Zoom Tool
 - Drag Tool
 - Adjust
 - Remove
- Utiliser les commandes avec des mots similaires

Les commandes vocales géreront la majeure partie des actions pouvant être effectuées par l'utilisateur. Il s'agira principalement de commandes pour interagir avec les IA; les commandes gestuelles ne s'y prêtant pas.

4.2.2 Tir

Le tir étant une solution de dernier recours, et donc non indispensable au fonctionnement du projet, ce sujet n'a été que survolé en suivant différents tutoriels (peu concluant) et en regardant le projet précédent, plus particulièrement la scène nommée *shotti_v2*

4.2.3 Autres

D'autres éléments ont aussi suscité un questionnement avant le développement du projet principal comme :

- Pathfinding des Hologrammes
- Collision entre Hologrammes et joueur

Ces sujets relevant de la technique et étant complexes, ils seront abordés plus en détail dans la section des problèmes rencontrés

Apprentissage

Ce chapitre relève de la phase d'apprentissage et de découverte de *Unity*, des *PlugIn* qui lui sont rattachés et du *Hololens*. Il traite principalement des points qui ne sont pas mentionnés dans les tutoriels de formation offerts par *Micosoft*¹. A noter que ces tutoriels sont relativement succincts, explorant certes le nécessaire, mais possédant plusieurs défauts. Ils sont très peu génériques et passent par-dessus certains points, n'expliquent pas certains morceaux de code qui pourraient être intéressants et utilisent des objets préfabriqués qui possèdent des *Components* sans pour autant le préciser (ex. Collision curseur-model 3D). Le dernier point est relativement problématique quand nous ne connaissons que très peu *Unity*.

5.1 Rain AI

5.1.1 Introduction

Bien que les sources d'informations et de formations sur *Rain AI* soient nombreuses (entre tutoriels sur des forums ou en vidéo sur *Youtube*, il semblerait qu'une partie soit dépréciée et que beaucoup de tutoriels soient obsolètes - principalement dû au fait que dans la dernière version il n'est plus possible de gérer les animations). Ce chapitre contient les informations nécessaires pour effectuer à nouveau le travail et parcourt les différents outils qu'offre *Rain AI*

5.1.2 Pathfinding

Ce chapitre traite du *Pathfinding* avec *Rain AI*. Il s'agit de faire naviguer/se déplacer nos IA de manière à ce qu'elles ne passent pas au travers des objets/murs et autres éléments en 3D ou ne se bloquent pas dans un coin en voulant rejoindre un point.

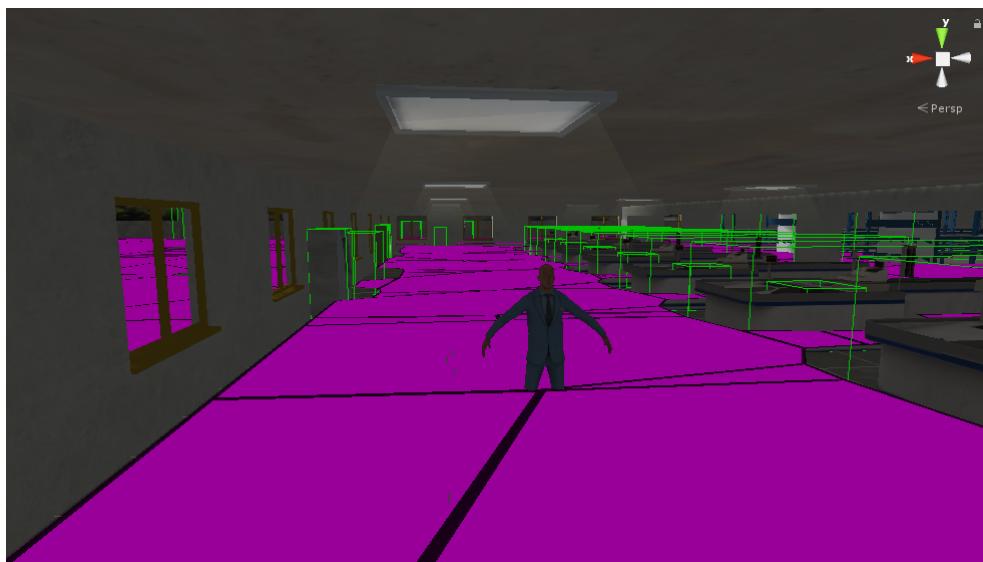
Après avoir installé le *PlugIn* depuis *l'Asset Store*, il suffit de suivre cette marche à suivre pour obtenir un premier *Navigation Mesh* (Zone où nos IA pourront se déplacer)

1. *RAIN -> Create New -> Navigation Mesh*
2. Sélectionner la *Navigation Mesh* et l'agrandir de manière à ce qu'elle couvre toute notre zone

¹ <https://developer.microsoft.com/en-us/windows/holographic/academy>

3. Avant de générer la *Navigation Mesh*, il faut lui indiquer quels sont les *Layers* que nous voulons prendre en compte et il est préférable d'exclure les éléments tels que les *UI* et les *IA*. Pour le reste, les valeurs par défaut sont satisfaisantes.
4. Pour terminer, cliquer sur *Generate Navigation Mesh*

Figure 5.1 – Navigation Mesh générée

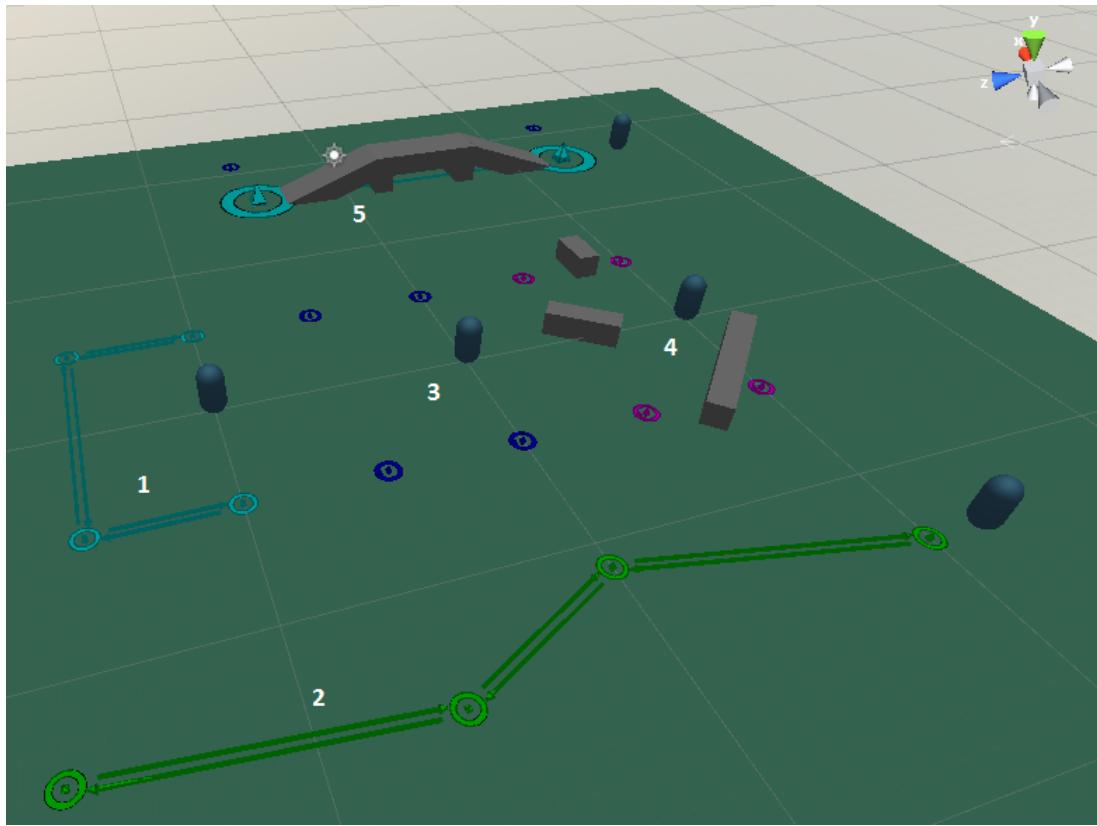


Si tout s'est bien passé nous devrions avoir un objet plat de couleur recouvrant la zone. Celle-ci représente les endroits où les IA peuvent se déplacer. Un exemple se trouve en 5.1 - Navigation Mesh générée. A partir de là, la zone de navigation est délimitée, et il reste les points suivants à effectuer : définir un chemin et l'associer avec une IA.

Pour définir un *Path* nous avons trois options à choix (voir figure 5.2 - Différents type de waypoint) :

- Waypoint Route (1 et 2)
- Waypoint Network (5)
- Navigation Target (3 et 4)

Figure 5.2 – Différents type de waypoint



Waypoint Route

La *Waypoint Route* pourrait s'amalgamer à une patrouille, notre IA va se déplacer d'un point précis à un autre. Pour ajouter une *Waypoint Route* :

1. Aller dans *Rain* -> *Create New* -> *Waypoint route*. Un nouvel élément vient s'ajouter au *Hierarchy Panel*
2. Le sélectionner
3. Cliquer sur *Add* dans *l'inspector* pour ajouter un *Waypoint*

Ensuite, il faut lier la *Waypoint Route*, à une IA

1. Aller dans *RAIN* -> *Behavior Tree Editor* -> *Create New Behavior Tree*
2. Le sélectionner
3. Clic droit sur *Root*, *Create* -> *descisions* -> *Waypoint Partol*

4. Dans le champ *Waypoint Route*, mettre le nom de la *Waypoint Route* souhaitée. IMPORTANT : le nom doit être entre guillemets.
5. Remplir le champ *Move Taget Variable* avec un nom quelconque, mais qui sera réutilisé par la suite.
6. Cliquer droit sur le *Waypoint Partol* du *Behavior Tree*, *Create -> Actions -> Move*. Dans le champ *Move Target* mettre le nom mis dans le champ *Move Taget Variable*.
7. Choisir une vitesse de déplacement
8. (Optionel) Pour permettre à notre IA de regarder vers une *Target* il faut spécifier le champ *Face Target*
9. Pour terminer, sélectionner l'objet auquel nous voulons donner l'intelligence artificielle, *RAIN -> Create New - AI* et lui assigner le *Behavior tree* créé.

Figure 5.3 – Configuration du Pathfinding



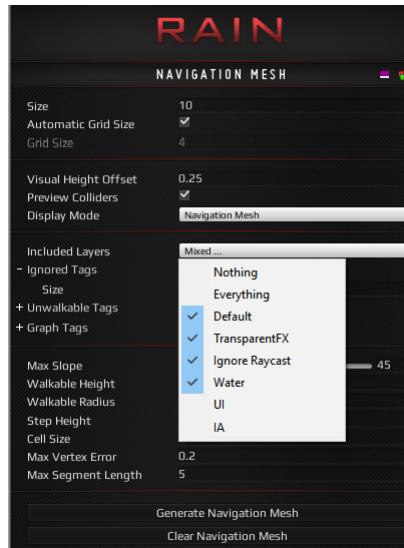
Sur l'image 5.2 - Différents type de waypoint, nous avons deux *Waypoint Routes* (1 et 2), la première est configurée en *Loop*, c'est à dire que l'IA va se déplacer du point 1 puis au 2 puis au 3 puis au 4 et à nouveau au 1. Dans notre second cas, le chemin est configuré en *One Way*, c'est à dire qu'il va aller au point 1, puis 2 puis 3 puis 4 et s'arrêter. Il reste un troisième cas, le *Ping Pong*, qui fait 1 puis 2 puis 3 puis 4 puis retourne au 3 et ainsi de suite.

Waypoint Network

Le *Waypoint Network* permet de créer un réseau de points reliés entre eux (ici nous pouvons relier un point à plus de deux autres points contrairement à la *Waypoint Route*) qui va nous permettre d'atteindre une *Navigation Target*. Sur l'image 5.2 - Différents type de waypoint, le *Navigation Network* est utilisé en 5 pour faire la traversée sur le pont. Il est défini avec les deux grands points turquoises, et permet de faire des allers-retours entre les deux points bleus en haut de l'image en passant par le pont. Pour un exemple plus concret, nous allons prendre le cas illustré par l'image en 5.5 - Fonctionnement du Navigation Network.

Notre bloc rouge va choisir une destination parmi les trois possibles (ronds rouge, bleu ou jaune). Admettons que la première destination soit la rouge, le bloc va aller au point bleu

Figure 5.4 – configuration de la Navigation Mesh



numéro 1, puis au 2, puis au 3, puis il va quitter le *Waypoint Network* et se diriger vers la *Navigation Target* rouge (numéro 4). Une fois arrivé au point 4, il repartira vers une autre destination. Admettons que cette fois il choisisse le point numéro 7, il prendra le chemin suivant 3 -> 5 -> 6 -> 7. S'il était parti depuis son point d'origine, il aurait fait 1 -> 2 -> 5 -> 6 -> 7.

Pour la création d'un *Waypoint Network*, pour procédons de la même manière que pour la création de la *Waypoint Route*, à la différence près que nous choisirons *Waypoint Network* au lieu de *Waypoint Route*.

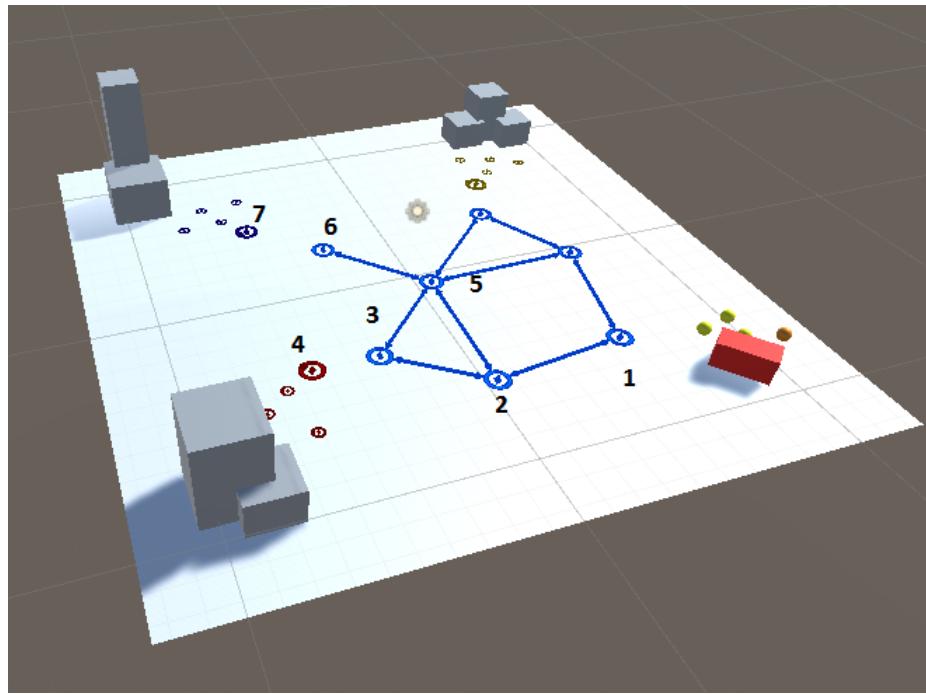
1. Aller dans *Rain* -> *Create New* -> *Waypoint Network*. Un nouvel élément vient s'ajouter au *Hierarchy Panel*
2. Le sélectionner
3. Cliquer sur *Add* dans *l'Inspector* pour ajouter un *Waypoint*
4. Aller dans *Rain* -> *Create New* -> *Navigation Target* et ajouter le point de destination

Pour la partie du *Behavior Tree*, la mise en place est une fois de plus pratiquement la même que pour la *Waypoint Route*.

Waypoint Network

1. Aller dans *RAIN* -> *Behavior Tree Editor* -> *Create New Behavior Tree*
2. Le sélectionner

Figure 5.5 – Fonctionnement du Navigation Network



3. un clic droit sur *Root*, *Create -> Descisions -> Waypoint Path*
4. Dans le champ *Waypoint Network*, mettre le nom du *Waypoint Network* souhaité. IM-PORTANT : le nom doit être entre guillemets.
5. Dans le champ *Path target*, mettre le nom de la *Navigation Target* (destination) souhaitée. Là encore il faut le mettre entre guillemets.
6. Remplir le champ *Move Target Variable* avec un nom quelconque ; il sera référencé par la suite.
7. Cliquer droit sur le *Waypoint Patrol* du *Behavior Tree*, *Create -> Actions -> Move*. Dans le champ *Move Target* mettre le nom mis dans le champ *Move Target Variable*.
8. Choisir une vitesse de déplacement
9. (Optionnel) Pour permettre à notre IA de regarder vers un *Target*, il faut spécifier le champ *Face Target*
10. Pour terminer, sélectionner l'objet auquel nous voulons donner l'intelligence artificielle, *RAIN -> Create New -> AI* et lui assigner le *Behavior Tree* créé.

Naviagtion Target

La troisième possibilité est la plus libre, il n'y a cette fois aucune connexion entre les différents points que nous allons placer. Encore une fois nous allons procéder pratiquement de la même manière que pour les deux solutions précédentes.

1. Aller dans *Rain* -> *Create New* -> *Navigation Target*. Un nouvel élément vient s'ajouter au *Hierarchy Panel*
2. Répéter l'opération pour chaque point désiré

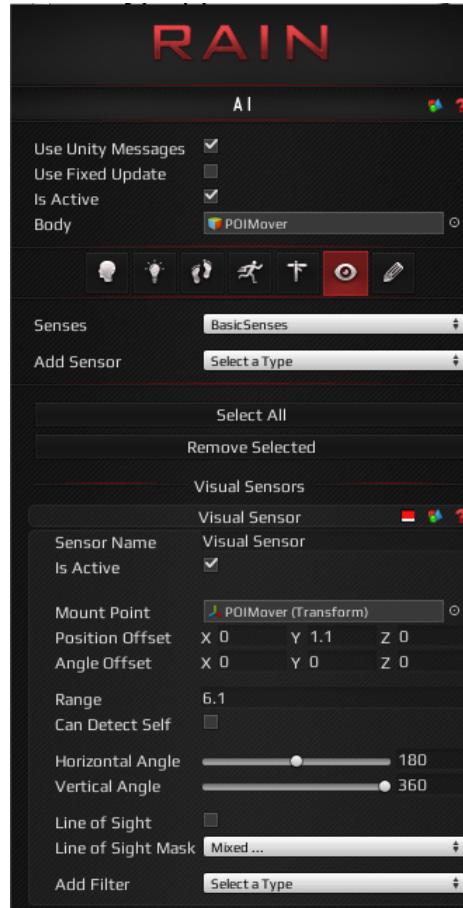
Pour la partie du *Behavior Tree*, la mise en place est une fois de plus pratiquement la même que pour la *Waypoint Route*, à la différence près que cette fois nous mettons directement un élément *move* associé à notre *Navigation Target*.

5.1.3 Detector

Un autre élément crucial pour la réalisation d'une IA est la capacité de détecter les autres éléments environnants. Pour implémenter cela, deux sens sont mis à contribution, la vue et l'ouïe.

Pour ajouter un sens à une IA, il suffit de se rendre dans l'*Inspector* de cette dernier et de cliquer sur l'œil comme illustré en 5.6 - Menu Perception de Rain + Visual Sensors, cliquer sur le menu déroulant en regard de *Add Sensor* et sélectionner le type de sens désiré. Dans l'image en 5.6 - Menu Perception de Rain + Visual Sensors, nous avons un exemple d'un senseur visuel projeté à 180° autour de l'IA - représenté en 5.10 par la demi-sphère rouge.

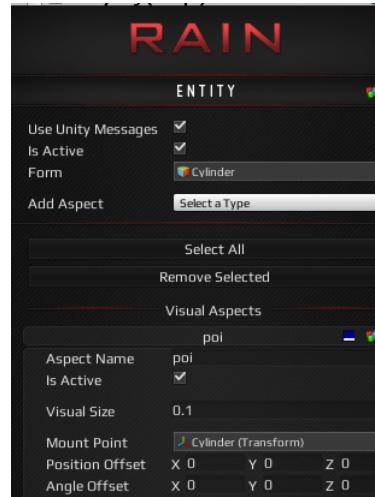
Figure 5.6 – Menu Perception de Rain + Visual Sensors



Pour être détecté par notre IA, un objet doit posséder un composant nommé *Entity*. Pour lui en associer un, il suffit de sélectionner l'objet, et de faire *RAIN -> Create New -> Entity*. Nous voyons qu'un objet fils s'est ajouté à notre objet précédemment sélectionné. Cet enfant nommé *Entity* possède un composant illustré en 5.7. Ce composant peut posséder des *Aspects* visuels ou auditifs. Dans l'illustration en 5.7, nous avons un aspect visuel nommé *poi*.

Il ne reste plus qu'à ajouter un détecteur dans notre *Behavior Tree*. Bien que cela ne soit pas fondamentalement nécessaire, il est préférable de mettre la détection en parallèle du reste des actions de notre IA (comme illustré en 5.8 Behavior Tree avec détecteur).

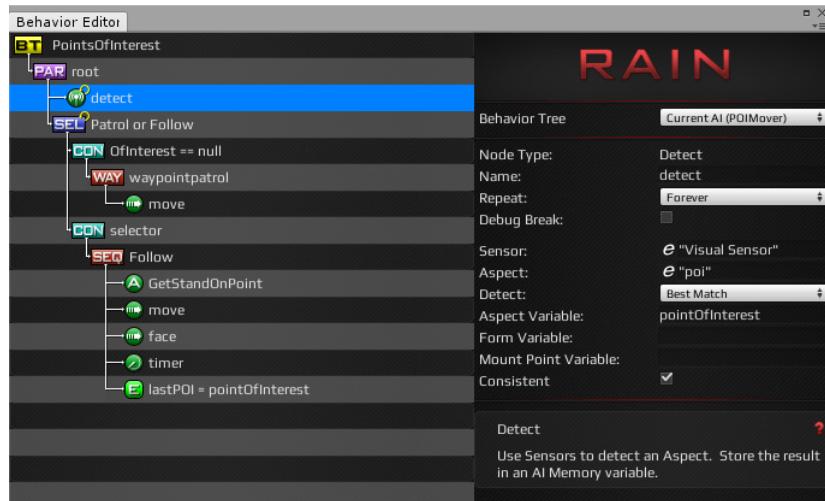
Figure 5.7 – Entity



1. Ouvrir le *Behavior Tree de notre IA*
2. Lui ajouter un bloc *parallel* Clic droit *Create -> Decisions -> Parallel*
3. Ajouter les éléments que l'on a envie de faire fonctionner en parallèle
4. Ajouter un détecteur *Create -> Actions -> Detect*
5. Dans le champ *Sensor*, ajouter le nom du senseur
6. Dans le champ *Aspect*, ajouter le nom de l'aspect
7. Dans le champ *Aspect Variable* ajouter un nom quelconque, ce nom sera référencé par la suite en tant que *Move Target*

L'IA de la figure en 5.10 - Sense visuel (capsule rouge) utilise le *Behavior Tree* avec détecteur de AI figure 5.8 Behavior Tree avec détecteur. Elle patrouille sur le chemin rouge, si elle voit un objet ayant une *Entity* du nom de *poi* elle va se diriger vers lui (dans notre cas il s'agit des cylindres intégrés au mur), reste pendant 2 secondes devant le cylindre, puis repart sur le tracé rouge, jusqu'à ce qu'elle tombe sur un *poi* différent du dernier.

Figure 5.8 – Behavior Tree avec détecteur



La figure 5.9 - Schéma Détecteur - Senseur - Entité, représente de manière schématique les liens entre les différents éléments pour la création d'un détecteur. Le bloc "Détecteur" représente l'élément *Detect* du *Behavior Tree* de notre IA. Le bloc "Senseur" représente l'élément *Sense* de notre AI. Le dernier élément est notre "Entité"; il est ajouté au *GameObject* qui sera détecté par le duo senseur-détecteur. Il est important de noter que le bloc "Entité" n'appartient pas au même *GameObject*.

