AKN_Regie : Un plugin dans Unreal Engine pour la direction d'avatar sur une scène mixte

Georges Gagneré

AKN_REGIE : UN PLUGIN DANS UNREAL ENGINE POUR LA DIRECTION D'AVATAR SUR UNE SCENE MIXTE

Georges Gagneré
INREV-AIAC – Université Paris 8
georges.gagnere [at] univ-paris8

RÉSUMÉ

L'article décrit le développement de l'outil AKN_Regie utilisé pour la mise en scène d'avatar sur une scène mixte dans le dispositif AvatarStaging. Après avoir résumé les fonctionnalités informatiques auxquelles devait répondre l'outil, nous décrivons l'environnement de programmation choisi, à savoir le langage visuel Blueprint dans Unreal Engine, le moteur de jeu vidéo développé par la société Epic Games. Nous expliquons le fonctionnement des EventDispatchers sur lesquels s'appuie la programmation. Nous décomposons les principaux nodes utilisés pour expliquer les relations entre les périphériques de contrôle et les avatars capturés en temps réel, ainsi que le fonctionnement de la conduite. Nous concluons en précisant les spécificités de l'outil concernant des questions d'offset et de player d'animations.

1. INTRODUCTION

Cette contribution approfondit les réflexions que j'avais proposées aux premières Journées d'Informatique Théâtrale en 2020 à Grenoble [4]. Il s'agit de présenter sous l'angle informatique AKN_Regie, l'outil que je développe depuis 2018 et dont la genèse et la chronologie de programmation sont décrites dans [10] jusqu'en 2019.

AKN_Regie est accessible à des non-spécialistes en informatique, comme le décrit Anastasiia Ternova dans une contribution parallèle dans ces JIT 2022 [13]. Il permet de résoudre des problèmes spécifiques à l'utilisation théâtrale en temps réel de données de capture de mouvement. Comme il est développé au sein d'un moteur de jeu vidéo, il reste en principe ouvert à l'intégration des vastes possibilités expressives de son environnement de programmation. Cet article vise à décrire l'architecture de l'outil et son environnement de programmation afin d'en faciliter l'appropriation pour les personnes intéressées par une prise en main de l'outil sous l'angle de son développement.

J'utiliserai les termes anglais génériques du moteur de jeu utilisé pour le développement.

2. CONTEXTE ET OBJECTIFS DE DEVELOPPEMENT

2.1. Le contexte AvatarStaging

AKN_Regie est l'outil informatique qui permet de contrôler des avatars en temps réel dans le dispositif AvatarStaging [8] (figure 1).

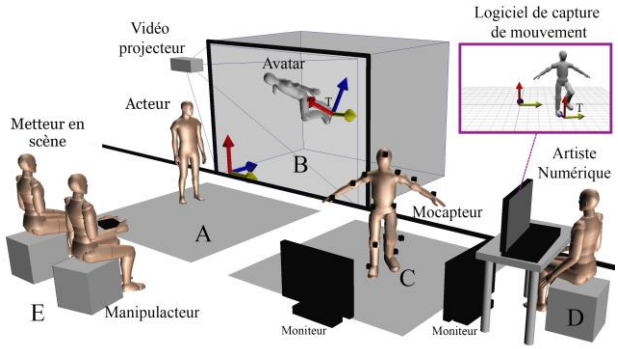

Figure 1. Le dispositif AvatarStaging

AvatarStaging consiste à faire interagir un acteur physique avec un avatar sur une scène mixte dans un contexte de spectacle vivant. L'animation de l'avatar se réalise de différentes manières. La figure 1 décompose la situation où un acteur portant une combinaison de capture de mouvement, le mocapteur, et évoluant dans l'espace C, contrôle en temps réel un avatar présent dans l'espace virtuel B. Cet avatar est lui-même en relation avec un acteur physique jouant dans la partie A de la scène mixte qui recouvre les parties A et B devant le public, ici représenté par le metteur en scène et le manipulacteur. Ce dernier est un acteur qui contrôle aussi l'avatar en utilisant non pas son corps mais un périphérique de type manette de jeu (gamepad) ou contrôleur MIDI. Ce deuxième type de contrôle est par exemple nécessaire pour ajuster l'adresse scénique de l'avatar en relation avec les mouvements de l'acteur physique [11].

Sur le plan informatique, AvatarStaging implique les besoins suivants, rassemblés dans l'outil AKN_Regie :

- Récupérer les données de périphériques de capture de mouvement en temps réel pour les appliquer sur des avatars
- Positionner précisément les avatars dans l'espace 3D et les orienter avec des contrôleurs de jeu
- Programmer une conduite de configurations successives de manipulation des avatars dans l'espace 3D
- Utiliser le clavier et des contrôleurs MIDI pour manipuler la conduite au fil d'une performance

AKN_Regie a été utilisé sur des projets différents [9] [5] [13] et possède un mode d'emploi accessible aux non-spécialistes en informatique sur un site web dédié [1]. La dimension temps réel de la capture de mouvement et de l'interaction avec un acteur physique a conduit à faire les développements dans le moteur de jeu vidéo Unreal Engine, développé par la société Epic Games [2]. Le moteur a été choisi principalement en raison de l'existence de l'environnement de programmation visuel Blueprint qui permet de programmer sans avoir à écrire de code sous forme de scripts.