Figure 5.9 – Schéma Détecteur - Senseur - Entité

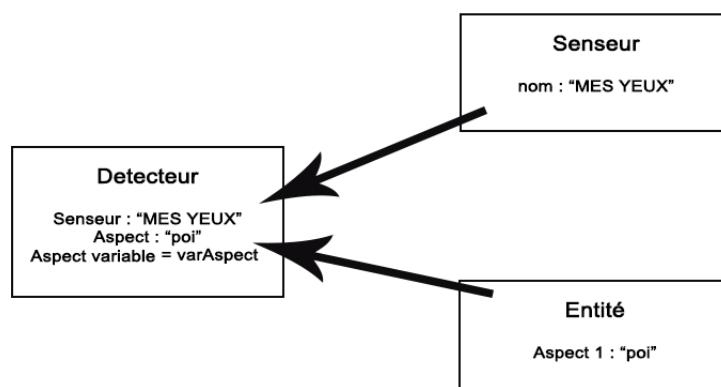
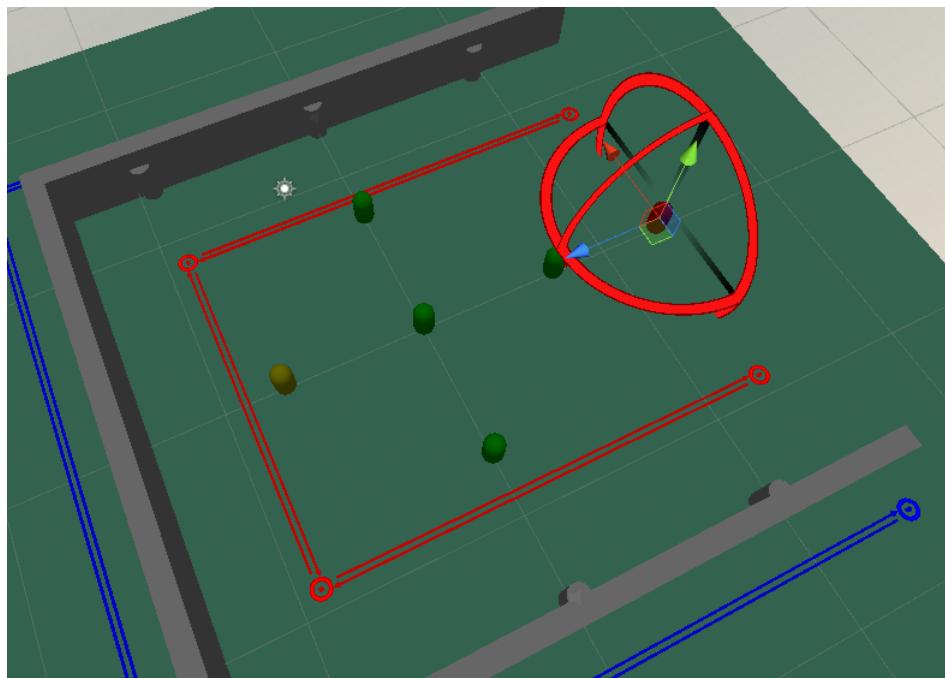


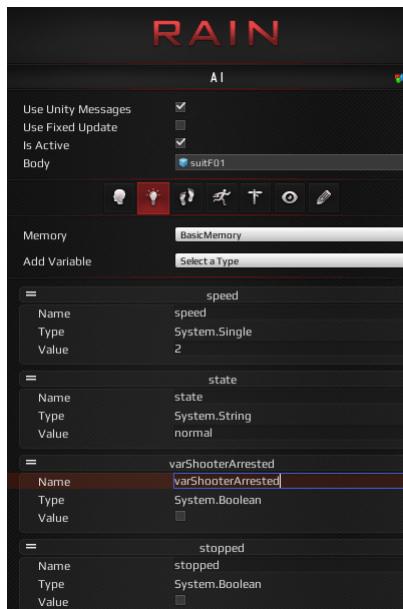
Figure 5.10 – Sense visuel



5.1.4 Interaction - Memory

Ce chapitre traite de l'interaction entre nos programmes C# et *Rain AI*, comment accéder à ce qui s'appelle la *Memory*. Pour tirer un parallèle avec la programmation, la *Memory* est l'équivalent des variables en programmation. Le code donné en annexe nommé *Memory.cs* est un bout de code de l'IA du civil du *Projet 2 : Shooter*. Dans cette méthode nous regardons si le *shooter* est arrêté par le policier. Si c'est le cas nous modifions la variable *varShooterArrested* créé dans la *Memory* de notre IA (voir figure en 5.11 - Memory)

Figure 5.11 – Memory



A noter qu'il est possible de créer des variables *Memory* sans passer par l'interface. Il est possible de les créer via une classe C# rattachée à notre IA, par exemple la ligne ci-dessous créera la variable *speed* si celle-ci n'est pas déjà créée.

```
1 tRig.AI.WorkingMemory.SetItem<float>("speed", 2)
```

Les illustrations de ce chapitre sont tirées du *starter Kit* de *Rain AI*, il est possible de le télécharger sur le site de *Rivaltheory*²

²<http://legacy.rivaltheory.com/?ddownload=34506>

5.1.5 Custom action

Il arrive que pour certaines actions, le *Behavior Tree* soit insuffisant ou mal adapté. Pour régler ce manquement, nous pouvons passer par des *Custom Actions* qui sont du code C#. Une possible implémentation dans notre cas serait un script qui se lance lorsqu'une IA en croise une autre. Ce script accède à l'état de l'IA croisée, et si cette dernière est en panique ou en fuite, elle propagera automatiquement son état aux IA qui la voient. Dans le cadre de ce projet, un *Custom Script* a été utilisé pour choisir de manière aléatoire une *Navigation Target*

Pour la création d'une *Custom Action*, il suffit de se rendre dans le *Behavior Tree* et de faire :

1. Clique droit -> *Create* -> *Action* -> *Custom Action*
2. Sélectionner le nouvel élément et cliquer sur le menu déroulant en regard de *Class* -> *Create Custom Action*

Il aurait aussi été possible d'utiliser l'élément *Random*, (clique droit *Create* -> *Decision* -> *Random*) mais cela encombrerait trop le *Behavior Tree*, de plus, d'un point de vue évolutivité cela n'est pas optimal.

5.2 Collision curseur-model 3D

Dans les tutoriels de formation sur le *Hololens* de *Microsoft*, une chose importante qui n'est pas précisée est la collision entre le curseur qui représente le regard de l'utilisateur du *Hololens* (que nous appellerons tout simplement curseur par soucis de simplicité) et les hologrammes projetés. En effet dans le tutoriel 210³ au chapitre 2 est abordé le feedback entre le curseur et l'hologramme qui est focus par l'utilisateur. En temps normal le curseur est blanc, et quand il entre en collision avec un hologramme, il change d'apparence, et devient un cercle bleu comme sur l'image donnée ci-dessous en 5.12 . Ce qui n'est pas précisé dans le tutoriel cependant, c'est que l'hologramme du tutoriel possède des *Components* nommés les *Colliders*. Ces composants permettent de réaliser la collision avec le curseur, nous pouvons les observer en vert sur l'image en 5.13 et 5.14. Pour résoudre ce problème nous devrons nous-même ajouter ces *Colliders*. L'explication est donnée plus bas, dans ce chapitre.

Figure 5.12 – Collision fonctionnelle



Pour cela nous allons utiliser un modèle 3D déjà présent dans le projet de base réalisé sur l'*Occulus Rift*, *VRKinectAmokShooter*, il s'agira du *Suitmen*. Dans notre cas nous allons utiliser un seul *Collider*, contrairement à l'*Astroman* qui lui en utilise plusieurs. La procédure est simple, il suffit de procéder de manière suivante :

1. Sélectionner notre modèle 3D
2. Dans l'*Inspector* cliquer sur *Add Component*
3. Ajouter un *Capsule Collider*
4. Régler la hauteur/largeur, décalage pour que cela convienne au mieux avec notre modèle 3D

³ https://developer.microsoft.com/en-us/windows/holographic/holograms_210

Figure 5.13 – Colliders de l'Astroman



Une fois l'opération terminée, nous aurons un résultat comme sur l'image en 5.12 - la capsule verte autour du *Suitman* représente la zone où notre curseur apparaîtra en bleu.

Par soucis de temps, nous ne ferons pas de modification sur le *Capsule Collider*, pour cela il est nécessaire de se référer au tutoriel mis à disposition par *Unity* intitulé *Unity 4.0 - Mecanism Animation Tutorial*⁴

⁴<https://www.youtube.com/watch?v=Xx21y9eJq1U>

Figure 5.14 – Collider du Suitman



Scénarios-maquettes

Ce chapitre aborde et décrit les différents scénarios-maquettes qui ont été développés afin de découvrir plus en détail les possibilités de *Unity* et du *Hololens*. Ces scénarios-maquettes ont aussi été réalisés dans le but d'être un fil conducteur pour la réalisation du projet. Chaque scénario peut être vu comme une version X.X du projet. En plus de ces deux utilités, ces scénarios-maquettes ont aussi pour but de jouer un rôle de mini-projet pour avoir une ligne d'apprentissage plus dirigée et plus structurée, ainsi l'erreur qui a été faite au début du projet (qui était de vouloir tout découvrir sans définir un réel but a poussé un apprentissage partiel et très disparate des sujets/points à apprendre et à découvrir) n'a pas été reproduite.

Ce chapitre ne traite pas la partie technique, cependant des points techniques spécifiques comme l'utilisation des IA dans *Rain AI* sont abordés dans les deux chapitres suivants.

6.1 Projet 1 : Rain AI

6.1.1 Objectifs

Ce premier projet a pour but de se familiariser avec le *Plug-in Rain AI* et d'adapter la matière vue dans les tutoriels de *Microsoft* sur le *Hololens* ainsi que les différents tutoriels *Rain AI*.

6.1.2 Description du projet

Les points suivants sont les objectifs de ce mini-projet :

- Interaction entre le policier et un hologramme (via le *Tap Movement*)
- Réaction de l'hologramme à l'action du policier
- Pathfinding de l'IA

L'IA de ce scénario sera très basique : pas de détection du son, pas d'interaction avec le jeu lui-même ou entre les IA (par exemple la peur si le tireur est à proximité). Il se déroulera de la manière suivante :

Une personne est paralysée dans une pièce d'un bâtiment, le policier doit trouver la personne et la toucher, ce qui déclenchera le "réveil" de la personne paralysée qui prendra le chemin le plus court vers la sortie.

6.2 Projet 2 : Shooter

6.2.1 Objectifs

L'objectif de ce scénario-maquette est d'explorer plus en profondeur les possibilités offertes par *Rain AI*. Il a pour but de prendre connaissance du système de détection et de *Memory* de *Rain AI*.

6.2.2 Description du projet

Ce scénario est une extension du premier, il reprend la même scène, et possède des IA plus complètes. Dans ce scénario, nous avons deux IA. La première est celle d'un civil normal, faisant les cents pas devant un bâtiment. S'il voit le forcené/shooter, il prend peur et court se mettre à l'abri, caché dans une pièce du bâtiment, que l'on nommera *Hiding Zone*. Dans le cas où le civil prendra la fuite vers la *Hiding Zone*, le travail du policier sera de retrouver le civil et d'interagir avec lui affin de le rassurer. S'il agit de la sorte, le civil se sentant rassuré, se dirigera vers la *Safe Zone*.

Dans le cas où le joueur/policier entre en contact avec le civil avant le forcené/shooter, le civil part en direction de la *Safe Zone*, zone où il sera considéré comme sauvé du forcené/shooter.

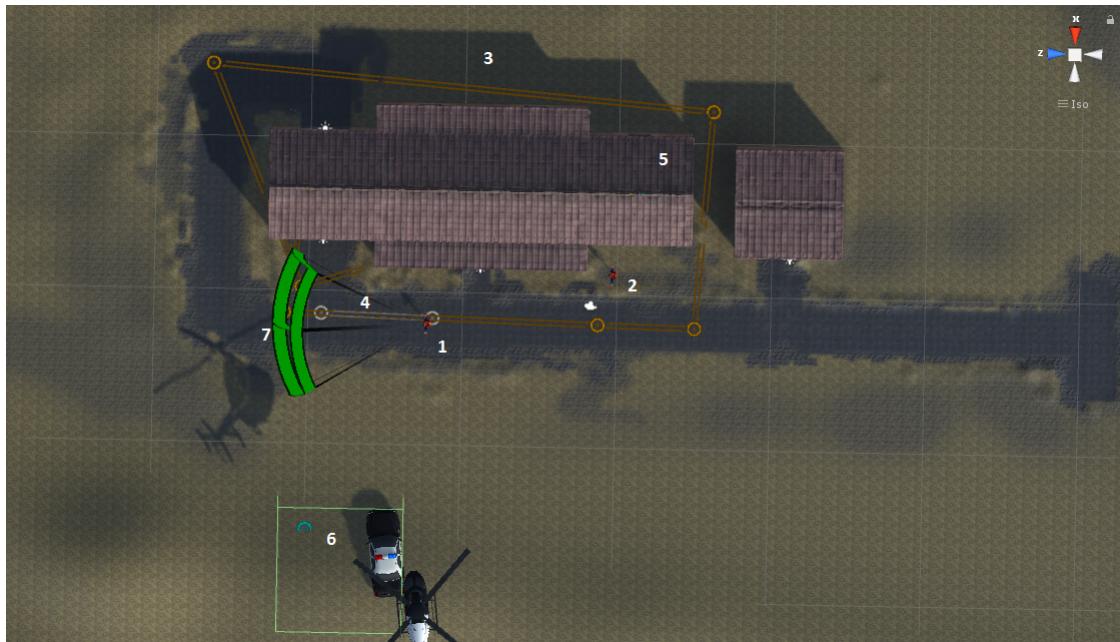
La seconde IA est celle du forcené/shooter, elle patrouille sur la map. Si elle voit le policier, elle se fige. Le policier peut alors interagir avec elle et en effectuant une commande gestuelle, peut l'arrêter. Dans ce scénario, nous partons du principe que nous avons un forcené/shooter de type coopératif qui ne va pas chercher à fuir ou à être menaçant avec le policier.

Cette liste décrit les différents éléments du scénario. (6.1 - Scénario 2)

1. Civil
2. Forcené/shooter
3. Chemin du forcené/shooter (orange)
4. Chemin du civil (blanc)
5. Hiding zone
6. Safe Zone
7. Vue du civil

Pour une meilleure compréhension des actions/événements possibles dans ce scénario, voici une liste plus détaillée des différentes actions/interactions/événements.

Figure 6.1 – Scénario 2



6.2.3 Civil

État normal

Dans son état de base le civil n'a pas vu le forcené/shooter, il est tranquille et fait les cents pas (chemin blanc sur la figure 6.1 point 4). A partir de là deux actions sont possibles :

- Le policier interagit avec le civil avant que le civil ne voie le forcené/shooter
 - Le civil passe en état "en sauvetage" et se dirige vers la *Safe Zone*
- Le civil voit le forcené/shooter qui n'est pas en état "arrêté"
 - Le civil court vers la *Hiding Zone* (5) (coin supérieur droit du grand bâtiment) et passe en état de panique

État de panique

Cet état est enclenché dès que le civil voit le forcené/shooter qui n'est pas en état "arrêté". Il va dans un 1er temps se mettre à courir vers la *Hiding Zone*, puis une fois arrivé il s'immobilisera. A partir de là une action est possible :

- Le policier interagit avec le civil paniqué qu'il voit en train de courir vers la *Hiding Zone* ou qu'il y soit déjà.
 - Le civil passe en état "en sauvetage" et se dirige vers la *Safe Zone*

État en sauvetage

Cet état est enclenché quand le policier interagit avec le civil, ce dernier va courir vers la *Safe Zone*. Durant cet état, deux actions sont possibles :

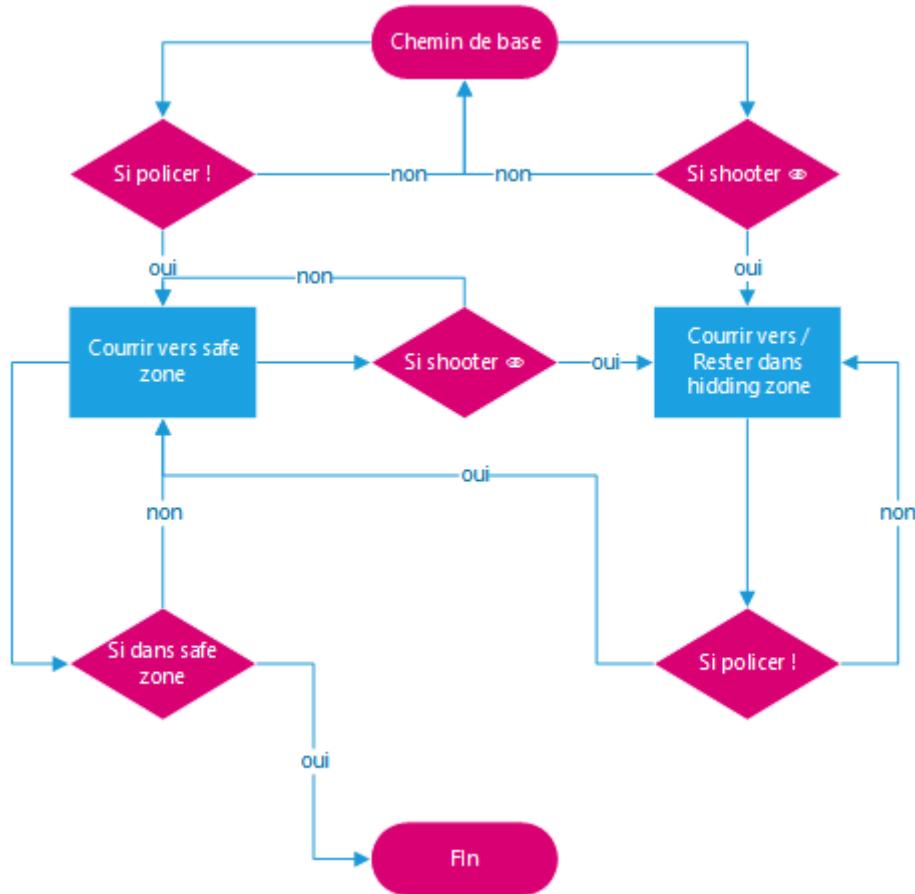
- Le civil voit le force/né/shooter qui n'est pas en état "arrêté"
 - Le civil court/retourne vers la *Hiding Zone* (coin supérieur droit du grand bâtiment) et (re)passe en état de panique
- Le civil atteint la *Safe Zone*
 - Il passe en état "terminé", c'est-à-dire que plus rien ne peut lui arriver

État terminé

Cet état est enclenché si le civil est arrivé à se rendre dans la *Safe Zone*. Depuis ce moment, plus aucune action n'est possible.

Un diagramme représentant les différents cheminements possibles se trouve en figure 6.2. Le "!" représente une interaction et l'œil représente la vision

Figure 6.2 – Diagramme de décision du civil



6.2.4 Forcené/Shooter

État normal

Cet état est l'état de base du forcené/shooter au début de la partie. Il court sur le chemin orange. A partir de cet état une seule action est possible :

- Le forcené/shooter voit le policier
 - Le forcené/shooter se fige, si le policier interagit avec lui, il passe en état "arrêté"

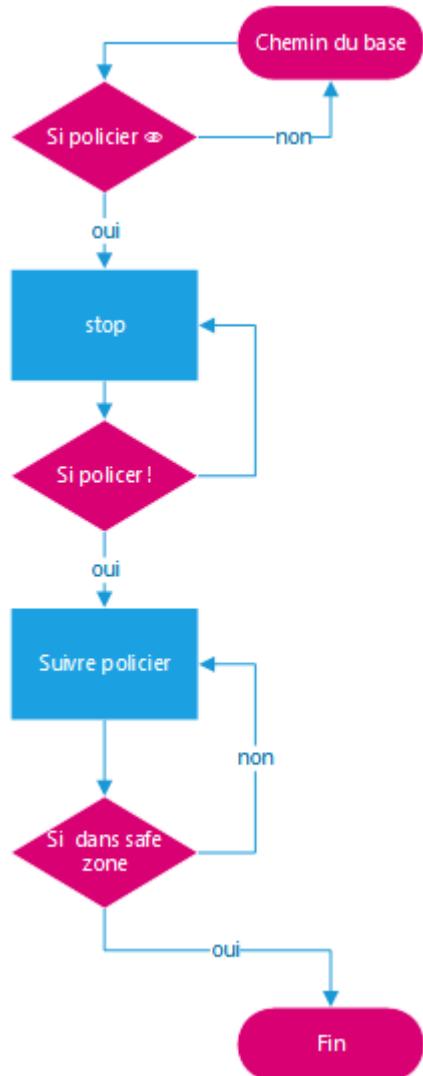
État arrêté

Cet état est déclenché si le policier interagit avec le forcené/shooter après que ce dernier l'ait vu. A partir de ce moment le forcené/shooter va suivre le policier. Une seule action sera possible

- Le shooter entre dans la *Safe Zone*

- Le forcené/shooter passe en état terminé, il ne suit plus le policier

Figure 6.3 – Diagramme de décision du forcené/shooter



6.3 Projet 3

6.3.1 Objectifs

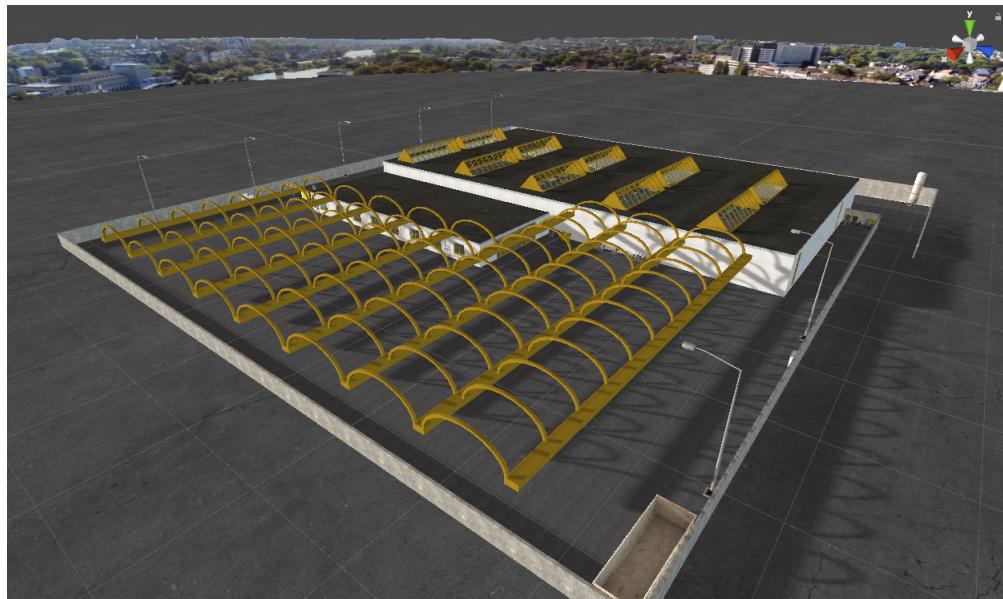
L'objectif de ce projet est de faire ce qui a été fait dans les projets précédents mais de manière plus conséquente : davantage de civils et tireur avec une IA plus poussée comme la possibilité de fuite. Le but ici est de faire une toute première version du projet avec ce qu'on pourrait considérer comme le minimum vital.

6.3.2 Description du projet

Le projet 3 a été un tournant majeur pour la réalisation du projet. Contrairement aux deux projets précédents, le projet 3 et tout ce qui suivra pourra être perçu comme une version 0.X du projet final. Là où les deux premiers projets pouvaient être vus comme un simple petit projet d'apprentissage avec des éléments réutilisables, le projet 3 sera l'application de ce qui a été appris.

A partir du projet 3, tout ajout sera directement ajouté à ce projet sans pour autant faire un nouveau projet avec un objectif cible. Un autre changement majeur à partir du projet 3 est le changement de scène. Nous passons de la *Map* illustrée en 6.1 représentant un hangar abandonné à un supermarché illustré à la figure 6.4. Le scénario prendra fin quand toutes les IA seront dans les carrés verts présents sur la figure 6.5 (1).

Figure 6.4 – Scène utilisée à partir du projet 3



Sur l'image ci-dessous nous avons une vue aérienne de la nouvelle *Map* avec différents points clés.

1. Safe Zone (carrés vert et points rouges)
2. Points de passage de civil en état normal (points blancs)
3. Points de passage du forcené/shooter (points rouges)
4. Points de fuite intermédiaire (pour éviter une fuite en file indienne des civils)
5. Points de fuite du forcené/shooter (points verts)

Figure 6.5 – Scène utilisée à partir du projet 3



6.3.3 Civil

La première version de ce civil est dépourvue d'interaction avec le policier, il s'agit d'un simple civil faisant ses courses en se déplaçant de manière aléatoire dans le supermarché jusqu'à ce qu'il voie le forcené/shooter. A partir de là deux actions sont possibles : passer en état de panique ou passer en état de fuite. Dans le premier cas, le civil se fige et plus rien n'est possible. Dans le deuxième cas, le civil choisit un point de fuite aléatoire et s'y dirige en courant.

État normal

Se déplace de manière aléatoire parmi X points d'arrêts possibles. Quand le civil arrive à un des points, il s'arrête un certain nombre de secondes puis repart vers un autre point. A partir de là deux états sont possibles, choisis de manière aléatoire (de manière équiprobable), les deux déclenchés par la vue du forcené/shooter :

- Passer en état de panique
- Passer en état de fuite

État de panique

L'état de panique arrête immédiatement le civil dans son déplacement actuel, il n'est plus capable de se déplacer.

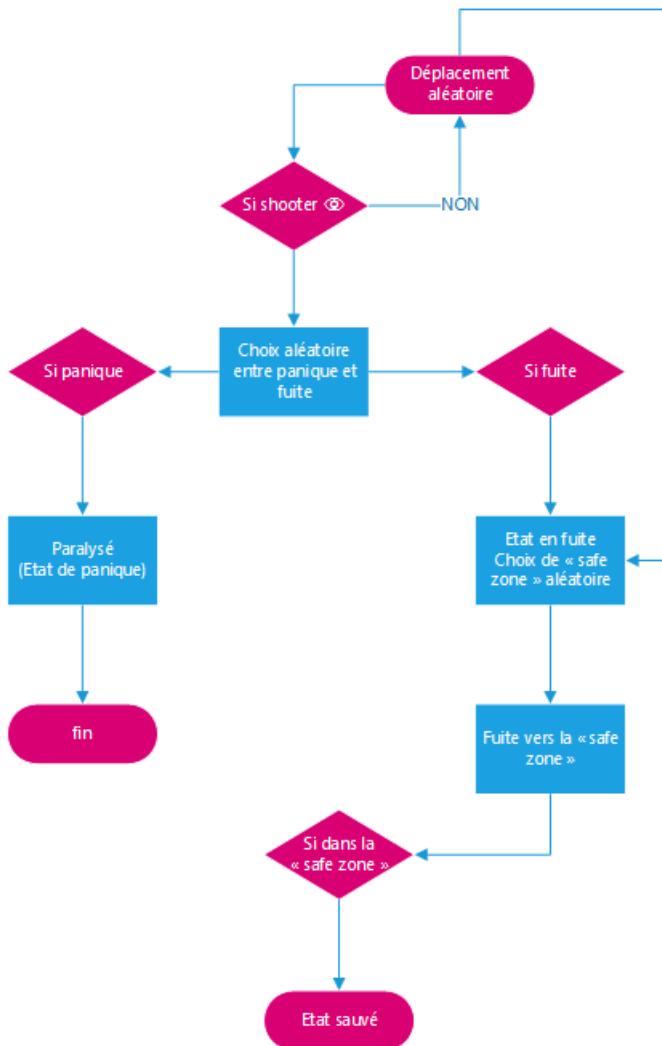
État de fuite

Dans cet état, le civil prend conscience du danger et quitte le magasin en courant. Il ira se réfugier dans un endroit à l'extérieur du magasin, choisi de manière aléatoire.

État de sauvé

État déclenché quand le civil entre dans la *Safe Zone*. A partir de là, plus aucune action n'est possible sur le civil.

Figure 6.6 – Diagramme utilisé pour le civil pour le projet 3.0



6.3.4 Forcené/Shooter

Dans ce scénario le forcené/shooter se déplace de manière aléatoire dans le magasin et, à la vue de la police, il prend la fuite et va se cacher à un emplacement choisi au hasard. Le policier devra le retrouver et, si ce dernier passe dans son champ de vision, il reprendra la fuite, et cela 3 fois. Au bout de la 3ème fois, il arrêtera de fuir. Si le policier arrive à interagir avec lui sans passer dans le champ de vision du forcené/shooter, ce dernier se rendra sur le champ.

État normal

Le tireur patrouille parmi X points choisis de manière aléatoire dans le supermarché, à partir de là un état est possible

- Passer en état en fuite

État de fuite

Le forcené/shooter court se cacher dans un endroit choisi aléatoirement dans le magasin. A noter que même s'il voit le policier durant sa fuite, cela ne l'affectera en rien. A partir de cet état, un seul état est possible, celui de l'état caché.

État caché

Le forcené/shooter arrive à cet état quand il a atteint une cachette. Par la suite, deux états sont possibles

- Passer en état en fuite
- Passer en état stoppé

État stoppé

Ce état est atteint quand le forcené/shooter voit pour la 3ème fois le policier. C'est un état d'abdication à la fuite, mais pas arrêté pour autant. Pour passer de l'état stoppé à l'état arrêté, il faudra que le policier interagisse avec le forcené/shooter via le *Tap Movement*

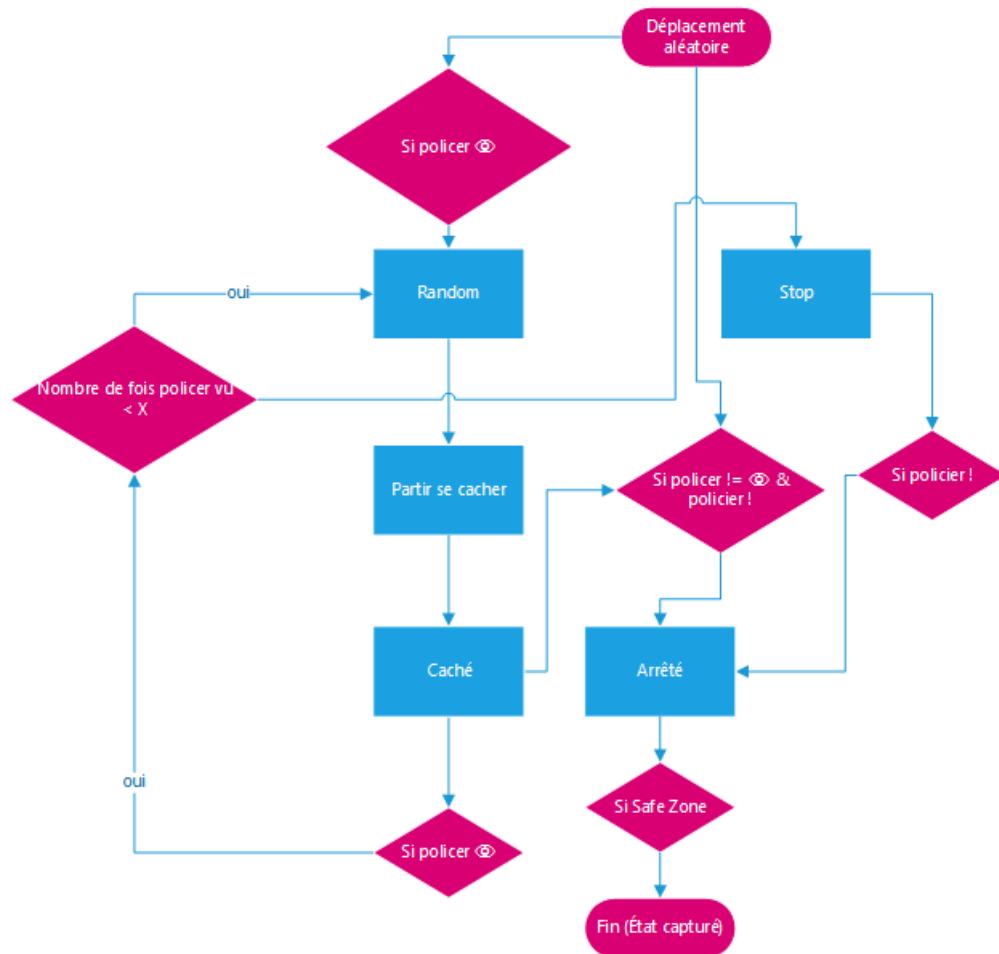
État arrêté

Cet état peut être atteint si le policier interagit avec le forcené/shooter quand celui-ci a déjà vu le policier trois fois ou si le policier interagit avec le forcené/shooter sans que celui-ci ne l'ait vu auparavant. Si le policier agit de la sorte le forcené/shooter suivra le policier jusqu'à ce que ce dernier le mène à la *Safe Zone*.

État capturé

Cet état est atteint quand le forcené/shooter est arrêté et mené vers la *Safe Zone*. Cet état est l'état final du forcené/shooter, s'il est atteint, plus aucune action ne sera possible sur lui.

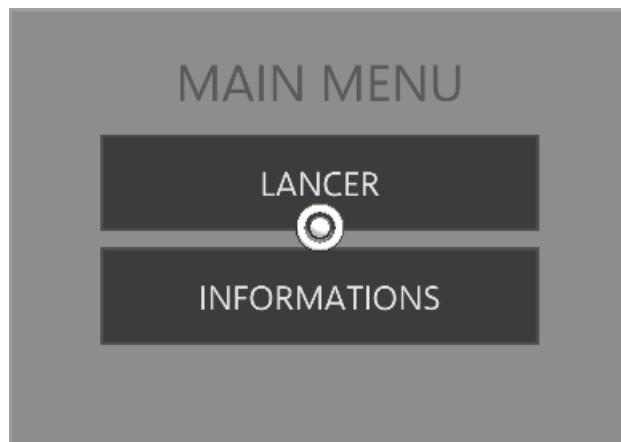
Figure 6.7 – Diagramme utilisé pour le forcené/shooter pour le projet 3.0



6.4 Projet 3.1 : HUD

Dans cette version, l'objectif est d'ajouter un menu avant de lancer le jeu. La création de ce menu est relativement simple au premier abord car un *Toolkit* nous est fourni par *Unity*, mais s'avère plus ardue au final. Une explication plus détaillée de cette situation se trouve dans le chapitre *Problèmes*. Le menu se présente sous la forme suivante : il possède deux boutons, un permettant de lancer la simulation, un autre permettant à l'utilisateur de prendre connaissance des possibilités au sein du jeu, telles que les commandes vocales et les commandes gestuelles ainsi que différentes informations sur les entités présentes dans le jeu.

Figure 6.8 – Menu de départ



6.5 Projet 3.2 : IA Avancées

6.5.1 Objectifs

L'objectif ici est de compléter les IA faites jusqu'à présent et d'en ajouter des nouvelles. Deux nouvelles IA seront ainsi rajoutées. La première sera nommée *Adam* (ce nom provient du nom du modèle 3D utilisé), et la deuxième sera nommée *Michael* pour les mêmes raisons que la première.

6.5.2 Adam

Le comportement d'*Adam* est un comportement héroïque, il aide la police. Tout comme les civils de base, il se promène de manière aléatoire dans le supermarché. S'il voit le forcené/shooter, il se mettra à courir dans le supermarché à la recherche d'un civil paniqué. Lorsqu'il en trouvera un, il se dirigera vers lui, restera à ses côtés durant 5 secondes, puis courra vers la *Safe Zone*. Le civil en question passera de l'état de panique à l'état de fuite.

Tout comme les autres civils, si le policier/joueur effectue un *Tap Movement* sur *Adam*, ce dernier passe en état de fuite.

État normal

Se déplace de manière aléatoire parmi X points d'arrêt possibles. Quand Adam arrive à un des points, il s'arrête pendant une courte durée puis repart vers un autre point. A partir de là deux états sont possibles, le premier est déclenché par la vue du force-né/shooter, le second par la vue d'un civil en panique.

- État de recherche de personne
- État aide une personne

État de recherche de personnes

Cet état est déclenché par la vue du force-né/shooter. Adam se déplace alors dans le supermarché durant un certain temps à la recherche de personnes en état de panique. S'il en trouve une, il passera à l'état d'aide d'une personne.

État aide une personne

Adam se rapproche d'une personne en état de panique, il reste à ses côtés durant une courte durée, puis, une fois la personne rassurée, il passera en état de fuite qui est le même que l'état de fuite des autres civils.

État de fuite

Dans cet état, Adam prend conscience du danger et quitte le magasin en courant, il ira se réfugier dans un endroit à l'extérieur du magasin choisi de manière aléatoire. Une fois la *Safe Zone* atteinte, il passera à l'état sauvé qui est le même que celui des civils standards.

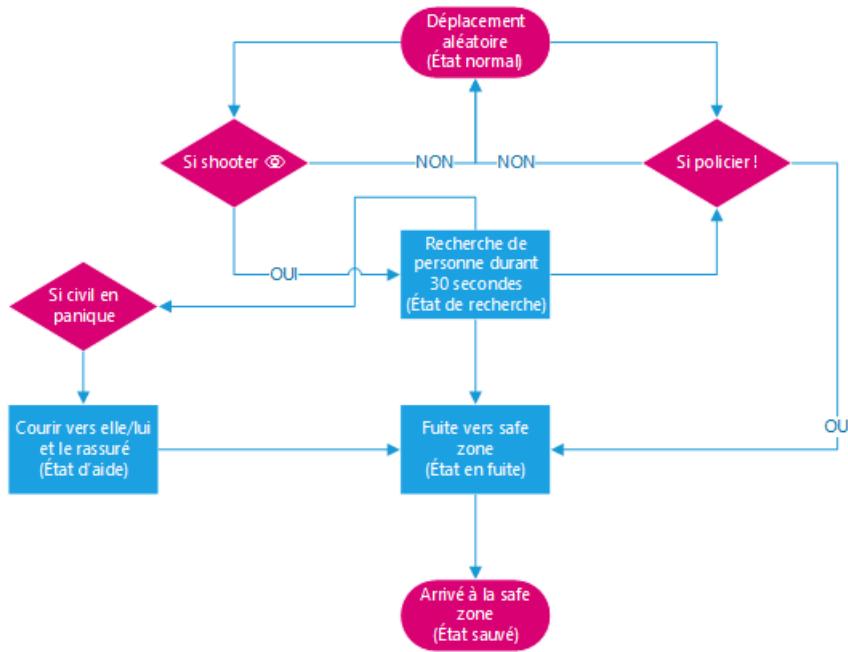
État de sauvé

État déclenché quand Adam entre dans la *Safe Zone*. A partir de ce moment, plus aucune action n'est possible.

6.5.3 Michael

Michael est la deuxième nouvelle IA implémentée dans cette version 3.2. Tout comme Adam, il a un comportement héroïque. Il se promène dans le supermarché comme toutes les autres IA. Au moment de voir le tireur, il se fige et commence à crier au danger. Les civils de base qui sont proches de Michael seront avertis du danger, et, si ils sont en état normal, ils passeront à l'état de fuite. Michael crie durant 30 secondes, puis passe en état de fuite.

Figure 6.9 – Diagramme de Michael



Tout comme Adam et les autres civils, si le policier/joueur effectue un *tap movement* sur Michael, ce dernier passe en état de fuite

État normal

Michael se balade dans le supermarché tout comme les autres civils et, s'il voit le forcené/shooter, une action est possible : Prévenir les autres (état aide les personnes)

État aide les personnes

Michael se fige et commence à avertir les personnes aux alentours du danger durant 30 secondes. Une fois ce délai passé, Michael passe en état de fuite.

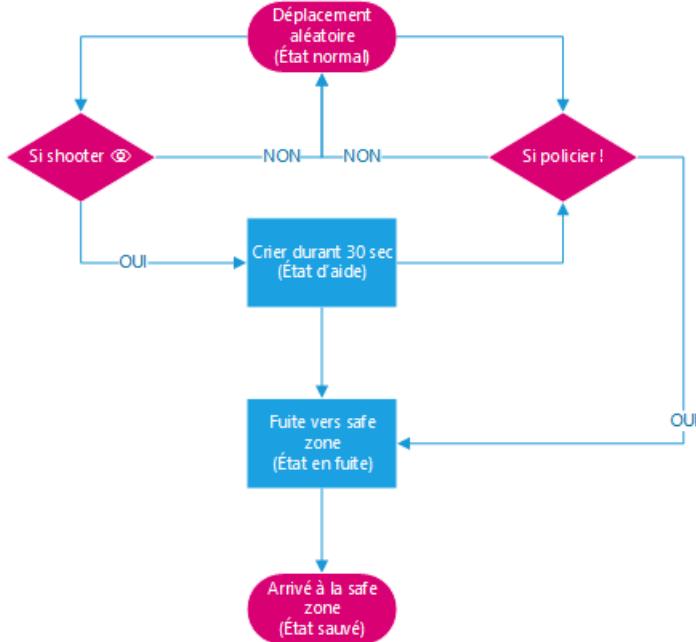
État de fuite

Dans cet état Michael prend conscience du danger et quitte le magasin en courant. Il ira se réfugier dans un endroit à l'extérieur du magasin choisi de manière aléatoire. Une fois la *Safe Zone* atteinte, il passera à l'état sauvé qui est le même que celui des autres civils.

État de sauvé

État déclenché quand Michael entre dans la *Safe Zone*. Une fois cet état atteint, plus aucune action n'est possible.

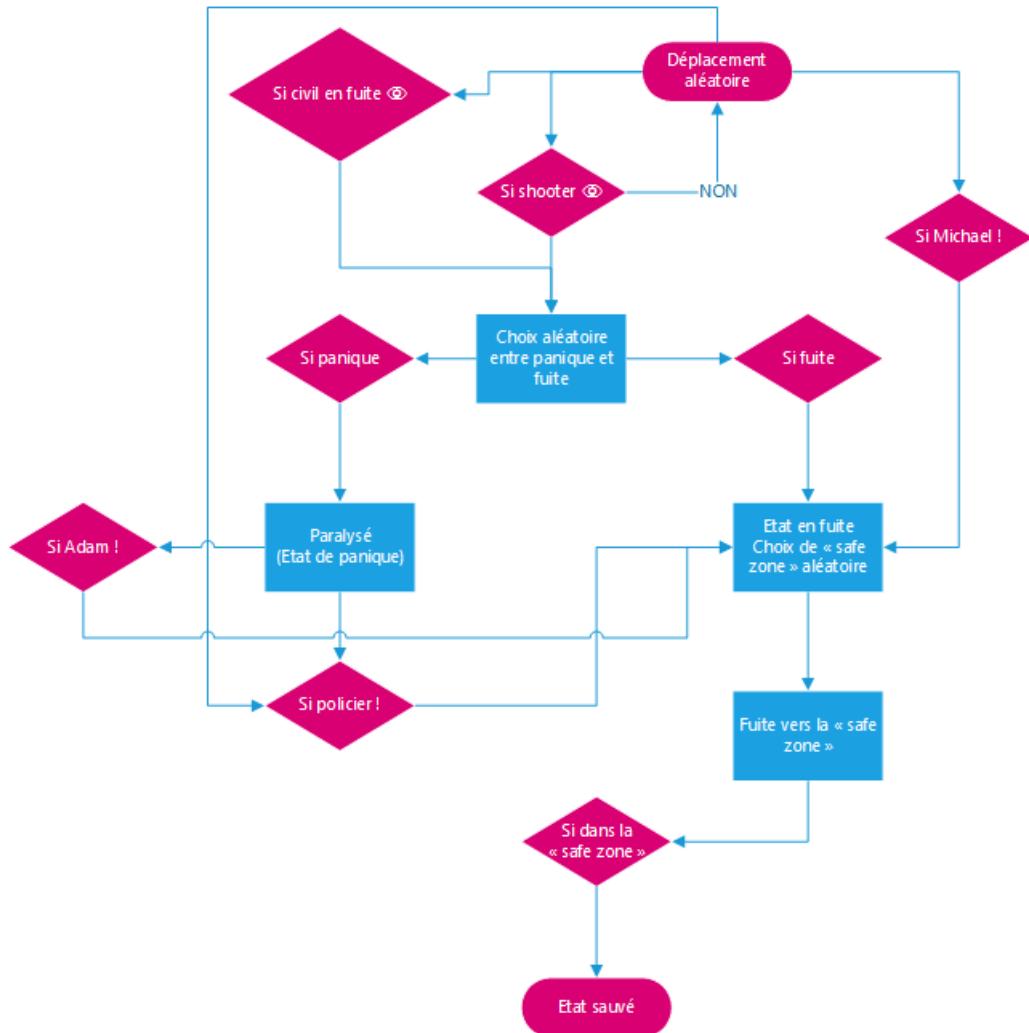
Figure 6.10 – Diagramme de Michael



6.5.4 Civil avancé

Dans cette nouvelle version, le civil est un peu plus intelligent. Il est désormais possible d'interagir avec le civil à tout moment via le *Tap Movement*. Le civil est aussi à l'écoute d'Adam et Michael. Si le civil est en état de panique et qu'Adam interagit avec lui, il passera en état de fuite. Si le civil entre dans la zone de Michael, il prendra la fuite s'il est en état normal. La dernière chose qui a été ajoutée au civil est la possibilité de fuir/paniquer si ce dernier voit un autre civil en fuite ou en panique.

Figure 6.11 – Diagramme du civil pour le projet 3.2



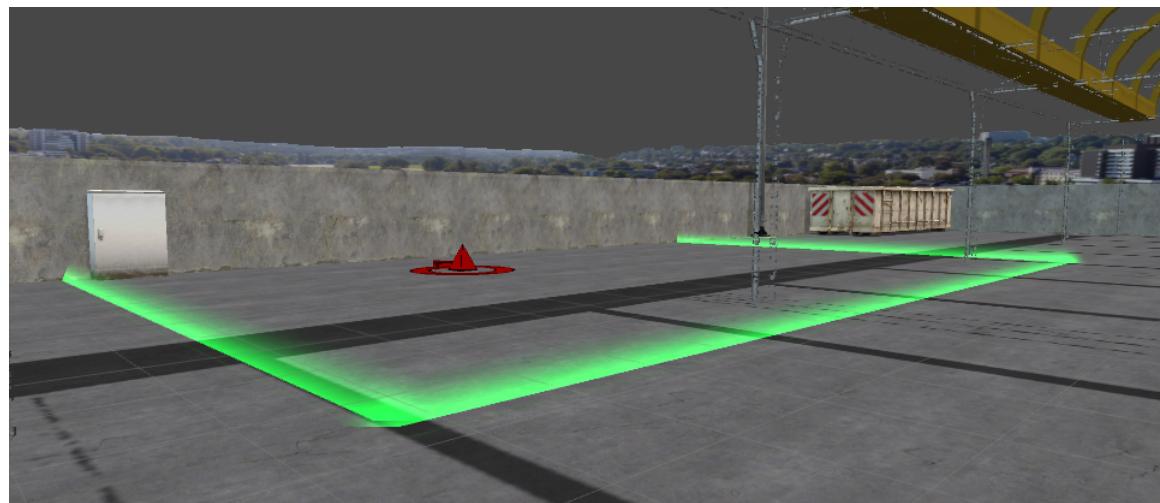
6.5.5 Shooter avancé

La seul modification sur le forcené/shooter est qu'il est possible de l'arrêter quand il est en état "normal" si nous arrivons à interagir avec lui sans qu'il nous voie.

6.5.6 Safe Zone

Pour que le joueur/policier puisse plus facilement repérer la *Safe Zone*, des néons de couleur ont été placés sur le sol.

Figure 6.12 – Safe Zone



Réalisation

Ce chapitre traite de la réalisation du projet, il est complémentaire au chapitre précédent traitant des scénario-maquettes, mais il les aborde d'un côté plus technique. Il met en avant et explique les diagrammes de classe, reprend également une explication sur les *Behavior Trees* de *Rain AI* ainsi que les différents choix techniques qui ont été pris. Il traite également des problèmes rencontrés durant l'ensemble de la réalisation du travail de Bachelor ainsi que les éventuelles solutions trouvées.

7.1 Classes et diagrammes de classes - RAINAction

7.1.1 ReassuringCivilian

Cette classe hérite de *RAINAction*, elle est utilisée par Adam quand il arrive à portée d'un civil en panique. Elle arrête le déplacement d'Adam et appelle la fonction *SayReassuring*.

Elle ne possède qu'une seule méthode :

- public override ActionResult Execute(AI ai) : fonction qui s'exécute quand la classe est appelée par un *Behavior Tree*. Cette fonction arrête le déplacement de l'IA et rassure le civil pris pour "cible". Le civil passera d'un état de panique à un état de fuite vers la *Safe Zone*.

7.1.2 RandomWayPoint

Cette classe hérite de *RAINAction*, elle est utilisée par l'ensemble des IA et permet un déplacement aléatoire parmi une liste de *Navigation targets*. Cette liste de *Navigation Targets* est basée sur un *Empty GameObject* contenant différents *Navigation Targets*. Cet objet est passé en paramètre à *HumanAI* via *Unity*.

Le grand avantage avec cette méthode, plutôt que de faire *Random* dans le *Behavior Tree*, est que nous avons une meilleur évolutivité. Si nous voulons ajouter un *Navigation Targets*, nous l'ajoutons simplement dans le *Empty GameObject* contenant l'ensemble de *Navigation Targets* comme enfants. De plus via cette méthode, nous ne prenons qu'une seul ligne dans le *Behavior Tree*, celle de l'appel de l'action.

Elle ne possède qu'une seule méthode :

- public override ActionResult Execute(AI ai) : méthode qui s'exécute quand la classe est appelée par un *Behavior Tree*. Cette fonction choisit un nouveau *Navigation Target* à chaque fois qu'elle est appelée.

7.1.3 CheckPanicRun

Cette classe hérite de *RAINAction*, elle est utilisée par l'ensemble des civils de manière continue quand ils sont dans leur état normal. Elle permet de vérifier si un civil vu par le civil est en état de panique ou de fuite.

Elle ne possède qu'une seule méthode :

- public override ActionResult Execute(AI ai) : méthode qui s'exécute quand la classe est appelée par un *Behavior Tree*. Elle vérifie si le civil vu par le détecteur est en état de fuite ou de panique, et, si c'est le cas, la *Memory targetPanic* sera modifiée et vaudra *true*.

7.1.4 CheckPanic

Cette classe hérite de *RAINAction*, elle est utilisée par Adam pour vérifier si le civil qu'il voit est paniqué ou non.

Elle ne possède qu'une seule méthode :

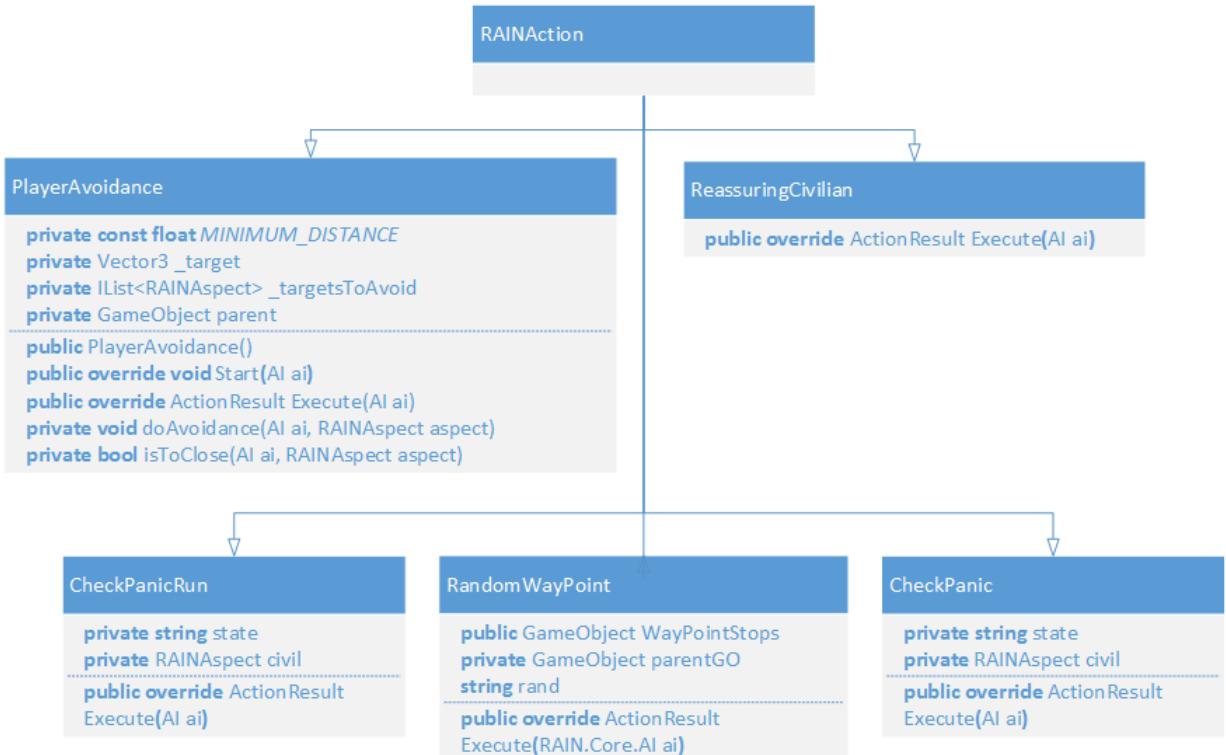
- public override ActionResult Execute(AI ai) : méthode qui s'exécute quand la classe est appelée par un *Behavior Tree*. Elle vérifie si le civil vu par le détecteur est en état de panique, et, si c'est le cas, la *Memory targetPanic* sera modifiée et vaudra *true*.

7.1.5 PlayerAvoidance

Cette *Custom Action*, permet d'éviter que deux IA se bloquent entre elles. Il vient d'un *topic* sur le forum de *Rain AI*¹. Il ne sera donc seulement mentionné et afficher dans le diagramme de classe.

¹<http://legacy.rivaltheory.com/forums/topic/ai-collision-solutions/page/2/>

Figure 7.1 – Diagramme de classe des Customs Actions



7.2 Classe Keywords

7.2.1 KeywordHandler

Cette classe est créée dans le seul but de centraliser les fonctions appelées par le *KeywordManager*. Ainsi dans nos composants *KeywordManager*, et *Keywords and Response* nous n'utiliserons qu'un seul fichier. La centralisation permet de simplifier la compréhension.

Cette classe ne possède qu'un seul attribut :

- private GazeManager gm : attribut représentant le *GazeManager* de notre *GameObject*

Elle possède trois méthodes

- public void OnSayPolice(): méthode appelée quand l'utilisateur dit le mot "Police", si l'utilisateur est en train de viser un civil, la méthode *OnPolice* sera appelée par réflexion. Cette méthode fait passer l'état du civil à un état de fuite vers la *safe zone*.
- public void OnSayStop(): méthode appelée quand l'utilisateur dit le mot "Stop", si l'utilisateur est en train de viser le forcené/shooter, la méthode *OnStop* sera appelée par réflexion. Cette méthode fait passer l'état du forcené/shooter à un état "arrêté".

- public void OnSayReset(): méthode appelée quand l'utilisateur dit le mot "Reset". Elle effectue un changement de scène, nous ramenant vers le menu de départ. La partie en cours n'est pas sauvegardée.

7.3 Classes et diagramme de classes - IA

Ce diagramme représente les différentes classes créées pour la réalisation des différentes IA. Une première classe principale nommée *HumanAI* hérite directement de la classe *MonoBehaviour* de *Unity*.

7.3.1 HumainAI

La classe *HumanAI* est la classe parente de toutes nos IA. Elle implémente deux interfaces permettant l'interaction gestuelle et le *Focus*: *IInputClickHandler*, *IFocusable*. Ces interfaces nous viennent du *Holotoolkit*. Elle possède les attributs, les constantes et les méthodes minimales pour le fonctionnement de nos IA; à savoir:

Pour les constantes

- protected const float ANIM_SPEED = 0.5f: vitesse des animations de marche
- protected const int ANIM_SPEED_RUN = 2: vitesse des animations de course
- protected const int NORMAL_SPEED = 1: vitesse de déplacement de marche (Pour *Rain AI*).
- protected const int RUN_SPEED_MIN = 3: vitesse de course minimale
- protected const int RUN_SPEED_MAX = 6: vitesse de course maximale
- private const float THRESHHOLD = 0.01f: valeur minimale pour savoir si notre IA bouge ou non. Si delta de *oldLocation* et la position actuelle est supérieure à *THRESHHOLD* alors l'IA est en mouvement.
- protected const float MASS_MIN = 1: masse minimale de notre IA
- protected const float MASS_MAX = 10: masse maximale de notre IA

Les deux dernières constantes sont là pour gérer les collisions: une des IA prend le dessus et pousse l'autre, évitant ainsi un blocage qui pourrait survenir lorsque les deux IA ont des valeurs identiques.

Pour les variables

- protected Animator anim : pour accéder aux animations de notre *GameObject*

- protected Rigidbody rigidbody: pour accéder à notre *RigidBody* et modifier la masse de notre IA
- protected AIRig: pour accéder au composant de *Rain AI*
- protected EntityRig tEntity: pour notre entité
- protected Vector3 oldLocation: va de pair avec IsMoving(), permet de savoir si notre IA est en mouvement ou non, ce qui permet par la suite de définir quelle animation jouer.
- public GameObject NavTargetsGO : *GameObject* parent des *Navigation target* qui représente les différents points où peut aller notre IA durant sa phase "normale"
- private bool isFocused : boolean qui vaut *true* s'il est visé par le pointeur du joueur, sinon il vaut *false*

Pour les méthodes et fonctions

- protected void init(): permet d'initialiser les variables pour se garantir des exceptions. De plus, il permet également de créer les *Memory* de *Rain AI*, évitant ainsi de faire les IA à la main via *Unity*.
- public string getState(): permet de modifier l'état de notre IA.
- public void setState(EnumState.EStates state): modifie l'état de notre IA.
- public void OnInputClicked(InputClickedEventData eventData): Implémentation de l'interface *IInputClickHandler* permettant de réagir au *tap movement* de l'utilisateur.
- public void OnFocusEnter(): implémentation de l'interface *IFocusable* permettant de réagir quand notre objet entre en collision avec le curseur
- public void OnFocusExit(): public void OnFocusEnter(): Implémentation de l'interface *IFocusable* permettant de réagir quand notre objet sort de collision avec le curseur
- private void InSafeZone(): fonction appelée par le principe réflexion par notre objet *safe zone* quand une IA entre dans cette dernière
- protected bool IsMoving(): permet de savoir si notre *GameObject* bouge
- protected VisualSensor CreateVisualSensor(bool IsActive, string SensorName, int HorizontalAngle, Vector3 PositionOffset, bool RequireLineOfSight, float range = 10f, Color color = default(Color)): permet de créer un senseur visuel sur notre IA. Fonction créée pour éviter de devoir ajouter à la main un senseur visuel sur chaque IA.
- protected RAINAspect CreateRainAspect(string name): fonction permettant d'ajouter à la main un aspect à notre IA

7.3.2 Civilian

La classe *Civilian* est la classe pour les civils dont le comportement est décrit dans le chapitre des scénarios (à partir du 3ème scénario). Il hérite de la classe principale pour les IAs qui est *HumanAI*.

Pour les variables

- private int randomSpeed: vitesse de déplacement lors de la fuite

Pour les méthodes et fonctions

- private void OnInDanger(): méthode appelée par réflexion quand le civil entre dans la *Trigger* de Michael. Elle modifie l'état du civil s'il n'est pas en panique.
- private void OnInputClicked(): méthode appelée quand nous effectuons un *Tap Movement* sur le civil.
- private void OnPolice(): fonction appelée quand le policier dit "Police"
- private void OnSelect(): fonction appelée par OnInputClicked() permettant de changer l'état de panique du civil en un état de fuite

7.3.3 ShopShooter

La classe *ShopShooter* est la classe de notre forcené/shooter (à partir du 3ème scénario). Elle hérite de la classe *HumanAI*.

Pour les constantes

- private const int NBR_POLICE_SEEN_FOR_SURRENDER = 3 : nombre de fois que le forcené/shooter doit voir le policier avant de se rendre

Pour les méthodes et fonctions

- public void OnInputClicked(): méthode appelée quand nous effectuons un *tap movement* sur le forcené/shooter. Elle appelle la méthode *OnSelect()* qui passe l'état du forcené/shooter à l'état arrêté.
- private void InSafeZone(): méthode appelée quand le forcené/shooter entre dans la *safe zone*; il devient *caught*.
- private void OnSelect(): méthode appelée par OnInputClicked() permettant de changer l'état du forcené/shooter en un état de "suivre le policier".
- private void OnStop(): méthode appelée par réflexion par le *KeywordHandler* lorsque le policier dit "stop" en visant le forcené/shooter avec le pointeur.

7.3.4 Adam

La classe *Adam* est la classe d'un de nos civils héroïques qui aide la police. Tout comme le reste des IA, il hérite également de la classe *HumainAI*.

Pour les variables

- private RAINAspect civil: civil "cible". Il s'agit du civil qui sera aidé.
- private float timeLeft: durée durant laquelle Adam est en recherche de personnes en panique

Pour les méthodes et fonctions

- public void SayReassuring(): méthode appelée par la *Custom action* de *Rain AI*.
- public void OnInputClicked(): méthode appelée quand nous effectuons un *tap movement* sur Adam. Elle appelle la méthode *OnSelect()* qui passe l'état d'Adam à "en fuite".
- private void InSafeZone(): méthode appelée lorsque le forcené/shooter entre dans la *safe zone*, il devient *caught*.
- private void OnSelect(): fonction appelée par *OnInputClicked()* permettant de changer l'état du forcené/shooter en un état de "suivre le policier".

7.3.5 Michael

Michael est le second personnage héroïque, et, tout comme les autres classes de IA, il hérite de *HumainAI*.

Pour les variables

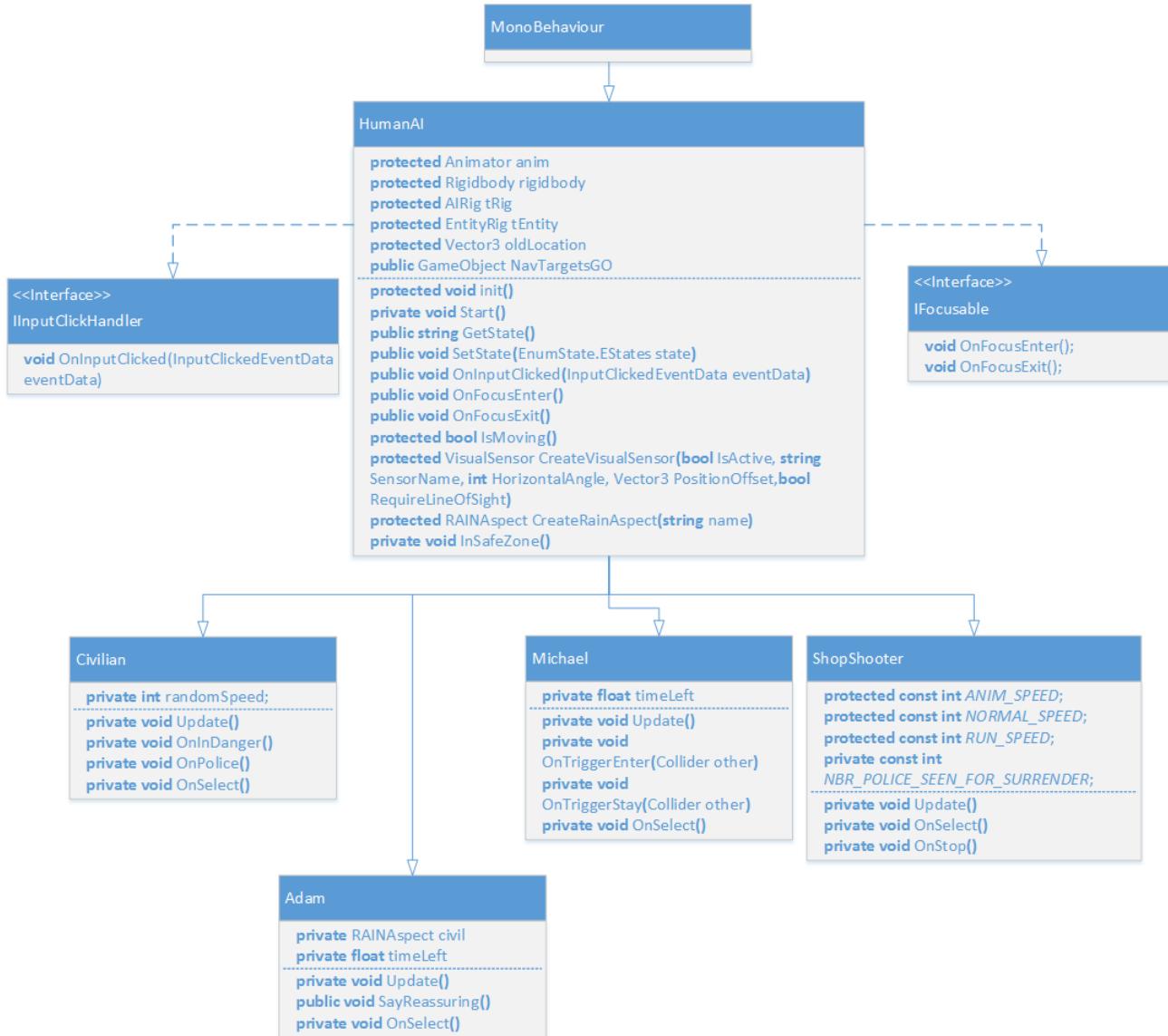
- private float timeLeft: temps restant avant de fuir quand Michael aide

Pour les méthodes et fonctions

- private void OnTriggerEnter(Collider other): *Unity evnet function*, est une fonction qui est appelée quand notre *gameObject* entre en collision avec un autre *gameObject*. Attention: un composant de type *Collider* sur les deux *GameObject* est nécessaire. Dans notre cas, nous utilisons cette méthode quand Michael entre en collision avec un civil. La collision avec ce dernier déclenche l'alerte chez le civil -> appel de la fonction *OnInDanger* du civil via réflexion.

- OnTriggerStay(Collider other) : *Unity Event Function*, même fonction que la précédente à la différence près que celle-ci est appelée continuellement et pas uniquement lors de l'impact. Utilisée dans le cas où un civil est dans la zone et que Michael voit le force/né/shooter.

Figure 7.2 – Diagramme de classes des IA



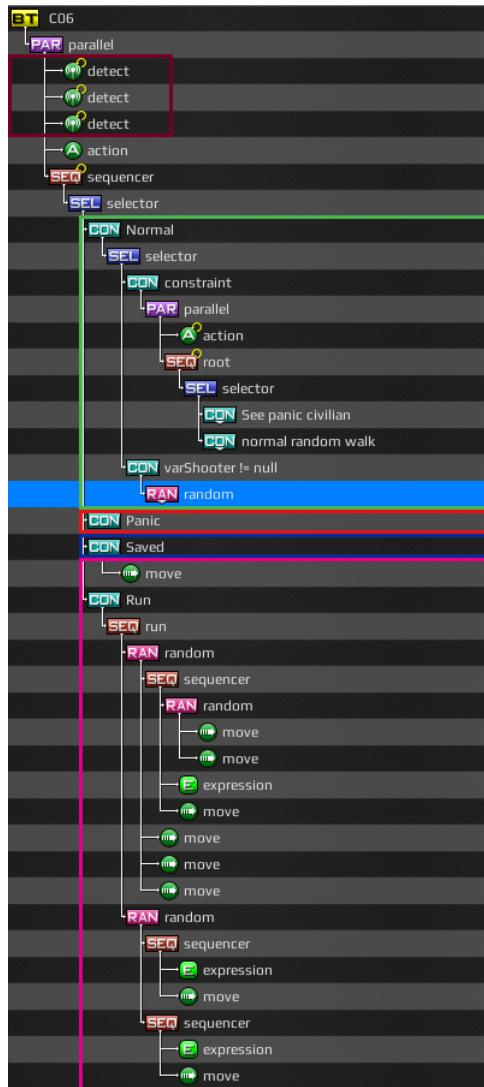
7.4 RainAI

7.4.1 Civilian

Le nom du *Behavior Tree* des civils se nomme **C06**, nom tiré du nom du 1er modèle 3D utilisé pour les civils. Le *Behavior Tree* du civil se trouve sur la figure en 7.3 – Behavior Tree du civil.

- Rouge écarlate: les trois détecteurs de *Visual Aspect*
 - détecteur pour le forcené/shooter
 - détecteur pour les civils en panique/fuite
 - détecteur pour éviter les collisions avec d'autres civils
- En dessous du bloc rouge écarlate : L'action qui lance la *Custom Action - PlayerAvoidance*
- Vert: état normal: nous avons dès le départ deux *Constraints*(élément turquoise nommé [CON] représentant une condition):
 - le premier: effectuera continuellement la vérification des civils éventuels qu'il voit tout en marchant en choisissant un point de destination aléatoire. Si à tout hasard il venait à voir un civil en fuite ou en panique, il entrera dans le bloc *[CON] See panic civilian*, qui choisira de manière aléatoire son futur état (état de panique ou état de fuite).
 - Le second: *[CON] varShooter != null* sera respecté quand le civil verra le forcené/shooter
- Rouge: état de panique: dans ce bloc *Rain AI* n'est responsable de rien
- Bleu: état sauvé: cet état est atteint quand le civil entre dans la *Safe Zone*, après cela il fera un dernier déplacement vers la *NavigationTarget* qui est à l'intérieur de la *Safe Zone* et s'arrêtera à une distance aléatoire. Le but de cette manœuvre est de répartir les IAs dans la *Safe Zone*
- Rose: état de fuite: ce bloc est responsable du chemin que va prendre le civil lors de sa fuite. Noter le nombre *Random* présent, bien supérieur à trois (trois étant le nombre de points de fuite possibles). Ces *Random* supplémentaires sont là pour ajouter des points de fuite intermédiaires, cela dans le but d'éviter un effet de fuite en file indienne.

Figure 7.3 – Behavior Tree du civil



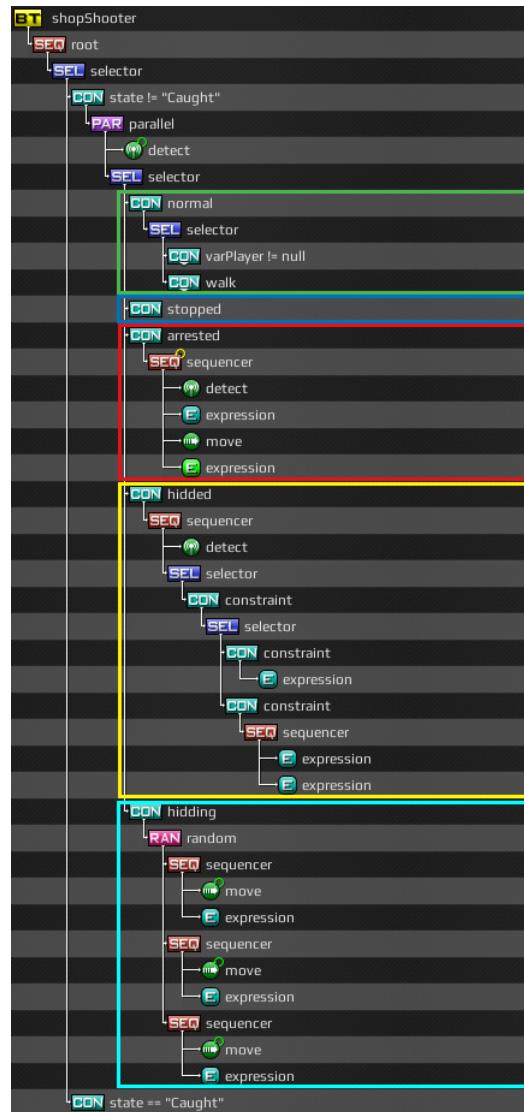
7.4.2 ShopShooter

Le nom du *Behavior Tree* du forcené/shooter se nomme **shopShooter**. Il est illustré à la figure 7.4 – Behavior Tree du forcené/shooter.

- Vert: état normal
- Bleu: état stoppé: le forcené/shooter s'arrête; dans ce cas *Rain AI* ne gère plus rien.
- Rouge: état arrêté: répète continuellement une détection pour récupérer l'emplacement du joueur, se déplace vers lui, puis s'arrête quand il a atteint le joueur
- Jaune: état caché: le forcené/shooter à atteint une de ses cachettes. S'il voit le policier, il prendra la fuite vers une nouvelle cachette

- Bleu clair: état "en train de se cacher": état lancé après avoir vu le policier. Prendra fin lorsqu'une des cachettes sera atteinte.

Figure 7.4 – Behavior Tree du forcené/shooter

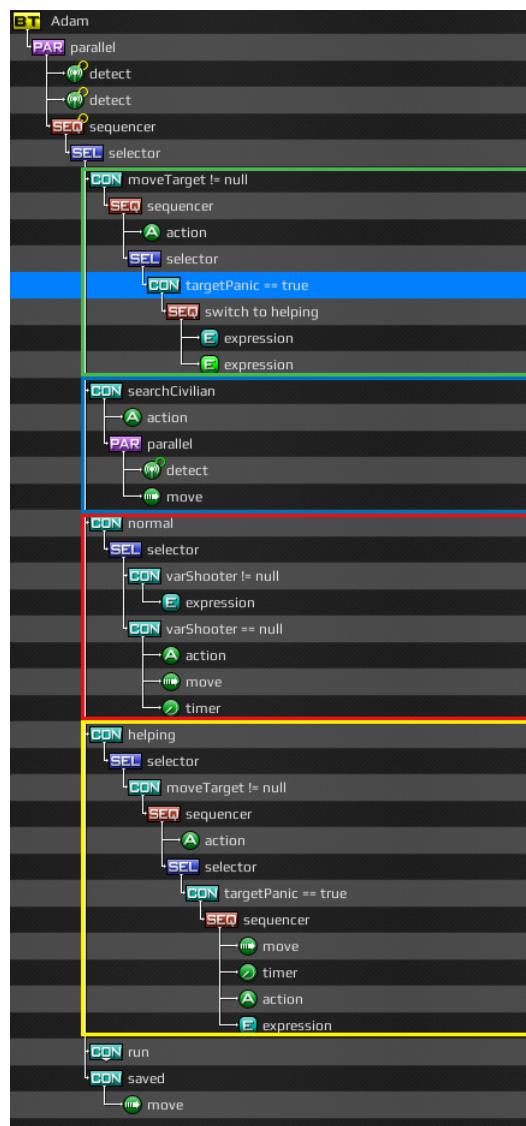


7.4.3 Adam

Le nom du *Behavior Tree* d'Adam se nomme **Adam**. Il est illustré à la figure 7.5 – Behavior Tree d'Adam.

- Deux détecteurs
 - Pour le forcené/shooter
 - Pour les civils
- Vert: déclenché à la vue d'un civil. Si ce dernier est en panique, Adam change d'état et va l'aider. Le bloc (A) action vérifie si le civil passé au détecteur est en état de panique. Dans un tel cas l'état et la vitesse sont modifiés.
- Bleu: recherche de civil: déclenché si Adam voit le forcené/shooter. Dans un tel cas, il continuera à se déplacer par les *Navigation Targets* de manière aléatoire, mais en courant. S'il voit un civil en panique, Adam change d'état et va l'aider. Le bloc (A) action vérifie si le civil passé au détecteur est en état de panique. Dans un tel cas l'état et la vitesse sont modifiés.
- Rouge: état normal.
- Jaune: en train d'aider: état déclenché si Adam voit un civil. Adam se déplace vers lui grâce au bloc *move*, reste à ses côtés durant cinq secondes grâce au bloc *timer* puis le bloc (A)action modifie l'état du civil qui passera de "paniqué" à "en fuite".
- Run et Saved: fonctionnent de la même manière que le civil

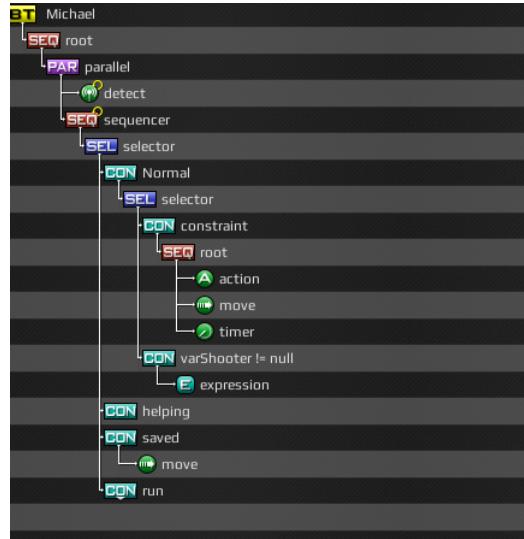
Figure 7.5 – Behavior Tree d'Adam



7.4.4 Michael

Le nom du *Behavior Tree* de Michael se nomme **Michael**. Il est illustré à la figure 7.6 – Behavior Tree de Michael.

Figure 7.6 – Behavior Tree de Michael



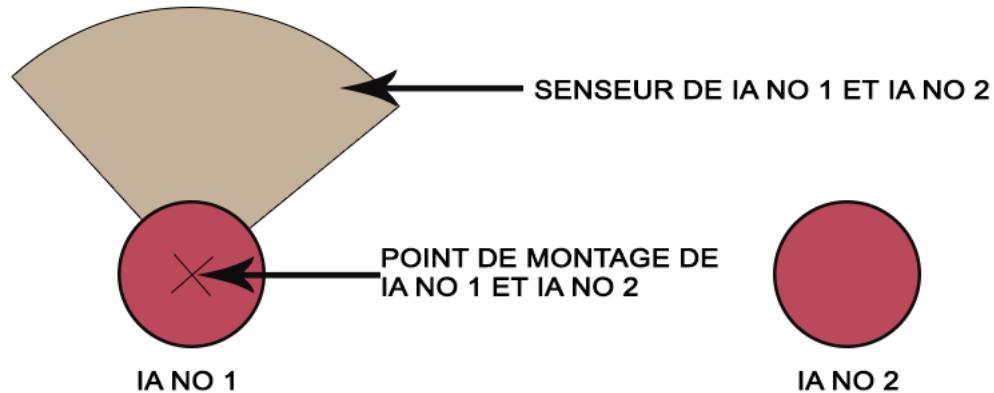
Le *Behavior Tree* de Michael est le plus léger du fait que son action est régie dans le code en C#. Sa phase normale fonctionne comme le civil. Au cas où le force/né/shooter passe dans sa ligne de vue, Michael modifie sa *Memory*, et passe en état "en train d'aider". Contrairement à Adam, qui lui utilise des *Custom Actions* pour aider les civils, Michael, lui, utilise uniquement son propre code C# (méthodes : OnTriggerEnter et OnTriggerEnter)

7.5 Code

7.5.1 Crédit d'entités et de senseurs pour Rain AI via C#

La création d'entités et de senseurs avec *Rain AI* a un grand défaut: le point de montage est statique. C'est-à-dire que si nous créons une IA sur un *gameObject* et que nous dupliquons notre *gameObject*, le point de montage de l'objet dupliqué restera celui de l'objet d'origine.

Figure 7.7 – Point de montage en cas de duplication



C'est pour cette raison que, pour la création d'aspects comme la création senseurs, nous passerons par le code. La création d'entités et de senseurs à la main via *Unity* serait trop longue et difficile à maintenir. Un bout de code tiré de *Civilian.cs* est donné en annexe en 13.8.

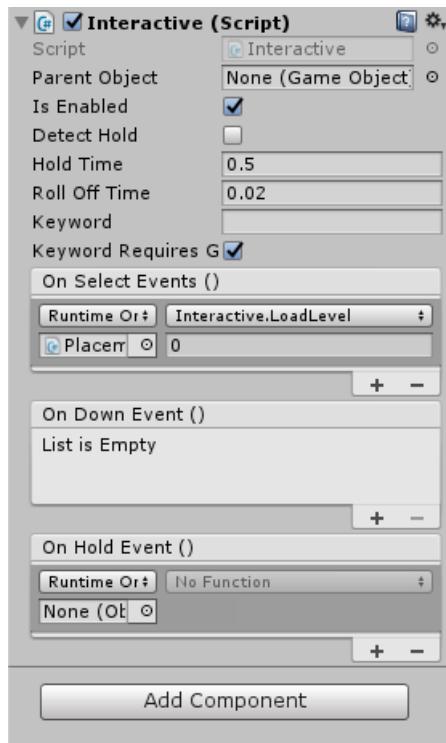
Pour la création, un bout de code tiré de *Civilian.cs* est disponible en annexe en 13.9. A noter que le champ *Body* de notre IA souffre du même problème: nous modifions donc cette valeur et nous lui attribuons le *GameObject* courant.

7.5.2 Changement de scène

Cette sous-section explique comment effectuer un changement de scène. Concrètement, dans la réalisation de ce projet, le changement de scène s'effectue quand nous sommes dans le menu de départ et nous voulons démarrer le jeu.

Dans un 1er temps il nous faut créer une classe qui hérite de *Interactive* (fournie dans le *Toolkit*). Ensuite, il est nécessaire de l'ajouter au *GameObject*, et dans *On Select Events* ajouter ce même objet, puis sélectionner la fonction désirée. Le code est donné en annexe (*InteractiveLoad.cs*).

Figure 7.8 – Configuration de notre bouton



7.6 Problèmes rencontrés

Ce chapitre traite des problèmes et difficultés qui ont été rencontrés durant la réalisation du projet. Il ne traite pas uniquement des problèmes rencontrés durant la phase de développement, mais durant l'ensemble du projet. Les questionnements apparus durant les phases d'apprentissage et de développement sont également abordés.

7.6.1 Nouvelle technologie

Un des problèmes principaux rencontré durant ce projet est le fait que la technologie est récente. Malgré le fait que le programme de formation en ligne de *Microsoft* soit relativement complet, on se heurte à des problèmes d'incompatibilités, comme le problème de *Rain AI* expliqué plus bas. De plus, les formations proposées par *Microsoft* sont pauvres d'un point de vue quantitatif. Le grand problème des tutoriels de *Microsoft* est qu'ils sont très peu génériques et très rigides. De plus, comme déjà expliqué dans ce rapport, les tutoriels de *Microsoft* partent du postulat que nous connaissons déjà bien *Unity* et se permet donc de faire des raccourcis, manquant de préciser des éléments de base.

7.6.2 Bonnes pratiques

Le fait d'utiliser un nouvel outil tel que *Unity* ne pose pas uniquement le problème de découverte d'un nouvel outil, mais également un problème de connaissances de bonnes pratiques qui ne sont pas enseignées dans les tutoriels de formations. Savoir manier un outil et faire le travail demandé est différent de savoir manier un outil, de faire le travail demandé et de le faire de manière correcte. A titre de comparaison, voilà une création (image 7.9) effectuée il y a quelques mois. Pour le commun des mortels (personne lambda en marketing/communication/graphisme), cette création plaît, elle est belle, attire l'œil et les informations nécessaires sont présentes, le travail est fait avec un outil maîtrisé. En revanche, lorsqu'il s'agit d'une personne expérimentée dans le domaine du/de la marketing/communication/graphisme, l'avis est différent.

Figure 7.9 – Flyer Festigeek



Voici un bref extrait des commentaire de cette personne.

- Les formes géométriques ont un impact visuel fort, mais c'est dommage tu n'as pas été jusqu'au bout
- Jeu de plan = richesse supplémentaire car c'est assez plat et linéaire pour le coup
- Sentiment d'unité absent, car tu regroupes tes formes entre elles (les triangles avec les triangles, les carrés avec les carrés)
- Laisser des vides ! Les vides sont aussi important que le reste, le vide laisse respirer la création
- La taille, et la situation dans l'espace = forme d'expression (un triangle au centre n'aura pas le même impact qu'un triangle dans le coin d'une feuille en bas à gauche, entouré de vide)

La question est donc: *J'ai réussi à faire ce que je voulais, mais l'ai-je fait correctement ?*

Pour un exemple lié directement au projet: Dans dans le script d'un *GameObject* je souhaite passer un autre *GameObject* en référence, deux solutions sont possibles :

- Faire un champ public dans le script et passer le *GameObject* directement depuis *Unity*
- Faire un champ privé et faire une recherche avec le nom de l'objet en utilisant la fonction *Find*

Les deux possibilités ont leurs avantages et inconvénients. La première a l'avantage d'être résistante au changement de nom de notre *GameObject*, en revanche elle nous force à avoir notre objet en public. La seconde en est l'opposé.

On peut encore citer l'utilisation ou non de la fonction *SendMessage* de *Unity* qui utilise la réflexion. Dans tous les cas nous avons l'embarras du choix, et seules l'expérience et une bonne réflexion peuvent se faire jugent des bonnes pratiques.

Solution choisie

Pour la 1ere question, la solution à privilégier dans notre cas est la première (solution utilisée dans les tutoriels). La seconde possibilité peut être envisagée quand les *GamesObjects* sont ajoutés au fil du déroulement du jeu, dans un cas où nous avons typiquement pas accès à l'objet dans *Unity*. Pour la question de l'utilisation de la réflexion, le débat reste ouvert.

7.6.3 Pathfinding sur AR et position de l'IA

Pathfinding

Le *Pathfinding* ou *Recherche de chemin* en français, est un problème récurant dans le monde de IA. Il consiste à trouver un chemin (le plus rapide) entre deux points, un point de départ, et un point d'arrivée.

Le fait de passer de la VR à l'AR pose un problème de *Pathfinding*. En effet, dans la version du projet en VR, nous avions un bâtiment en 3D, ce qui permet aux IAs de faire du *Pathfinding* avec *Rain AI*. Malheureusement cela n'est plus possible en AR, qui possède un univers dynamique, il nous est impossible de connaître la topologie des lieux à l'avance. Les IAs n'ont donc pas connaissance de l'environnement dans lequel elles évoluent, il n'est donc plus possible d'appliquer le *Pathfinding*. Un autre problème est le positionnement de départ des IAs, tireur comme civils. Dans le cas du projet en VR, les IA étaient positionnées dans le bâtiment en 3D, dans le cas de l'AR, nous avons juste connaissance de l'environnement qui nous entoure sur le moment. La problématique est la suivante: comment gérer le "spawn" des IA ? Comment dire que le tireur se trouve en salle B32 alors que nous sommes à la cafétéria ? Le fait que nous passons d'un univers statique en 3D à un univers réel et dynamique pose un réel problème à ce niveau-là.

Une solution envisageable est de modéliser le bâtiment 3D grâce à des murs invisibles et de superposer le bâtiment modélisé en 3D avec le bâtiment réel/dynamique. Pour effectuer cette superposition, nous avons deux choix possibles d'implémentation. La première consiste à donner un point de départ à l'utilisateur, proche de deux (voir trois) autres points dits "points de calibrage". Ces deux/trois points peuvent être, par exemple, des marques de couleur placées au sol. Leurs positions doivent être connues par le logiciel. L'utilisateur va regarder successivement les deux/trois points, ce qui va permettre une calibration et donc de superposer le modèle virtuel en 3D sur le bâtiment réel. La seconde solution est de définir un point de départ et une orientation de départ de manière arbitraire, et de superposer directement le modèle en 3D avec le bâtiment, en partant du principe que le policier/joueur se trouve sur le point de départ convenu.

A noter qu'il n'est pas nécessaire de faire en sorte que les murs invisibles correspondent aux murs intérieurs du bâtiment. En effet, il serait clairement envisageable de faire des murs voir même d'autres objets en 3D si le bâtiment d'entraînement est par exemple un grand hangar. Dans un cas également, il serait clairement possible de faire différents cas d'entraînement en variant le monde. Le fait d'avoir des murs en 3D visibles va poser un problème qui sera abordé au sous-chapitre suivant.

La principale problématique de cette solution et qu'il faut réaliser en 3D le bâtiment d'entraînement. Les problèmes liés à *Occulus Rift* (moins d'objets en 3D et plus de liberté) dont nous cherchions à nous débarrasser, reviennent en force.

Une autre solution apparemment envisageable est de faire un tour du bâtiment pour le cartographier². Cette solution permettrait un meilleur dynamisme et une infinité de possibilités d'implantation. Malheureusement cette solution semble impossible à réaliser car trop compliquée, considérant qu'avec cette méthode tout devient relatif : position de départ du policier, position de départ de la foule, *Pathfinding* vers la/les sortie, position de départ du tireur.

Solution choisie

La solution choisie a été de faire au plus simple, il n'y aura pas de gestion de recalibrage. Nous nous contenterons de réaliser/reprendre un bâtiment en 3D et définir un point de départ.

7.6.4 Traversée des hologrammes par le joueur

Un autre problème du passage de la VR à l'AR est que le joueur/policier n'est pas dans le même monde que l'environnement qui l'entoure; comment détecter que le joueur/policier ne triche pas en traversant des murs en hologrammes? Que faire s'il agit de la sorte? Il est possible de connaître la position du joueur/policier, et donc il est sûrement possible de connaître si oui ou non il entre en collision avec un mur en hologramme ou un autre objet en hologramme. Mettre fin à la partie si le joueur/policier entre en contact avec un mur ou un objet serait trop punitif. Une solution envisageable serait de définir une zone tampon dans laquelle le joueur/policier peut se trouver sans pour autant déclencher un arrêt du jeu. La figure 7.10 représente une implémentation possible. Dans le premier cas, si le joueur/policier n'est pas du tout en contact, il n'y a rien à gérer et tout se passe bien. Dans le second cas, le joueur/policier est dans l'intervalle(0-50)%, soit à moitié dans le mur. Dans ce cas nous pouvons informer le joueur/policier, par un texte en hologramme et/ou les bords de la vue qui clignotent en rouge (comme dans un *FPS* quand notre vie est basse), qu'il entre dans la zone tampon et qu'il doit faire attention et/ou en sortir. Dans le troisième cas, il est à plus de 50% dans le mur, le jeu se met en pause jusqu'à ce que le joueur soit à nouveau en position "OK".

Une autre solution envisageable est de gérer le *GameObject(MainCamera)* qui représente le joueur ou les *GameObjects* qui représentent nos civils et notre force/shotter, c'est à dire de leur ajouter un *Collider* et un *rigidBody*. Ainsi en cas de collision avec un mur ou un autre *GameObject* la caméra cessera de fonctionner malgré le fait que le joueur continue de bouger.

La première solution pose un problème de justesse, n'est-ce pas trop punitif de mettre fin à la partie si on traverse un mur ? La seconde solution pose le problème suivant: nous risquons au bout d'un moment d'être trop décalé et donc de heurter un obstacle réel

²https://developer.microsoft.com/en-us/windows/mixed-reality/holograms_230

Solution choisie

La seconde solution a été choisie car elle n'est pas punitive comme la première, de plus elle est aussi plus réaliste que la première.

État

Fonctionnel sur la *live compilation* de *Unity*, non-fonctionnel sur le *Hololens*.

Figure 7.10 – Collision mur-Hololens



7.6.5 Reconnaissance vocale

La reconnaissance vocale sur le *Hololens* est relativement avancée. En voici un exemple tiré du tutoriel de base du *Hololens*³ et donné en annexe sous le nom de *SpeechManager.cs*. Nous y implémentons deux commandes vocales, "Rest world" qui appelle par réflexion la méthode "OnRest" sur tous les objets (dans le cas du tutoriel, cela va remplacer les sphères à leur emplacement d'origine). La seconde commande fonctionne avec l'expression "Drop

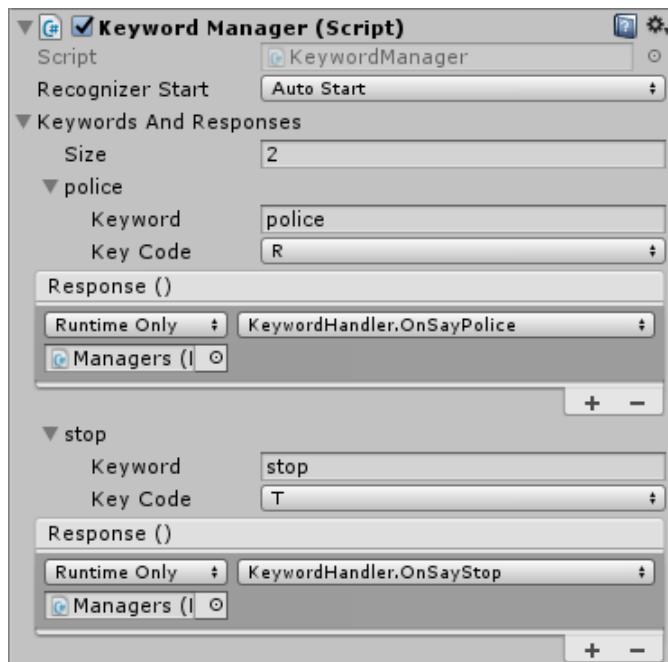
³ https://developer.microsoft.com/en-us/windows/holographic/holograms_101e

Sphere" qui rend sensible à la gravité la sphère ciblée par le pointeur. Nous pouvons constater que l'utilisation de commandes vocales est un problème: non seulement seul l'anglais est supporté, mais également que la reconnaissance est très stricte, et fonctionne au mot pour mot, et non pas avec une logique ou un contexte. Pour reprendre notre exemple de la sphère, dire "fall sphere" ne fonctionnera pas. Ces deux problèmes rendent donc les commandes vocales peu immersives. Un autre problème est la maintenance et la lisibilité du code, car tout mettre dans un fichier comme cela se fait rend la chose relativement "brut de décoffrage".

Solution

Pour faciliter l'implémentation de reconnaissance vocale, nous pouvons travailler avec le *HoloToolkit*, qui fournit un fichier C# du nom du *Keyword Manager* et qui permet facilement de lier un mot clé avec une fonction/méthode. Pour davantage de cohérence, nous créons un *Handler* afin de regrouper l'ensemble des fonctions appelées lors de commandes vocales (KeywordHandler.cs).

Figure 7.11 – Keyword Manager du Holotoolkit



7.6.6 Apprentissage

Un autre problème survenu durant le projet et la découverte de nouvelles technologies, a été de suivre un apprentissage non-progressif. Le fait d'aborder des technologies nouvelles et qui plus est récentes, a conduit à une première phase d'apprentissage chaotique, beaucoup de mini-projets faits en ayant suivi un tutoriel sur Internet finis à 80%. De plus, ces mini-projets étaient distincts entre eux et sans but/cadre réel.

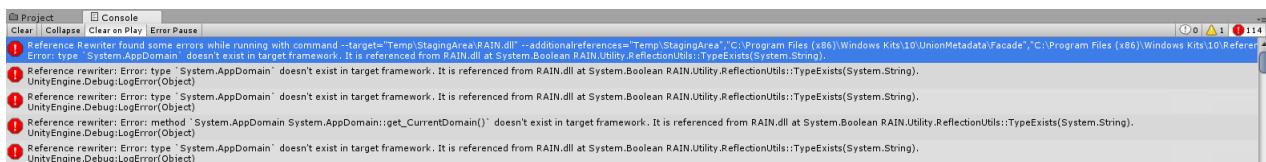
Solution

Cela a été résolu au bout de la cinquième semaine quand un premier scénario de test a été défini. Il s'agit du *Projet 1 : Rain AI*. Par la suite d'autres projets ont été ajoutés en définissant clairement les objectifs de chacun.

7.6.7 RainAI et .Net

Le grand problème avec *Rain AI* est qu'il n'est pas compatible avec .Net. Lors de la compilation du projet sur *Unity* nous avons les erreurs suivantes :

Figure 7.12 – Erreurs de compilation avec .Net



Solution

La résolution de ce problème fut relativement ardue dû au fait que personne n'avait implémenté *Rain AI* sur un projet *Unity* avec *Hololens* et donc *Windows 10*. En revanche, cela a déjà été fait sur *Windows 8.1*⁴. Dans le dernier message de ce sujet un membre explique comment faire pour compiler un projet *Unity* qui utilise *Rain AI* pour *Windows 8.1* et *Windows 10*. Pour passer de .Net à IL2CPP, il suffit de changer le *Scripting backend* de .Net à IL2CPP dans *Player Setting*.

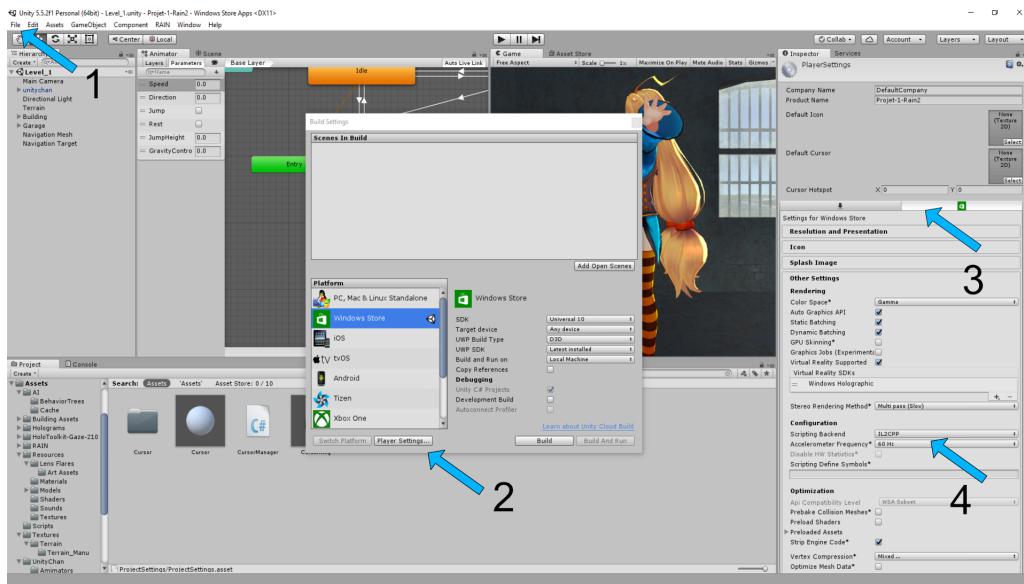
1. Aller dans file puis "build settings"
 2. Player Settings
 3. Sélectionner l'onglet Windows (carré vert)
 4. Passer "Scripting Backend" de .Net à "**IL2CPP**"

De plus amples informations sont disponibles sur le site de *Unity*⁵.

⁴ <http://legacy.rivaltheory.com/forums/topic/rain-2-1-7-2-windows-store-universal-8-1-fail/>

⁵<https://docs.unity3d.com/Manual/IL2CPP-BuildingProject.html>

Figure 7.13 – Passer de .Net à IL2CPP

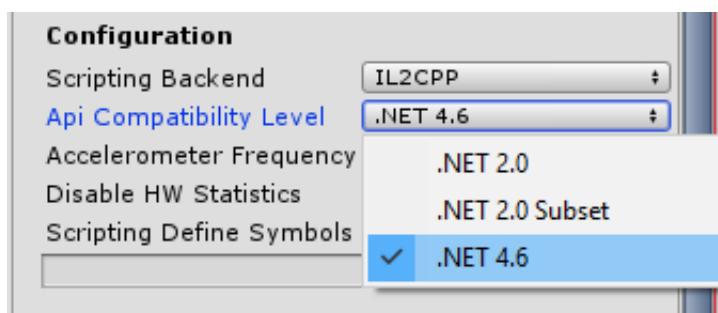


7.6.8 Holotool kit de Microsoft

Micosoft offre un *PlugIn* avec un set de code d'exemple et de *Prefab* permettant de démarrer un projet avec une base de code solide ainsi qu'un set d'outils de base déjà fonctionnel comme par exemple un menu de navigation, des *Slidebars* etc. Des erreurs sont générées lorsque nous *buildons* le projet sous *Unity* avec *IL2CPP*. Comme mentionné précédemment la solution est alors de travailler avec *IL2CPP* pour *Rain AI*.

Solution

La première étape à faire est d'utiliser la version 5.6.1f1 de *Unity* qui permet d'utiliser *IL2CPP* avec une compatibilité *.NET 4.6* et de modifier *API Compatibility Level* dans les *Player Settings*. A partir de là il nous restera deux erreurs qui sont les suivantes :

Figure 7.14 – Utiliser la *API Compatibility Level* avec *.NET 4.6*

```

1 Assets\\HoloToolkit\\SpatialMapping\\Scripts\\RemoteMapping\\MeshSaver.cs(162,53): error CS1061:←
  'StorageFile' does not contain a definition for 'OpenStreamForReadAsync' and no extension ←
  method 'OpenStreamForReadAsync' accepting a first argument of type 'StorageFile' could be ←
  found (are you missing a using directive or an assembly reference?)←
2 Assets\\HoloToolkit\\SpatialMapping\\Scripts\\RemoteMapping\\MeshSaver.cs(189,53): error CS1061:←
  'StorageFile' does not contain a definition for 'OpenStreamForWriteAsync' and no extension ←
  method 'OpenStreamForWriteAsync' accepting a first argument of type 'StorageFile' could be ←
  found (are you missing a using directive or an assembly reference?)←
3 Error building Player because scripts had compiler errors`←

```

Pour résoudre ces erreurs, il faut se rendre dans le fichier générant l'erreur et supprimer ce qui se trouve dans le *if* d'instructions pré-processeur (instructions pré-processeur comprises) et ne conserver que ce qui se trouve dans le *else*.

```

1 private static Stream OpenFileForRead(string folderName, string fileName)
2 {
3     Stream stream = null;
4
5     #if !UNITY_EDITOR && UNITY_METRO
6         Task<Task> task = Task<Task>.Factory.StartNew(
7             async () =>
8             {
9                 StorageFolder folder = await StorageFolder.GetFolderFromPathAsync(folderName);
10                StorageFile file = await folder.GetFileAsync(fileName);
11                stream = await file.OpenStreamForReadAsync();
12            });
13            task.Wait();
14            task.Result.Wait();
15        #else
16            stream = new FileStream(Path.Combine(folderName, fileName), FileMode.Open, FileAccess.Read);
17        #endif
18        return stream;
19    }

```

Devenir

```

1 private static Stream OpenFileForRead(string folderName, string fileName)
2 {
3     Stream stream = null;
4     stream = new FileStream(Path.Combine(folderName, fileName), FileMode.Open, FileAccess.Read);
5     return stream;
6 }

```

```

1 private static Stream OpenFileForWrite(string folderName, string fileName)
2 {
3     Stream stream = null;
4
5     #if !UNITY_EDITOR && UNITY_METRO
6         Task<Task> task = Task<Task>.Factory.StartNew(
7             async () =>
8             {

```

```

9             StorageFolder folder = await StorageFolder.GetFolderFromPathAsync(←
10                folderName);
11             StorageFile file = await folder.CreateFileAsync(fileName, ←
12                CreationCollisionOption.ReplaceExisting);
13             stream = await file.OpenStreamForWriteAsync();
14         });
15     task.Wait();
16     task.Result.Wait();
17 #else
18     stream = new FileStream(Path.Combine(folderName, fileName), FileMode.Create, FileAccess.←
19       Write);
#endif
20     return stream;
21 }
```

Est modifié comme suit :

```

1 private static Stream OpenFileForWrite(string folderName, string fileName)
2 {
3     Stream stream = null;
4     stream = new FileStream(Path.Combine(folderName, fileName), FileMode.Create, FileAccess.←
5       Write);
6     return stream;
}
```

Malheureusement le passage de .Net2.0 à .Net4.6 a rendu *Rain AI* inopérant. La solution à ce problème se trouve à la section suivante.

7.6.9 RainAI sur Hololens et .Net 4.6

Pour utiliser le *Holotoolkit* il faut construire notre solution sur *Unity* avec .Net 4.6 Le problème est que lors du déploiement sur le Hololens, tout ce qui est généré par *Rain AI* n'est pas chargé, alors que lors de la *live compilation* sur *Unity*, tout se passe à merveille. La source du problème est que *Rain AI* ne tourne pas avec un *API Compatibility Level* avec .Net 4.6, il fonctionne uniquement avec la configuration suivante :

- *scripting backend* : IL2CPP
- *API Compatibility level* : .Net 2.0

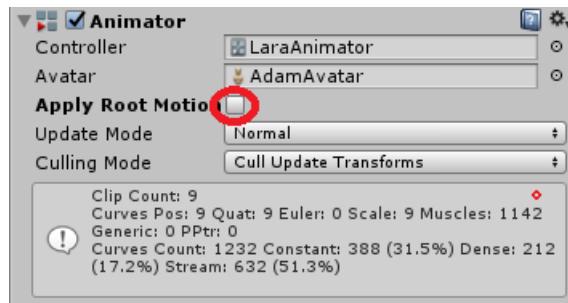
Solution

Supprimer tous les fichiers inutiles qui ne fonctionnent qu'avec .Net 4.6.

7.6.10 Non-unicité des ressources

Le fait de devoir aller chercher des ressources sur *l'Asset Store* a posé un problème inhérent au fait que ces ressources ne sont pas faites par les mêmes personnes. En conséquence, leurs choix impactent directement le rendu final de leur travail et qui pour deux ressources similaires, impliquent une implémentation différente. A titre d'exemple nous pouvons citer la vitesse d'animation entre les civils et le forcené/shooter. Dans le cas du forcené/shooter une valeur de 1 pour la vitesse d'animation correspond à une vitesse de marche, alors que pour les civils une vitesse de 1 représente une vitesse de course voir même de sprint. Un autre problème du même acabit: l'animation du forcené/shooter gère aussi son déplacement: au moment où sa vitesse est supérieure à 0, le forcené/shooter se déplacera (en ligne droite) et cela même s'il n'a pas d'IA qui lui est attribuée. En revanche, pour les civils, si une vitesse supérieure à 0 leur est attribuée, l'animation de marche ou de course sera jouée mais ils ne se déplaceront pas. La raison à cela se trouve dans le *Component Animator* qui gère les animations de nos objets: dans le cas de notre forcené/shooter, *Apply Root Motion* était activé, ce qui fait que les animations ont une influence sur le déplacement ainsi que sur la physique de l'objet. Bien que ce problème soit facile à résoudre, il est tout de même à relever.

Figure 7.15 – Component d'animation



7.6.11 Collision (Detection) Avoidance

La présence d'un grand nombre de personnes (IA) se déplaçant de leur plein gré au sein d'un endroit pose un problème classique dans le domaine des IA: celui de l'évitement de collision entre IA. Il est important de faire la différence entre le problème du *Pathfinding* et le problème de détection de collision. L'algorithme de détection de collision peut trouver un chemin entre deux points, mais il éprouvera des difficultés au moment où il devra passer par des objets de forme concave.

Un *PlugIn* intéressant pour ce problème se nomme *Polarith*. Malheureusement celui-ci cohabite mal avec *Rain AI*.

Solution

Une solution viable aurait été d'utiliser *Polarith*, malheureusement, il n'a pas été possible de le faire cohabiter avec *Rain AI*. Une solution secondaire a donc été mise en place. Il s'agit du script *PlayerAvoidance.cs* qui est utilisé comme *Custom Action* par les *Behavior Tree*. Bien que cette solution soit moins esthétique sur le rendu du jeu, elle évite tout de même des inter-blocages.

Tests

Nous diviserons la phase de tests en deux parties: la première sur la *live compilation* de *Unity* et la seconde directement sur le *Hololens*. Bien que la première méthode de test ne soit pas en corrélation avec le produit final, elle est intéressante car elle nous permet de nous déplacer facilement et rapidement dans la totalité du bâtiment. De plus, le fait de passer par une première batterie de tests via la *live compilation* nous permet de constater si des exceptions sont levées et lesquelles.

8.1 Live compilation

8.1.1 Test sans forcené/shooter

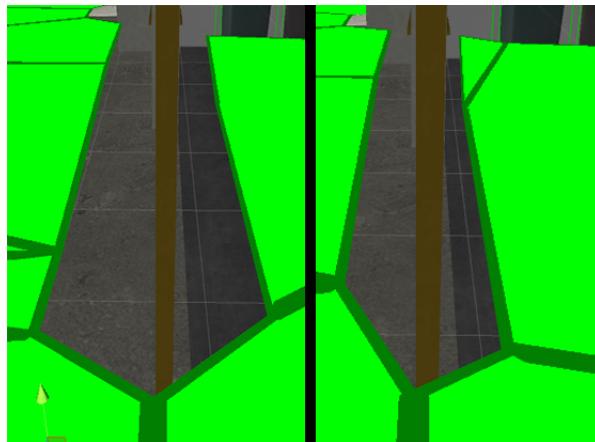
Un premier test sera effectué avec une trentaine de civils (Adam et Michael compris) afin de vérifier qu'il n'y ait pas de problème au niveau des déplacements des civils (pas de blocage contre un mur, waypoints inaccessibles). Le jeu a donc été exécuté environ cinq minutes, seul, sans l'intervention du policier.

Problèmes/bugs trouvés

La porte d'entrée a posé un léger problème car, selon l'angle d'entrée du civil, il arrivait qu'il prenait une trajectoire trop tangente, ce qui faisait que le civil se retrouvait bloqué, la tête dans l'angle de la porte.

Pour résoudre ce problème, il a été nécessaire d'augmenter le *Walkable radius* de la *Navigation Mesh*, augmentant ainsi la distance minimale entre le civil et le mur.

Figure 8.1 – Différence entre un Walkable radius de 0.4 et de 0.2



8.1.2 Test avec forcené/shooter

Un second test a été réalisé avec le forcené/shooter, mais toujours sans intervention du joueur/policier. La caméra a volontairement été mise à l'écart pour éviter que le forcené/shooter ne parte se cacher. Le but ici était de vérifier que les civils passaient bien en état de panique ou en état de fuite. Il servait également à vérifier si Adam et Michael faisaient bien leur travail "héroïque".

Problèmes/bugs trouvés

Petite faute chez Adam lors de la vision du forcené/shooter :

`state = "searchCivilian" au lieu de state = "SearchCivilian"`. Problème dû au fait qu'Adam n'a pas été totalement retesté au passage à l'enum.

8.1.3 Ramener le forcené/shooter dans la safe Zone

L'idée dans ce test est de vérifier si le forcené/shooter réagit correctement à la vue du policier et à ses actions. Dans un premier temps nous testerons si le forcené/shooter prend correctement la fuite, puis nous essayerons de le ramener vers la *safe Zone*, et ce afin de constater s'il prend fin correctement.

Problèmes/bugs trouvés

Si nous faisions un clic sur le forcené/shooter alors qu'il était de dos, il passait en état arrêté, mais pour une obscure raison la *constraint* : `state == "Arrested"` échouait. Le moyen pour éviter ce problème a été de repasser dans la ligne de vue du forcené/shooter après l'avoir arrêté ce qui avait pour conséquence de "relancer" le *Behavior tree*, de réévaluer la *constraint* : `state ==`

"Arrested", qui cette fois-ci n'échouait pas.

Pour résoudre ce problème, il a été nécessaire d'ajouter un *Sequencer* avant le *Selector* des différents *states*, de lui mettre un *Repeat : forever* et de modifier la *constraint : state == "Arrested"* en lui mettant *Repeat : until failure*.

8.1.4 Ramener le forcené/shooter et les civils dans la safe Zone

Après avoir ramené uniquement le forcené/shooter, il va de soit qu'il faut ramener/faire sortir l'ensemble des IA, et tester si le jeu prend bien fin lorsque tous les civils et le forcené/shooter sont dans la *safe Zone*.

Problèmes/bugs trouvés

La raison pour laquelle le jeu ne prend pas fin dans le cas où nous désactivons certains civils est la suivante.

Quand nous testons les conditions de fin, il faut avoir tous les enfants du *gameObject* dans la *safe Zone* et le forcené/shooter doit être *Caught*. Pour avoir le nombre d'enfants du *gameObject* contenant les civils, nous utilisons la fonction donnée ci-dessous.

```
1    civilList.GetComponent<Transform>().childCount
```

Ainsi nous avons le nombre de civils dans le jeu, mais pas le nombre de civils **ACTIFS**. Le problème est qu'il faut ramener tous les civils dans la *safe Zone* pour finir le jeu. Un civil désactivé sera quand même compté avec cette fonction, mais il sera impossible de le voir, d'interagir avec lui et donc de le sauver. Pour résoudre ce problème nous allons compter uniquement les civils actifs.

Listing 8.1 – ChildCountActive

```
1  public int ChildCountActive(Transform t)
2  {
3      int k = 0;
4      foreach (Transform c in t)
5      {
6          if (c.gameObject.activeSelf)
7              k++;
8      }
9      return k;
10 }
```

Ainsi nous jouissons d'une meilleure évolutivité.

8.2 Sur Hololens

8.2.1 Navigation

Le premier test réalisé sur *Hololens* a été le test de la navigation du menu de départ. Tous les boutons ont été testés pour voir s'il n'y avait pas de problème de liens (mauvais menu qui s'ouvre), et si le bouton "fermer" fermait correctement le menu en question. Au passage nous avons vérifié que les textes d'informations étaient suffisamment lisibles.

Problèmes/bugs trouvés

Hormis les fautes d'orthographe, aucun problème n'a été trouvé.

8.2.2 Commande gestuelle

Dans ce test nous vérifions que la commandes gestuelle *tap movement* est bien fonctionnelle sur le *Hololens*. La difficulté principale pour tester la commandes gestuelle sur le *Hololens* est d'atteindre un civil et/ou le forceéné/shooter pour interagir avec eux car nous ne disposons pas d'un bâtiment où nous sommes libres de nos mouvements.

8.2.3 Commande Vocale

Dans ce test nous ne vérifions que les commandes vocales. Pour les même raisons que la commande gestuelle, les commandes vocales furent difficiles à tester.

Problèmes/bugs trouvés

Commande *Reset* mal orthographié.

8.2.4 Fin du jeu

Le but de ce test était de vérifier si le jeu prenait bien fin lorsque l'ensemble des IAs actives se trouvaient dans la *safe Zone*. Malheureusement cela n'a pas pu être testé car nous ne disposions pas d'un emplacement suffisamment grand pour tester le tout du début à la fin.

Conclusion

9.1 Etat du projet

9.2 Utilité du Hololens

Malgré l'avancée technologique dont fait preuve le *Hololens*, il reste tout de même décevant. Décevant car l'immersion est moindre, ce qui, pour un projet comme celui-ci, développé dans un cadre de travail de Bachelor, est un point crucial. Cette mauvaise immersion est due à quatre éléments, le premier est le poids du casque qui au bout d'un moment pèse sur les oreilles et le nez. Le second point et non des moindres est la zone dans laquelle les hologrammes sont visibles. Cette zone est presque "tunellique": un regard sur la droite sans tourner la tête et nous n'avons plus d'hologramme dans notre champ de vision. Une application plus satisfaisante de ce casque serait dans des formations plus théoriques que pratiques comme par exemple l'observation du corps humain (muscles, assertion, nerfs, etc.). Un autre problème est que pour le moment il n'existe pas de version française, ce qui pose un problème pour la reconnaissance vocale. Le dernier est qu'en tant qu'utilisateur, nous ne sommes pas sur le même "plan" que le monde dans lequel nous sommes plongés: les hologrammes ne peuvent pas interagir physiquement avec nous. Si nous prenons le cas d'un jeu vidéo classique, lorsque nous entrons en contact avec une foule, on pourrait imaginer que la caméra (vision du joueur) se met à trembler pour simuler la collision entre le joueur et son environnement pour donner une immersion plus forte, une telle chose avec le *Hololens* est impossible. Pour terminer un dernier point important faisant défaut au *Hololens* est les nausées qu'il procure au bout de 30 minutes d'utilisation, les yeux se fatiguent en effet très vite. Les différents points faibles du *Hololens* laissent susciter une question: est-ce vraiment un bon moyen de formation ?

9.3 Paradoxe des jeux pédagogiques et de la simulation

De base, le jeu tout comme la simulation comporte sa propre finalité et il n'a pas de conséquence directe sur son activité externe. Le jeu, tout comme la simulation transpose le joueur dans un univers parallèle coupé de sa réalité. Le fait d'être détaché de la réalité au profit d'un univers factice est-il vraiment bénéfique à l'apprentissage ? Un avantage certain entre l'apprentissage par simulation et l'apprentissage brute (sur le terrain) est que les actions du sujet/joueur n'ont pas de conséquences sur sa vie : une erreur dans la simulation n'aura

aucun impact sur sa vie, ce qui n'est pas le cas dans une situation réelle où une erreur peut avoir de lourdes conséquences. Mais ce cadre de sûreté n'est-il également pas néfaste à la formation ? La simulation sera-t-elle suffisante pour que le sujet/joueur soit capable de reproduire et répliquer les connaissances et expériences acquises durant la formation sur simulateur ? L'entraînement, qui est une forme d'apprentissage en soit, doit être là pour préparer l'être humain à une situation pour laquelle il n'est pas forcément préparé. Nous pouvons prendre comme exemple une personne pratiquant la boxe de manière régulière en salle, sans pression... Le jour où cette personne aura vraiment besoin de ce savoir-faire acquis durant l'entraînement (exemple : une agression dans la rue) sera-t-elle capable d'utiliser ce savoir faire ? Nous pourrions prendre exemple sur les ouvrages de Georges Héber qui prônait la méthode naturelle d'apprentissage et dont la doctrine était "*être fort pour être utile*" qui favorisait la formation en terrain accidenté ou en milieu naturel plutôt qu'en terrain aménagé.

9.4 Réponse aux questions de l'introduction

"Mais où en sommes-nous aujourd'hui ?"

Nous pouvons clairement affirmer que la technologie en est à ses premiers pas, et a encore de grands défauts. Pour le projet avec *Occulus Rift*, nous pouvons citer son déplacement peu réaliste et son obligation d'être rattaché à un ordinateur via un câble. Pour ce qui est du *Hololens* la réponse à la question a été traitée dans la section *Utilité du Hololens*, où l'on dénonce les problèmes comme le faible champ de vision, la capacité de mouvement très limitée ou encore les nausées au bout d'un moment d'utilisation. Mais malgré ces défauts, il ne faut pas enterrer cette technologie

"Actuellement quelles sont les possibilités d'une telle technologie dans le cadre de la formation ?"

Tout ce qui a été dit précédemment met clairement à mal la réalité augmentée / réalité virtuelle dans le cadre de la formation pratique. En revanche la question reste ouverte pour ce qui est d'une formation plus théorique, comme l'astronomie ou l'anatomie. Mais à la vue des expériences de ces derniers mois, le *Hololens* trouverait nettement mieux son utilité dans des formations plus théoriques. Imaginons une seule seconde avoir un hologramme d'un corps humain que nous pourrions séparer en morceau pour observer les différents muscles, les os et les organes.

"L'immersion dans la simulation est-elle satisfaisante ? Est-il possible de réaliser une simulation proche de la réalité ?"

Considérant ce qui a été mentionné précédemment, nous pouvons clairement en attendre plus du *Hololens*. Mais ce n'est pas le seul problème, l'IA est également un problème au niveau du réalisme de la simulation, car chaque personne est différente et a une manière de penser

qui lui est propre. Retranscrire cela dans un jeu est une tâche quasiment impossible.

9.5 Les plus

Ce chapitre relate les points positifs relevés durant le développement du projet.

9.5.1 Peu de grands problèmes

Dans la globalité il n'y a pas eu de gros problème de type casse-tête à résoudre. Les seuls problèmes étaient souvent mineurs et rapidement résolus.

9.5.2 Une communauté active et à l'écoute

Entre le *Discord* et le forum de *Unity*, il est relativement facile de trouver une solution. Les gens sur le *Discord* de *Unity* sont vraiment à l'écoute et n'hésite pas à aider les gens.

9.5.3 Unity - Game design

Bien que s'appeler *Game designer* est un peu hautain et prétentieux, le fait de sortir des sentier battu et d'endosser l'étiquette de *Game designer* fût instructive et intéressante car elle à fait toucher du bout des doigts un univers lié à l'informatique mais avec des pendants tous autres cela principalement dû au fait que le développement un jeu est fondamentalement différent que celui d'un logiciel classique.

Un grand intérêt que l'on peut relever et que pour le développement/création d'un jeu, la créativité est de vigueur, ce qui n'est pas forcément le cas pour un logiciel classique.

9.6 Erreurs commises - Les moins

Ce chapitre traite des erreurs commises durant le projet, et il s'agira ici de *débriefer* dessus afin de ne pas les reproduire dans un futur projet similaire.

9.6.1 Apprentissage

La première erreur faite dans ce projet est due au fort enthousiasme qu'a suscité ce dernier. Le fait de tout vouloir découvrir a fait passer d'un sujet à l'autre sans avoir maîtrisé le premier. On suit un tutoriel, puis durant ce tutoriel un autre sujet nous interpelle, et on ne finit pas le premier tutoriel. Le fait de définir des scénarios-maquettes avec un objectif clair et précis a permis de recentrer la phase d'apprentissage.

9.6.2 Premier jeu et jeu != logiciel classique

Le fait de développer un jeu plutôt qu'un logiciel "classique" impose la phase d'analyse supplémentaire qu'on pourrait appeler le *game design*. De bons conseils (surtout pour la création d'un premier jeu) sont donnés dans cette vidéo explicative d'*ExtraCredits*¹ :

- Définir la portée de notre jeux et ce que l'on est capable de faire *focus on the scope*.
 - Prendre en compte le temps que l'on a
 - Le nombre de personnes dans l'équipe
 - Nos capacités: nos point forts et nos points faibles
- Partir du principe que le projet a pour objectif l'apprentissage et non pas la réalisation du projet de notre vie.
- Se concentrer sur les choses simples mais les faire bien et les mener à bout
- Ne pas être trop attaché aux idées que l'on veut mettre dans notre jeu
- Et surtout, surtout, être capable de définir un projet minimum

Dans la figure en 9.1 - Liste des différents types de jeux dans l'ordre de difficulté, ExtraCredits nous donne la liste des différents types de jeux dans l'ordre de difficulté. Nous pouvons constater que les FPS sont en 7ème position. Le projet étant relativement proche de ce type de jeu, la phase d'analyse aurait dû être d'autant plus poussée.

¹<https://www.youtube.com/user/ExtraCreditz/featured>

Figure 9.1 – Liste des différents types de jeux dans l'ordre de difficulté

Game genres in order of difficulty to produce a minimum viable product, from simplest to most difficult:

- | | |
|-------------------------------|----------------------|
| 1. Racing Game | 7. FPS |
| 2. Top Down Shooter | 8. JRPG |
| 3. 2D Platformer | 9. Fighting Game |
| 4. Color Matching Puzzle Game | 10. Action Adventure |
| 5. 2D Puzzle Platformer | 11. Western RPG |
| 6. 3D Platformer | 12. RTS |

9.6.3 Incapacité à définir le projet minimum et de s'y tenir

La plus grande erreur a été de ne pas avoir été capable de définir un projet minimal et de s'y tenir. L'envie de toujours vouloir ajouter des éléments, sans même que le minimum vital ne soit atteint a été sans doute le plus gros problème (exemple : faire des recherches sur le tir alors que les interactions avec les IA ne sont pas finies, testées et fonctionnelles). *ExtraCredits* résume parfaitement la chose avec le jeu le plus connu de tous les temps.

Quel est le minimum vital pour faire Mario ?

- Bowser ? Non
- Champignons ? Non
- Tortues ? Non
- Tunnels ? Non

Tous cela est juste inutile, le simple fait de pouvoir sauter, se déplacer, tomber dans les trous est amplement suffisant. La tendance de vouloir ajouter des éléments qui en fin de compte sont futiles ne mène qu'à se perdre et à dériver pour au final arriver avec un produit dont l'essentiel n'est pas pleinement fonctionnel et où l'expérience utilisateur ne sera pas satisfaisante. Pour reprendre notre exemple de *Mario*, si nous ajoutons un *Bowser* alors que la base du jeu n'est

pas pleinement achevée, notre utilisateur en sera perturbé. Il se posera alors des questions comme "Quel est ce monstre mi-dragon mi-tortue ?" ou "Quel est le but ?" et son expérience ainsi que son plaisir en seront amoindris. Alors que si nous faisons qu'un seul niveau mais dans lequel les collisions sont fonctionnelles, le système de saut également, l'utilisateur prendra alors bien plus de plaisir.

9.7 Travaux futurs

Bien que l'utilisation du *Hololens* pour ce projet soit finalement un échec à cause des différents problèmes de ce dernier, il ne faut cependant pas enterrer le projet de formation de policier avec de la réalité augmentée. D'autres pistes sont envisageables avec d'autres technologies. Il est impossible de démentir le fait que travailler avec une technologie qui est actuellement de l'ordre de la science-fiction, à l'image de *Total Recall* ou encore *l'Animus* de *Assassin's creed* ne serait pas jouissif.

9.8 Mots de la fin

Les premières facultés qui se forment et se perfectionnent en nous sont les sens. Ce sont les premières qu'il faudrait cultiver: ce sont les seules qu'on oublie, ou celles qu'on néglige le plus... mais, au lieu d'occuper éternellement mon élève à des gambades, je le mènerais au pied d'un rocher; là, je lui montrerais quelle attitude il faut prendre, comment il faut porter le corps et la tête, quel mouvement il faut faire, de quelle manière il faut poser, tantôt le pied, tantôt la main, pour suivre légèrement les sentiers escarpés, raboteux et rudes, et s'élançer de pointe en pointe tant en montant qu'en descendant. J'en ferais l'émule d'un chevreuil plutôt qu'un danseur de l'Opéra

J.J.Rousseau.Émile ou de l'éducation

Planification

Ce chapitre explique plus en détail et défend les tâches du diagramme de gantt présent ci-dessous.

Installation de l'environnement et planification

Les deux premiers jours du travail de bachelor seront consacrés à l'installation de l'environnement et des outils nécessaires pour mener à bien la réalisation du projet. Ces outils sont :

- Unity 3D
- Visual Studio 2015
- HoloLens Emulator
- Vuforia

Une explication plus détaillée se trouve à l'adresse suivante : https://developer.microsoft.com/en-us/windows/holographic/install_the_tools

Découverte & Apprentissage

Compte tenu du fait que ce projet est un saut dans l'inconnu, le premier mois sera uniquement consacré à l'apprentissage d'Unity et de sa "sur-couche" pour le Hololens. Une partie du temps sera aussi allouée à la découverte du *VRKinectAmokShooter* qui est le projet ayant été déjà réalisé au sein de l'école.

Réalisation de la maquette et points critiques

Pour le développement du projet, la décision suivante a été prise : le projet sera découpé en deux grandes parties. La première prendra fin le 19 juin et sera consacrée à la réalisation d'une maquette sans scénario, et à partir cette date, il sera possible de tout faire dans le programme de formation, et nous devrions être au clair sur tous les sujets du projet (pas de problèmes laissés ouverts). La seconde partie sera uniquement consacrée à l'implémentassions de notre

scénario dans notre projet. Cette section traite de la première partie. Il est essentiel qu'à cette date tous les points critiques soient résolus et qu'il n'existe plus de problème laissé en suspend.

Détection de l'environnement

La première tâche sera de comprendre comment implémenter la détection de l'environnement perçu par le Hololens pour, par la suite, y intégrer des personnages/objets 2D/3D

Affichage d'un/des personnages (objet 2D/3D)

Cette partie sera consacrée à l'ajout de la couche 2D/3D. Le but au terme de cette étape est d'être capable d'ajouter facilement des objets virtuels en prenant compte de l'environnement.

Modélisation du bâtiment

Malgré le fait que nous sommes passés d'un projet de réalité virtuelle à un projet de réalité augmentée afin de pas être soumis aux limitations techniques de la VR, nous nous heurtons à un autre problème. Le fait de passer d'un univers fictif réalisé en 3D à un univers réel "sur-couché" par de la 3D, nous impose une nouvelle difficulté. Les intelligences artificielles n'auront pas connaissance du monde dans lequel elles évoluent, ce qui pose un réel problème de "pathfinding". Une solution envisageable est de modéliser en 3D le bâtiment (mur invisible) et de superposer cette modélisation avec le bâtiment réel. Ainsi les IA auront connaissance de la topographie des lieux et le pathfinding pourra fonctionner.

Actions du policier

Actions possibles par le policier, sont :

- interagir avec les civils via le *tap movement* et/ou la voix (mots clés)

- interagir avec le shooter via le *tap movement* et/ou la voix (mot clés)

Intégration de l'IA du tireur

L'IA du tireur est déjà présent dans le projet **VRKinectAmokShooter** et il faudra faire la migration du projet de base vers le projet actuel.

Mouvement de foule

Le mouvement de foule étant déjà présent dans le projet **VRKinectAmokShooter**, il faudra faire la migration du projet de base vers le projet actuel.

Définition du scénario

La dernière étape avant le travail à temps plein est la définition du scénario. Cette étape doit prendre en compte les possibilités actuelles du projet.

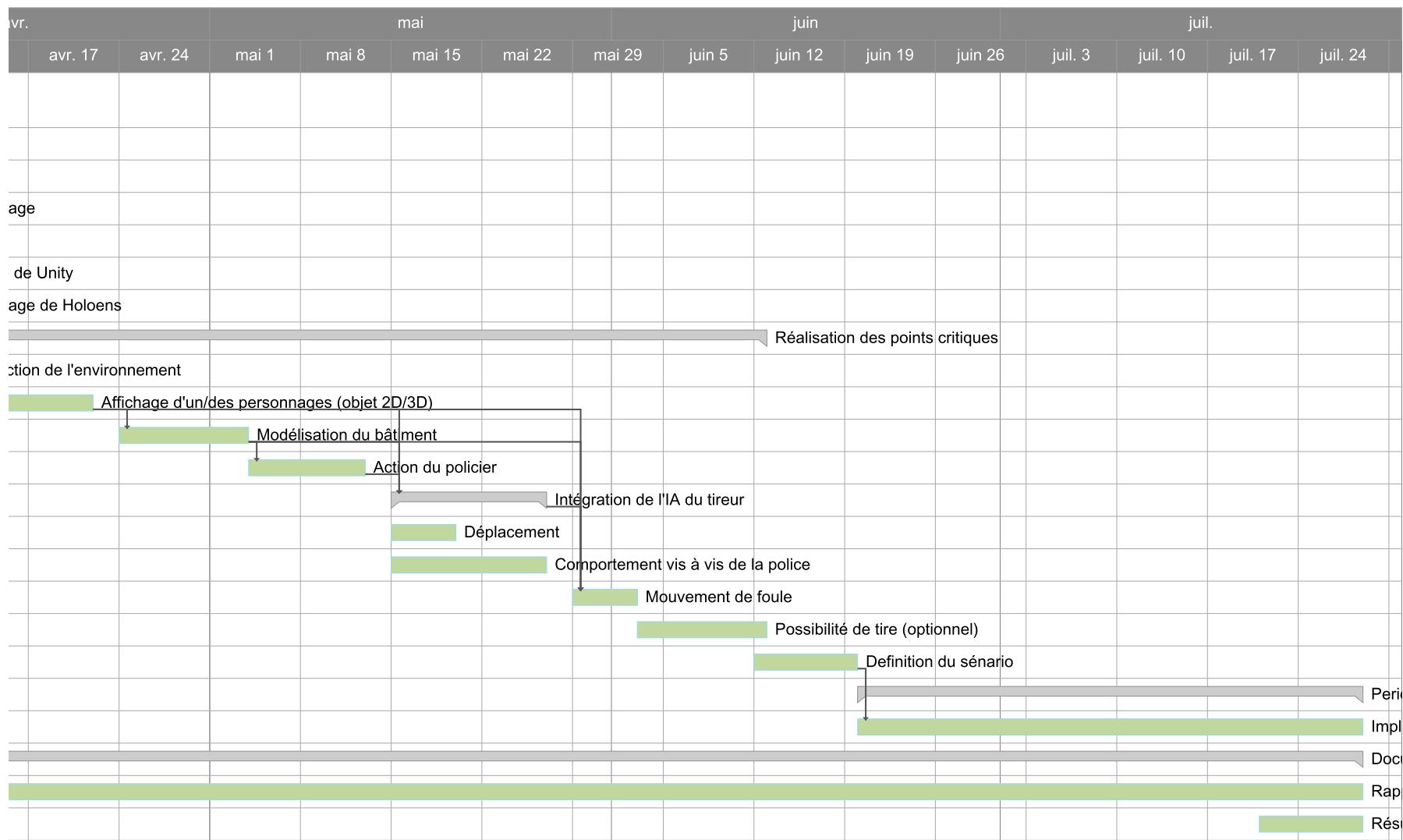
Implémentations du scénario

La dernière étape de réalisation du travail de bachelor est l'implémentassions du scénario défini au préalable.

Gantt TB



Nom de la tâche	La date										
	janv.	févr. 6	févr. 13	févr. 20	févr. 27	mars 6	mars 13	mars 20	mars 27	avr. 3	avr. 10
1 Installation de l'environnement et Planification						Installation de l'environnement et Planification					
2 Installation de l'environnement					Installation de l'environnement						
3 Planification					Planification						
4 Découverte & Apprentissage									Découverte & Apprentissage		
5 Découverte du travail VRKinectAmokShooter					Découverte du travail VRKinectAmokShooter						
6 Découverte & Apprentissage de Unity									Découverte & Apprentissage		
7 Découverte & Apprentissage de Holoens										Découverte & Apprentissage	
8 Réalisation des points critiques											
9 Détection de l'environnement									Détection de l'environnement		
10 Affichage d'un/des personnages (objet 2D/3D)											Affichage d'un/des personnages (objet 2D/3D)
11 Modélisation du bâtiment											
12 Action du policier											
13 Intégration de l'IA du tireur											
14 Déplacement											
15 Comportement vis à vis de la police											
16 Mouvement de foule											
17 Possibilité de tire (optionnel)											
18 Définition du scénario											
19 Période à plein temps											
20 Implémentation du scénario											
21 Documentation											
22 Rapport											
23 Résumé du TB											



Webographie

11.1 Microsoft

MICROSOFT - HOLOGRAMS 101E

https://developer.microsoft.com/en-us/windows/mixed-reality/holograms_101e

MICROSOFT - HOLOGRAMS 101

https://developer.microsoft.com/en-us/windows/mixed-reality/holograms_101

MICROSOFT - HOLOGRAMS 210

https://developer.microsoft.com/en-us/windows/holographic/holograms_210

MICROSOFT - GESTURES

<https://developer.microsoft.com/en-us/windows/mixed-reality/gestures>

11.2 Youtube

YOUTUBE - UE4: ADVANCED MATERIALS (EP.1 USING CRAZYBUMP)

<https://www.youtube.com/watch?v=jdq-hVCc0So>

YOUTUBE - RAIN v2.1 QUICK START

<https://www.youtube.com/watch?v=YuaBBCL5PSs>

YOUTUBE - UNITY 5 TUTORIAL - ANIMATION CONTROL

<https://www.youtube.com/watch?v=wdOk5QXYC6Y>

YOUTUBE - UNITY 4.0 - MECANIM ANIMATION TUTORIAL

<https://www.youtube.com/watch?v=Xx21y9eJq1U>

YOUTUBE - EXTRACREDITZ

<https://www.youtube.com/user/ExtraCreditz/featured>

GET STARTED | POLARITH AI UNITY TUTORIAL [BASICS]

<https://www.youtube.com/watch?v=FYuSEJFiJMc>

UNITY VS UNREAL ENGINE | GRAPHICS COMPARISON

<https://www.youtube.com/watch?v=Hr5IOEQI7eg>

11.3 Autre

COMMON INTERMEDIATE LANGUAGE

https://fr.wikipedia.org/wiki/Common_Intermediate_Language

IL2CPP

<https://docs.unity3d.com/Manual/IL2CPP.html>

HOLOTOOLKIT

<https://github.com/Microsoft/HoloToolkit-Unity/>

WHAT COULD POSSIBLY GO WRONG

<http://www.what-could-possibly-go-wrong.com/il2cpp/>

INFINITESQUARE

<http://blogs.infinitesquare.com/posts/mobile/unity-il2cpp-presentation-et-principe-de-fonctionnement>

UNITY 3D

<https://docs.unity3d.com/Manual/index.html>

RAIN AI

<http://legacy.rivaltheory.com/rain/>

HOW TO ADD VOICE COMMANDS TO YOUR HOLOLENS APP

<https://hololens.reality.news/how-to/holotoolkit-add-voice-commands-your-hololens-app-0175284/>

POLARITH

<https://forum.unity3d.com/threads/released-polarith-ai-free-pro-for-movement.466846>

SE PLONGER DANS UN JEU POUR MIEUX APPRENDRE? THÉORIE, CONCEPTION ET EXPÉRIMENTATION AUTOEUR DES JEUX VIDÉO PÉDAGOGIQUES

<http://tecfa.unige.ch/tecfa/maltt/memoire/SutterWidmer10.pdf>

COMPARAISON ENTRE UNITY 5 ET UNREAL ENGINE 4

<https://www.supinfo.com/articles/single/2139-comparaison-unity-5-unreal-engine-4>

Liste des symboles et abréviations utilisés

Ce chapitre bien que bref liste les différent symboles utilisé dans ce document, ainsi que les différentes abréviations utilisées.

12.1 Symboles

- Oeil  dans le diagrammes : représente le fait de voir un certain type d'entité
- Point d'exclamation "!" dans les diagrammes : représente l'action du policier sur l'entité (action peut être gestuelle et/ou vocale)

12.2 Abréviations

- IA (ou AI en anglais) : Intelligence Artificielle
- VR : Virtual reality
- AR : Augmented reality

Références

IMAGE DU HOOLENS

http://www.etr.fr/devices_images/980-hololens_face.png

IMAGE DE RAINAI

<http://legacy.rivaltheory.com/rain/features/>

<http://legacy.rivaltheory.com/wp-content/uploads/navmesh.jpg>

<http://legacy.rivaltheory.com/wp-content/uploads/waypoints1.jpg>

<http://legacy.rivaltheory.com/wp-content/uploads/behaviortree-e1375387925601.jpg>

<http://legacy.rivaltheory.com/wp-content/uploads/Perception.jpg>

FONCTIONNEMENT DE IL2CPP

<https://blogs.unity3d.com/2015/05/06/an-introduction-to-ilcpp-internals/>

<https://blogs.unity3d.com/wp-content/uploads/2015/04/il2cpp-toolchain-smaller.png>

GESTES POSSIBLES AVEC LE HOOLENS

<https://abhijitjana.files.wordpress.com/2016/05/image51.png?w=608&h=412>

https://kbdevstorage1.blob.core.windows.net/asset-blobs/12462_en_3

HOOLENS EMULATOR

<https://az835927.vo.msecnd.net/sites/mixed-reality/Resources/images/Emulator.PNG>

Table des illustrations

4.1	Oculus DK2	10
4.2	Hololens	11
4.3	Navmesh	12
4.4	Route	12
4.5	arbre de décision	13
4.6	Perception	14
4.7	Fonctionnement de IL2CPP	15
4.8	Air tap	17
4.9	Bloom	17
5.1	Navigation Mesh générée	22
5.2	Différents type de waypoint	23
5.3	Configuration du Pathfinding	24
5.4	configuration de la Navigation Mesh	25
5.5	Fonctionnement du Navigation Network	26
5.6	Menu Perception de Rain + Visual Sensors	28
5.7	Entity	29
5.8	Behavior Tree avec détecteur	30
5.9	Schéma Détecteur - Senseur - Entité	30
5.10	Sense visuel	31
5.11	Memory	32
5.12	Collision fonctionnelle	34
5.13	Colliders de l'Astroman	35

5.14 Collider du Suitman	36
6.1 Scénario 2	39
6.2 Diagramme de décision du civil	41
6.3 Diagramme de décision du forcené/shooter	42
6.4 Scène utilisée à partir du projet 3	43
6.5 Scène utilisée à partir du projet 3	44
6.6 Diagramme utilisé pour le civil pour le projet 3.0	46
6.7 Diagramme utilisé pour le forcené/shooter pour le projet 3.0	48
6.8 Menu de départ	49
6.9 Diagramme de Michael	51
6.10 Diagramme de Michael	52
6.11 Diagramme du civil pour le projet 3.2	53
6.12 Safe Zone	54
7.1 Diagramme de classe des Customs Actions	57
7.2 Diagramme de classes des IA	62
7.3 Behavior Tree du civil	64
7.4 Behavior Tree du forcené/shooter	65
7.5 Behavior Tree d'Adam	67
7.6 Behavior Tree de Michael	68
7.7 Point de montage en cas de duplication	69
7.8 Configuration de notre bouton	70
7.9 Flyer Festigeek	71
7.10 Collision mur-Hololens	75
7.11 Keyword Manager du Holotoolkit	76
7.12 Erreurs de compilation avec .Net	77

7.13 Passer de .Net à IL2CPP	78
7.14 Utiliser la <i>API Compatibility Level</i> avec .NET 4.6	78
7.15 Component d'animation	81
8.1 Différence entre un Walkable radius de 0.4 et de 0.2	84
9.1 Liste des différents types de jeux dans l'ordre de difficulté	91
14.1 Exemple de civil	122
14.2 Exemple d'Adam	123
14.3 Exemple de Michael	123
14.4 Exemple de Shooter	124
14.5 Installation de Visual Studio	126
14.6 Build Settings sous Unity	127
14.7 Build Settings sous Unity	128
14.8 Build Settings sous Visual Studio	128

Listings

8.1 ChildCountActive	85
14.1 GazeGestureManager.cs	115
14.2 SphereCommands.cs	116
14.3 Memory.cs	116
14.4 Civilian_One_AI.cs	117
14.5 SpeechManager.cs	118
14.6 KeywordHandler.cs	119
14.7 InteractiveLoad.cs	120
14.8 Création de sensor	120
14.9 Création d'une Entity de Rain	120

Annexe

14.1 Code

Listing 14.1 – GazeGestureManager.cs

```

1  using UnityEngine;
2  using UnityEngine.VR.WSA.Input;
3
4  public class GazeGestureManager : MonoBehaviour
5  {
6      public static GazeGestureManager Instance { get; private set; }
7
8      // Represents the hologram that is currently being gazed at.
9      public GameObject FocusedObject { get; private set; }
10
11     GestureRecognizer recognizer;
12
13     // Use this for initialization
14     void Start()
15     {
16         Instance = this;
17
18         // Set up a GestureRecognizer to detect Select gestures.
19         recognizer = new GestureRecognizer();
20         recognizer.TappedEvent += (source, tapCount, ray) =>
21         {
22             // Send an OnSelect message to the focused object and its ancestors.
23             if (FocusedObject != null)
24             {
25                 FocusedObject.SendMessageUpwards("OnSelect");
26             }
27         };
28         recognizer.StartCapturingGestures();
29     }
30
31     // Update is called once per frame
32     void Update()
33     {
34         // Figure out which hologram is focused this frame.
35         GameObject oldFocusObject = FocusedObject;
36
37         // Do a raycast into the world based on the user's
38         // head position and orientation.
39         var headPosition = Camera.main.transform.position;
40         var gazeDirection = Camera.main.transform.forward;
41
42         RaycastHit hitInfo;

```

```

43     if (Physics.Raycast(headPosition, gazeDirection, out hitInfo))
44     {
45         // If the raycast hit a hologram, use that as the focused object.
46         FocusedObject = hitInfo.collider.gameObject;
47     }
48     else
49     {
50         // If the raycast did not hit a hologram, clear the focused object.
51         FocusedObject = null;
52     }
53
54     // If the focused object changed this frame,
55     // start detecting fresh gestures again.
56     if (FocusedObject != oldFocusObject)
57     {
58         recognizer.CancelGestures();
59         recognizer.StartCapturingGestures();
60     }
61 }
62 }
```

Listing 14.2 – SphereCommands.cs

```

1  using UnityEngine;
2
3  public class SphereCommands : MonoBehaviour
4  {
5      // Called by GazeGestureManager when the user performs a Select gesture
6      void OnSelect()
7      {
8          // If the sphere has no Rigidbody component, add one to enable physics.
9          if (!this.GetComponent<Rigidbody>())
10         {
11             var rigidbody = this.gameObject.AddComponent<Rigidbody>();
12             rigidbody.collisionDetectionMode = CollisionDetectionMode.Continuous;
13         }
14     }
15 }
```

Listing 14.3 – Memory.cs

```

1  void Update()
2  {
3      if (tRig != null && shooterScript.isArrested())
4      {
5          tRig.AI.WorkingMemory.SetItem<bool>("varShooterArrested", true);
6      }
7      if (tRig != null && !shooterScript.isArrested())
8      {
9          tRig.AI.WorkingMemory.SetItem<bool>("varShooterArrested", false);
10     }
11     if (tRig != null)
12     {
13         if (tRig.AI.WorkingMemory.GetItem<int>("speed") >= 5)
14         {
15             anim.SetBool("run", true);
```

```

16         anim.SetBool("walk", false);
17     }
18     else
19     {
20         anim.SetBool("run", false);
21         anim.SetBool("walk", true);
22     }
23 }
24 }
```

Listing 14.4 – Civilian_One_AI.cs

```

1 using RAIN.Core;
2 using RAIN.Serialization;
3 using RAIN.Navigation;
4 using RAIN.Navigation.Targets;
5 using UnityEngine;
6
7
8 public class Civilian_One_AI : MonoBehaviour
9 {
10    AIRig tRig = null ;
11    Animator anim = null;
12    int speed;
13    public GameObject shooter = null;
14    private Shooter shooterScript;
15
16    void onSaved()
17    {
18        tRig.AI.WorkingMemory.SetItem<string>("state", "saved");
19    }
20    void Start()
21    {
22        anim = GetComponent<Animator>();
23        tRig = gameObject.GetComponentInChildren<AIRig>();
24        shooterScript = shooter.GetComponent<Shooter>();
25    }
26
27    void Update()
28    {
29        if (tRig != null && shooterScript.isArrested())
30        {
31            tRig.AI.WorkingMemory.SetItem<bool>("varShooterArrested", true);
32        }
33        if (tRig != null && !shooterScript.isArrested())
34        {
35            tRig.AI.WorkingMemory.SetItem<bool>("varShooterArrested", false);
36        }
37
38        //Si notre vitesse est ésupprieur ou égale à 5 nous utiliserons l'animation de cours, ←
39        //dans le cas contraire nous utiliserons celle de la marche
40        if (tRig != null)
41        {
42            if (tRig.AI.WorkingMemory.GetItem<int>("speed") >= 5)
43            {
44                anim.SetBool("run", true);
45                anim.SetBool("walk", false);
46            }
47        }
48    }
49 }
```

```

46         else
47         {
48             anim.SetBool("run", false);
49             anim.SetBool("walk", true);
50         }
51     }
52 }
53
54 //action se selection
55 void OnSelect()
56 {
57     if (tRig != null && (tRig.AI.WorkingMemory.GetItem<string>("state") == "panique" || tRig→
58         .AI.WorkingMemory.GetItem<string>("state") == "normal"))
59     {
60         OnGoEscapePoint();
61     }
62 }
63 void OnRun()
64 {
65     Destroy(gameObject);
66     Destroy(this);
67 }
68 //Paraliser/Normal -> Escape Point
69 void OnGoEscapePoint()
70 {
71     tRig.AI.WorkingMemory.SetItem<string>("state", "run");
72 }
73 void OnEndGame()
74 {
75     print("On end game");
76 }
77 void OnReset()
78 {
79     print("OnReset");
80 }
81 }
```

Listing 14.5 – SpeechManager.cs

```

1  using System.Collections.Generic;
2  using System.Linq;
3  using UnityEngine;
4  using UnityEngine.Windows.Speech;
5
6  public class SpeechManager : MonoBehaviour
7  {
8      KeywordRecognizer keywordRecognizer = null;
9      Dictionary<string, System.Action> keywords = new Dictionary<string, System.Action>();
10
11     // Use this for initialization
12     void Start()
13     {
14         keywords.Add("Reset world", () =>
15         {
16             // Call the OnReset method on every descendant object.
17             this.BroadcastMessage("OnReset");
18         });
19 }
```

```

19     keywords.Add("Drop Sphere", () =>
20     {
21         var focusObject = GazeGestureManager.Instance.FocusedObject;
22         if (focusObject != null)
23         {
24             // Call the OnDrop method on just the focused object.
25             focusObject.SendMessage("OnDrop");
26         }
27     });
28 });
29
30 // Tell the KeywordRecognizer about our keywords.
31 keywordRecognizer = new KeywordRecognizer(keywords.Keys.ToArray());
32
33 // Register a callback for the KeywordRecognizer and start recognizing!
34 keywordRecognizer.OnPhraseRecognized += KeywordRecognizer_OnPhraseRecognized;
35 keywordRecognizer.Start();
36 }
37
38 private void KeywordRecognizer_OnPhraseRecognized(PhraseRecognizedEventArgs args)
39 {
40     System.Action keywordAction;
41     if (keywords.TryGetValue(args.text, out keywordAction))
42     {
43         keywordAction.Invoke();
44     }
45 }
46 }
```

Listing 14.6 – KeywordHandler.cs

```

1 using System.Collections;
2 using System.Collections.Generic;
3 using HoloToolkit.Unity.InputModule;
4 using UnityEngine;
5
6 public class KeywordHandler : MonoBehaviour
7 {
8     private GazeManager gm;
9     // Use this for initialization
10    void Start ()
11    {
12        gm = gameObject.GetComponentInChildren<GazeManager>();
13    }
14    public void OnSayPolice()
15    {
16        if (gm.HitObject != null && gm.HitObject.GetComponent<Civilian>() != null)
17        {
18            //gm.HitObject.GetComponent<Civilian>().OnPolice();
19            gm.HitObject.GetComponent<Civilian>().SendMessage("OnPolice");
20        }
21    }
22    public void OnSayStop()
23    {
24        if (gm.HitObject != null && gm.HitObject.GetComponent<ShopShooter>() != null)
25        {
26            //gm.HitObject.GetComponent<Civilian>().OnPolice();
27            gm.HitObject.GetComponent<ShopShooter>().SendMessage("OnStop");
28    }}
```

```

28     }
29 }
30 }
```

Listing 14.7 – InteractiveLoad.cs

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  public class InteractiveLoad : HoloToolkit.Examples.InteractiveElements.Interactive
7  {
8      public void LoadLevel(int level)
9      {
10         SceneManager.LoadScene(level);
11     }
12 }
```

Listing 14.8 – Crédit de sensor

```

1 tRig.AI.Senses.AddSensor(CreateVisualSensor(true, "eyes", 120, new Vector3(0,1.6f ,0), true, ←
2   10, Color.green));
3
4
4  protected VisualSensor CreateVisualSensor(bool IsActive, string SensorName, int HorizontalAngle←
5   , Vector3 PositionOffset,
6   bool RequireLineOfSight, float rang = 10f, Color color = default(Color))
7  {
8    VisualSensor s = new VisualSensor
9    {
10      IsActive = IsActive,
11      SensorName = SensorName,
12      MountPoint = gameObject.transform,
13      HorizontalAngle = HorizontalAngle,
14      PositionOffset = PositionOffset,
15      RequireLineOfSight = RequireLineOfSight,
16      LineOfSightMask = 1024,
17      Range = rang,
18      SensorColor = color
19    };
20    return s;
21  }
```

Listing 14.9 – Crédit d'une Entity de Rain

```

1 GameObject entity = new GameObject("Entity");
2 entity.tag = "aCivil";
3 entity.transform.parent = gameObject.transform;
4 entity.AddComponent<EntityRig>();
5
6 tRig.AI.Body = gameObject;
```

14.2 Manuel d'utilisation

14.2.1 Menu

Lors du lancement du jeu, nous arrivons directement dans le menu. A partir de là le jeu est lancé en effectuant un *Tap movement* sur le bouton "Lancer". Dans le cas où vous ne connaissez pas le jeu, rendez vous dans "Informations" en effectuant un *Tap movement*. Un nouveau menu apparaîtra à droite du menu précédent, avec quatre boutons :

- But : Explique le but du scénario et les différents points pour le mener à terme.
- Commandes : Explique les différentes commandes qu'il est possible de faire et leurs impacts respectifs
- Elements : Présente les différents protagonistes du jeu (Civils classiques, Shooter, Adam, Michael) ainsi que l'explication de ce qu'est la *safe Zone*
- Fermer : Ferme le menu d'informations.

14.2.2 Jeu

Deux commandes sont possibles dans le jeu

- Gestuelle : via le *tap movement*.
- Vocale : via des mots clés donnés dans le chapitre des commandes vocales.

Gestuelle

La seule commande utile en jeu est le *tap movement*. Ses effets sont les suivants :

- Sur les civils, Adam et Michael : Les fait passer à un état de fuite vers la *safe zone*
- Shooter : Le fait passer à un état arrêté : il se mettra à suivre l'utilisateur du *Hololens*.

Vocale

Trois commandes sont implémentées pour le moment quand les mots clés suivants sont prononcés

- "Police" : Si un civil (y compris Adam et Micheal) est visé en disant ce mot clé, la cible part vers la *Safe Zone* en courant.

- "Stop" : Si le shooter est visé en disant ce mot clé, il passera à l'état arrêté et suivra le porteur du *Hololens*
- "Reset" : Arrête le jeu et nous renvoie au menu de départ

Gagner la partie

Pour gagner la partie, il vous faudra ramener l'ensemble des civils et le shooter à l'intérieur des *safe Zone*.

Protagoniste

Durant le jeu, vous allez croiser plusieurs protagonistes que voici :

Figure 14.1 – Exemple de civil



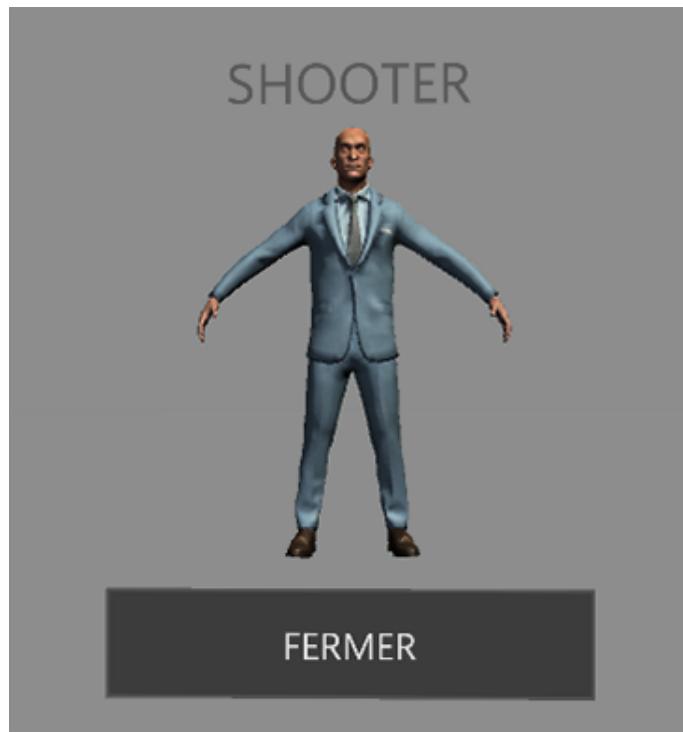
Figure 14.2 – Exemple d'Adam



Figure 14.3 – Exemple de Michael



Figure 14.4 – Exemple de Shooter



14.3 Manuel d'installation

14.3.1 Installation des logiciels

Le déploiement de l'application se fait en deux temps. Dans un premier temps sous *Unity* et dans un second sous *Visual Studio*. Les indispensables pour un déploiement de A à Z sont les suivants :

- Unity
 - .NET Scripting Backend
 - IL2CPP Scripting Backend
- Visual Studio
 - Avec Windows 10 SDK
 - Unity ToolKit
- Hololens Emulator (Optionnel)

Les logiciels sont disponibles ici :

<https://store.unity.com/download?ref=personal> pour *Unity*

<https://www.visualstudio.com/fr/downloads/> pour *Visual Studio*

Dans un premier temps, lisez et acceptez les conditions générales d'utilisation. Ensuite, installez *Unity AVEC .NET Scripting Backend et IL2CPP Scripting Backend*. Il est impératif que ces composants soient installés, sinon, lors du *build* avec *Unity*, des erreurs seront levées.

Passez ensuite à l'installation de *Visual Studio*. Lisez attentivement à nouveau les conditions générales d'utilisation et acceptez-les, puis lancez l'installation. Il est impératif d'avoir le SDK de *Windows 10* installé ainsi que le *ToolKit* de développement pour *Unity*.

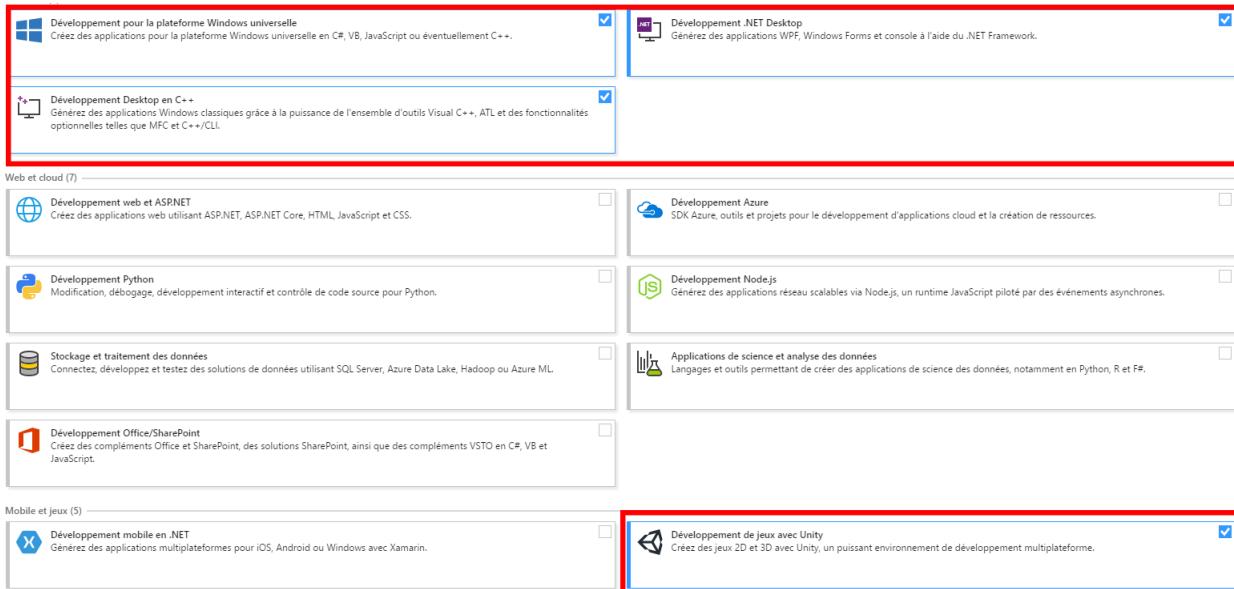
14.3.2 Installation de l'émulateur

Le *Hololens* coûte relativement cher (chose qui pourrait contraindre les développeurs à laisser tomber le *Hololens*), l'utilisation d'un émulateur rend donc possible le développement accessible à tous.

Notez une chose : avant de tenter de déployer l'application sur l'émulateur il faut une quantité conséquente de RAM disponible. Si vous n'avez pas *Windows 10 Pro*, l'émulateur HoloLens ne fonctionnera pas car il a besoin qu'Hyper-V soit activé.

L'émulateur de *Micosoft* se trouve ici :

Figure 14.5 – Installation de Visual Studio



- <http://go.microsoft.com/fwlink/?LinkId=823018>

Suivez l'installation en laissant les paramètres par défaut et assurez-vous que lors de la dernière étape que *Microsoft Hololens Emulator* et *Microsoft Hololens App Templates* soient bien cochés.

14.3.3 Déploiement de l'application

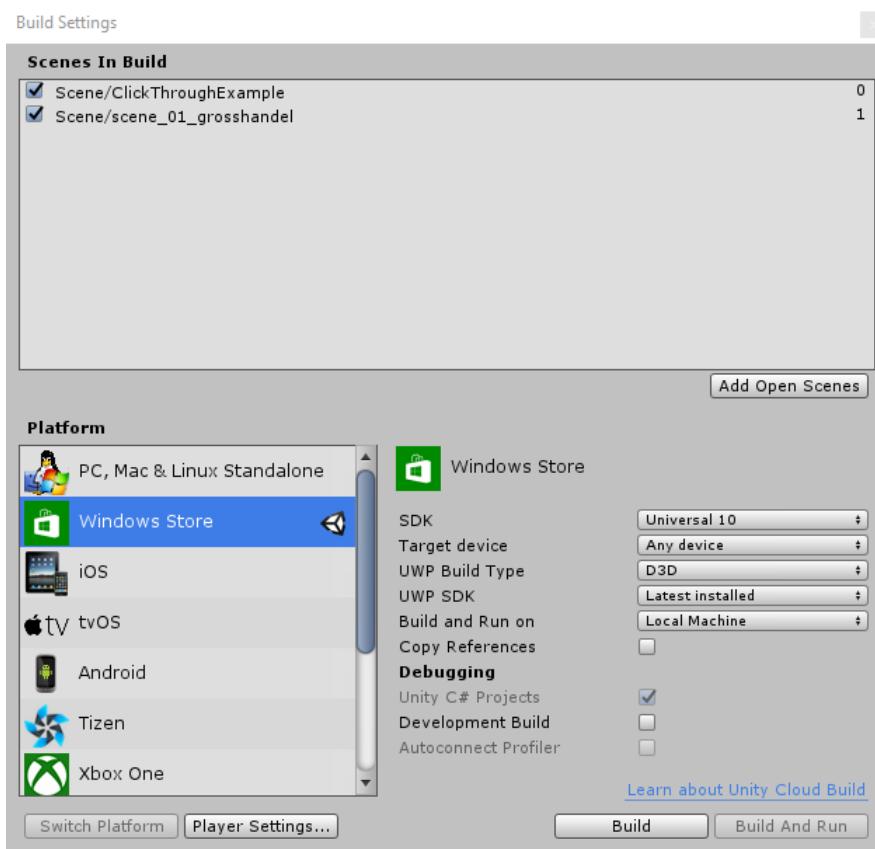
Le déploiement de l'application commence sur *Unity*. Une fois le projet ouvert sur *Unity*, allez dans :

- *File -> Build Settings*
- Assurez-vous que *Windows Store* soit bien sélectionné (Logo *Unity* en regard)
- Définissez le SDK à *Universal 10*
- *UWP Build type* doit être à *D3D*
- *UWP SDK* doit être à *Last installed*
- *Unity C Project* doit être coché

Avant de lancer le build, allez dans *Player Setting* et assurez-vous que :

- *Virtual Reality Supported* soit bien coché

Figure 14.6 – Build Settings sous Unity



- *Virtual Reality SDK* soit bien *Windows Holographic*
- *Scripting backend* soit bien *IL2CPP*
- *API compatibility level* soit bien *.Net 2.0*

Pour terminer, cliquez sur *build* et créez un dossier au nom de *App* dans lequel le *Build* va se faire.

Une fois le *build* terminé, allez dans le dossier *App* et cliquez sur *.sln*. Normalement Visual Studio se lance automatiquement.

Pour un déploiement sur *Hololens*, branchez le *Hololens* en *USB*. Pour la configuration de déploiement, choisissez *Release*, *x86* et *Device*.

Figure 14.7 – Build Settings sous Unity



Figure 14.8 – Build Settings sous Visual Studio



Journal de travail

Vendredi 24 février 2017

Tâche	Description	temps
Rendez-vous	Rendez vous avec M.Juergen afin de décider ce qui serait intéressant de faire	40mn
Installation	Installation de Unity, Visual Studio, Hololens Emulator	4h
Rapport	Téléchargement et prise en main du rapport avec Latex	2h

Mardi 28 février 2017

Tâche	Description	temps
Rapport et planification	réalisation du chapitre "planification"	3h

Mercredi 1er mars 2017

Tâche	Description	temps
Apprentissage Unity	Configuration d'un premier projet avec Unity et Hololens.	6h

Vendredi 3 mars 2017

Tâche	Description	temps
Apprentissage Unity	Apprentissage de Unity et Hololens.	6h

Mercredi 8 mars 2017

Tâche	Description	temps
Apprentissage Unity	Apprentissage de Unity et Hololens.	4h

Vendredi 10 mars 2017

Tâche	Description	temps
Rendez-vous	Rendez vous avec M. Juergen afin de parler de la planification et de relever les problèmes et questionnements de début de projet	40mn
Apprentissage Unity	Apprentissage des animations sur Unity.	8h

Mardi 14 mars 2017

Tâche	Description	temps
Apprentissage Unity	Apprentissage de Unity, Rain IA and pathfinding.	6h

Mercredi 15 mars 2017

Tâche	Description	temps
Apprentissage Unity	Apprentissage de Unity, Rain IA and pathfinding.	4h

Mercredi 22 mars 2017

Tâche	Description	temps
Apprentissage Unity et Hololens	Apprentissage de Unity, Rain IA and pathfinding et animation. Retour sur les tutoriels de formation de Microsoft sur le Hololens	6h

Vendredi 25 mars 2017

Tâche	Description	temps
Rendez-vous	Rendez vous avec M. Juergen afin de définir un 1er scénario de demo	30mn
Projet	Début du Projet 1 : Rain Ai	8h

Mardi 29 mars 2017

Tâche	Description	temps
Documentation	Avancement des chapitres d'apprentissage et des problèmes rencontrés	4h

Mercredi 30 mars 2017

Tâche	Description	temps
Rain AI	Résolution du problème avec Rain AI et .Net	7h

Vendredi 31 mars 2017

Tâche	Description	temps
Documentation	Présentation des technologies utilisées, rédaction du rapport sur les futurs mini-projet et le problèmes rencontrés jusqu'à aujourd'hui	6h

Mardi 4 avril 2017

Tâche	Description	temps
Documentation	Présentation des technologies utilisées, rédaction du rapport sur les futurs mini-projet et le problème rencontrés jusqu'à aujourd'hui	4h

Mercredi 5 avril 2017

Tâche	Description	temps
Rain AI	Interaction entre deux IA, détection des IA entre elle	7h

Mardi 11 avril 2017

Tâche	Description	temps
Rain AI	Interaction entre les deux IA, détection des IA entre elle	7h

Jeudi 20 avril 2017

Tâche	Description	temps
Documentation	Documentation du scénario 2	3h

Mardi 25 avril 2017

Tâche	Description	temps
Rain AI Scenario 2	"Rework" des IA du scénario 2	6h

Mercredi 26 avril 2017

Tâche	Description	temps
Rain AI Scénario 3	Réalisation du début du scénario 3	6h

Vendredi 28 avril 2017

Tâche	Description	temps
Rendez-vous	Rendez vous avec M. Juergen, démonstration du travail déjà effectué et redéfinition du scénario 3	30mn
Projet	Début de réadaptation du Projet 3	6h

Mardi 2 mai 2017

Tâche	Description	temps
Doc	Documentation de la partie apprentissage	7h

Mercredi 3 mai 2017

Tâche	Description	temps
Doc	Documentation de la partie apprentissage	7h

Vendredi 4 mai 2017

Tâche	Description	temps
Doc	Documentation de la partie apprentissage et relecture	7h

Lundi 8 mai 2017

Tâche	Description	temps
Formation	Suivie des tutoriels sur les commandes vocales	2h
Documentation	Analyse et comparaison entre les commandes gestuelles et vocales	2h

Mardi 9 mai 2017

Tâche	Description	temps
Formation	Suivie des tutoriels sur les commandes gestuelles	2h
Documentation	Analyse et comparaison entre les commandes gestuelles et vocales	2h
Code	Adaptation du tutoriel de formation sur les commandes gestuelles sur le projet	4h

Mercredi 10 mai 2017

Tâche	Description	temps
Formation	Suivie des tutoriels sur les commandes gestuelles	2h
Code	Adaptation du tutoriel de formation sur les commandes gestuelles sur le projet	4h

Vendredi 12 mai 2017

Tâche	Description	temps
Formation	Suivie du tutoriel sur les commandes gestuelles	2h
Code	Adaptation du tutoriel de formation sur les commandes gestuelles sur le projet	4h

Mardi 16 mai 2017

Tâche	Description	temps
Formation	Suivi des tutoriels sur le tir	3h
Code	Adaptation de l'exemple du toolkit de menu	4h

Mercredi 17 mai 2017

Tâche	Description	temps
Code	Adaptation de l'exemple du toolkit de menu	4h

Vendredi 19 mai 2017

Tâche	Description	temps
Code	Adaptation de l'exemple du toolkit de menu	4h

Mardi 23 mai 2017

Tâche	Description	temps
Rendez-vous	Rendez vous avec M. Juergen, démonstration du travail déjà effectué et décision des modifications à faire sur le rapport	30mn
Documentation	Mise à jours de la documentation	3h
Unity	Ajout d'un civil et complexification du shooter	3h

Mercredi 24 mai 2017

Tâche	Description	temps
Documentation	Mise à jours de la documentation	1h
Unity	Ajout d'un civil et complexification du shooter	3h

Mardi 30 mai 2017

Tâche	Description	temps
Documentation	Mise à jours de la documentation	2h
Unity	complexification du shooter + compatibilité du Holo-toolkit avec IL2CPP	5h

Mercredi 1 juin 2017

Tâche	Description	temps
Documentation	Mise à jours de la documentation	2h
Unity	complexification du shooter + compatibilité du Holo-toolkit avec IL2CPP	5h

Vendredi 3 juin 2017

Tâche	Description	temps
Rendez-vous	Rendez vous avec M. Juergen, démonstration du travail déjà effectué et décision des modifications/ajout	30mn
Unity	Ajout de plusieurs civils et complexification du shooter et des civils	6h

Mardi 6 juin 2017

Tâche	Description	temps
Documentation	Mise à jours de la documentation	2h
Unity	Ajout de plusieurs civil et complexification du shooter et des civils	6h

Mercredi 7 juin 2017

Tâche	Description	temps
Documentation	Mise à jours de la documentation	2h
Unity	complexification du shooter et des civils et correction des problèmes de collision	6h

Vendredi 9 juin 2017

Tâche	Description	temps
Documentation	Mise à jours de la documentation	4h
Unity	complexification du shooter et des civils et correction des problèmes de collision	3h

Mardi 13 juin 2017

Tâche	Description	temps
Documentation	Mise à jour de la documentation et relecture	4h

Mercredi 14 juin 2017

Tâche	Description	temps
Documentation	Mise à jour de la documentation et relecture	4h

Jeudi 15 juin 2017

Tâche	Description	temps
Documentation	Mise à jour de la documentation et relecture	4h

Lundi 19 juin 2017

Tâche	Description	temps
Unity	Suppression de la chasse du shooter. Ajout de la commande vocale, réimplantation des commandes gestuelles via le biais du <i>Holotoolkit</i>	8h

Mardi 20 juin 2017

Tâche	Description	temps
Unity	Réimplantation des commandes gestuelles via le biais du <i>Holotoolkit</i>	4h
Déploiement	Déploiement sur Hololens et recherche du bug de RainAi et .Net4.6	4h

Mercredi 21 juin 2017

Tâche	Description	temps
Unity	Réimplantation des commandes gestuelles et vocale via le biais du <i>Holotoolkit</i>	4h
Déploiement	Déploiement sur Hololens et recherche du bug de RainAi et .Net4.6	4h

Jeudi 22 juin 2017

Tâche	Description	temps
Unity	Réimplantation des commandes gestuelles et vocale via le biais du <i>Holotoolkit</i>	2h
Déploiement	Déploiement sur Hololens et recherche du bug de RainAi et .Net4.6	4h
Documentation	Mise à jour de la documentation	2h

Vendredi 23 juin 2017

Tâche	Description	temps
Développement	Correction du bug de .Net4.6 et nettoyage du code	4h
Rendez-vous	Rendez-vous avec M. Juergen, debriefing du rapport intermédiaire	30mn
Documentation	Mise à jour de la documentation en fonction de ce qui s'est dit précédemment	4h

Lundi 26 juin 2017

Tâche	Description	temps
Développement	Correction du bug de .Net4.6 et nettoyage du code	4h
Développement	Simplification de l'IA du shooter et ajout de plus de <i>navigation taget</i> , généralisation du script de choix de <i>navigation taget</i>	2h
Documentation	Mise à jour de la documentation en fonction de ce qui s'est dit précédemment	2h

Mercredi 28 juin 2017

Tâche	Description	temps
Développement	Simplification de l'IA du shooter et ajout de plus de <i>navigation taget</i> , généralisation du script de choix de <i>navigation taget</i>	4h
Documentation	Mise à jour de la documentation	2h

Jeudi 29 juin 2017

Tâche	Description	temps
Développement	Création d'une nouvelle IA civil	5h
Documentation	Mise à jour de la documentation	2h

Vendredi 30 juin 2017

Tâche	Description	temps
Développement	Ajout de davantage de civils Mise à jour de l'IA du civil	7h
Démonstration	Démonstration du travail effectué	30mn

Lundi 3 juillet 2017

Tâche	Description	temps
Développement	Ajout deux deux IA, Adam et Michael	7h

Mardi 4 juillet 2017

Tâche	Description	temps
Développement	Ajout deux deux IA, Adam et Michael	2h
Documentation	Documentation de Adam et Michael	5h

Mercredi 5 juillet 2017

<i>Tâche</i>	<i>Description</i>	<i>temps</i>
Développement	Ajout d'animation pour Michael, refactor du code, ajout d'un enum pour les états	7h

Jeudi 6 juillet 2017

<i>Tâche</i>	<i>Description</i>	<i>temps</i>
Développement	Tentative d'amélioration des déplacements des civils pour éviter les collisions	7h

Vendredi 7 juillet 2017

<i>Tâche</i>	<i>Description</i>	<i>temps</i>
Développement	Ajout d'un Handler pour les commandes vocales	4h
Documentation	Mise à jours des schémas UML et leurs documentations relatives	3h

Lundi 10 juillet 2017

<i>Tâche</i>	<i>Description</i>	<i>temps</i>
Développement	Ajout d'un Handler pour les commandes vocales	1h
Documentation	Mise à jour du ReadME et documentation	6h

Mardi 11 juillet 2017

<i>Tâche</i>	<i>Description</i>	<i>temps</i>
Test	Test de commandes vocales et debug	2h
Développement	Navigation inGame et ajout de lumières pour marquer la safeZone	3h
Documentation	Mise à jour du ReadME et documentation	3h

Mercredi 12 juillet 2017

<i>Tâche</i>	<i>Description</i>	<i>temps</i>
Développement	Découverte de Polarith et test l'implémentations sur un mini-projet	6h

Jeudi 13 juillet 2017

<i>Tâche</i>	<i>Description</i>	<i>temps</i>
Développement	Découverte de Polarith et test l'implémentations sur le projet de base	7h

Vendredi 14 juillet 2017

<i>Tâche</i>	<i>Description</i>	<i>temps</i>
Développement	Découverte de Polarith et test l'implémentations sur le projet de base	6h
Documentation	Refactor du code et commentaires + documentation	2h

Lundi 17 juillet 2017

<i>Tâche</i>	<i>Description</i>	<i>temps</i>
Développement	Suppression des éléments de Polarith et correction de la navigation du menu de départ	1h
Documentation	Refactor du code et commentaires + documentation	6h

Mardi 18 juillet 2017

<i>Tâche</i>	<i>Description</i>	<i>temps</i>
Documentation	documentation	7h

Mercredi 19 juillet 2017

<i>Tâche</i>	<i>Description</i>	<i>temps</i>
Développement	Correction des BT et ajout de la commande reset	2h
Documentation	Refactor du code et commentaires + documentation	5h

Jeudi 20 juillet 2017

<i>Tâche</i>	<i>Description</i>	<i>temps</i>
Développement	Correction du civil et de Adam	2h
Documentation	Refactor du code et commentaires + documentation	5h

Vendredi 21 juillet 2017

<i>Tâche</i>	<i>Description</i>	<i>temps</i>
Tests	Test sur live Compilation et Hololens + doc des tests	8h

Lundi 24 juillet 2017

<i>Tâche</i>	<i>Description</i>	<i>temps</i>
Documentation	Relecture et correction	7h

Mardi 25 juillet 2017

<i>Tâche</i>	<i>Description</i>	<i>temps</i>
Documentation	Relecture et correction	7h