2.2. Les blueprints dans Unreal Engine

Le moteur Unreal Engine est développé en C++ et les sources sont accessibles, modifiables et compilables. La programmation d'un jeu avec le moteur peut se faire de deux manières : en utilisant le langage C++ sous forme de script ou bien en utilisant une sorte de transposition du langage C++ sous forme visuelle spécifiquement développée dans Unreal Engine par Epic Games sous le nom de Blueprint [3]. Par extension, le nom blueprint désigne aussi chaque élément de code programmé avec le langage visuel Blueprint, composé de blocs de programmation appelés nodes et agencés dans un graphe.

La figure 2 montre l'interface d'édition des blueprints avec l'exemple du blueprint Regie_Cuesheet réalisé avec l'outil AKN_Regie. Un utilisateur non spécialiste en informatique est capable de manipuler les boîtes bleutées (correspondant aux nodes) (figure 2B) et de paramétrer une conduite sous la forme de menu dans la colonne de droite (figure 2A). L'utilisation de l'outil du point de vue utilisateur est détaillée dans [13].

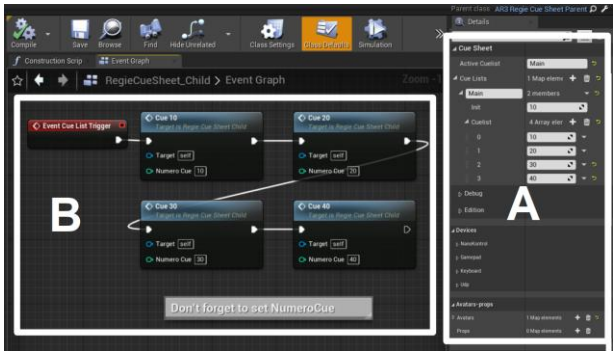

Figure 2. Le blueprint Regie_Cuesheet. A : édition des cues, B : paramétrage de la conduite

Unreal Engine suit les principes de programmation des moteurs de jeu vidéo qui reposent sur une interface graphique permettant d'accéder à l'espace 3D, appelé level, afin de le remplir des maillages du décor, des lumières et de tous les éléments de programmation nécessaires au jeu. Pendant la programmation, le level est lancé en mode jeu pour être testé par le développeur. Et lorsque l'ensemble de la programmation est terminé et validé, le jeu est packagé sous la forme d'un standalone qui se lance directement en mode jeu par le joueur et qui n'a plus besoin de l'environnement de programmation global, comme la traduction française de l'expression le confirme (se tenir seul). Ainsi un standalone représente un ensemble de fichiers d'un poids de quelques centaines de Mo alors que l'environnement de programmation du moteur vidéo représente des dizaines de Go de fichiers.

La version actuelle d'AKN_Regie nécessite l'interface d'édition et l'environnement de programmation d'Unreal Engine, qui doit être installé au préalable Après avoir programmé l'outil pour manipuler les avatars, l'utilisateur lance le moteur en mode jeu et visualise les résultats dans le level souhaité. Il n'y a pas pour le moment de standalone d'AKN_Regie.

2.3. Développement en blueprint dans un plugin

Pour des raisons de facilité d'appropriation des nombreuses possibilités de programmation offertes par Unreal Engine, le développement d'AKN_Regie est aussi conduit en langage blueprint en se limitant aux nodes du moteur faisant partie du noyau central présent par défaut à l'installation (Built-In Engine) et ne nécessite l'activation que d'un plugin complémentaire pour la gestion des événements MIDI.

La notion de plugin dans l'environnement de programmation d'Unreal Engine constitue un moyen de compartimenter de manière autonome du code qui peut alors facilement s'ajouter à un projet existant et donner accès à de nouveaux nodes pour réaliser des actions spécifiques. On peut par exemple ajouter un plugin de périphérique de capture de mouvement qui permettra alors de recevoir les datas du périphérique en question connecté à l'ordinateur. AKN_Regie a donc été placé dans un plugin autonome qui peut être ajouté à n'importe quel projet existant afin d'utiliser l'outil avec les avatars souhaités. Grâce à cette fonctionnalité, l'outil peut être utilisé sur des projets différents.

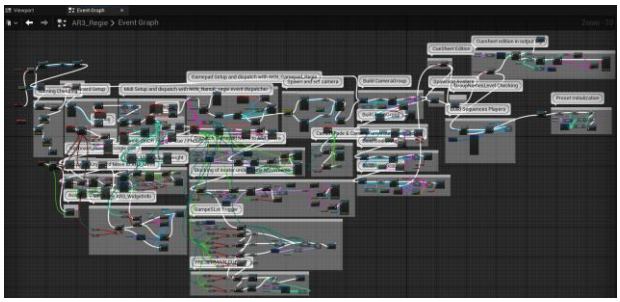

Figure 3. L'EventGraph du blueprint AR3_Regie

Le plugin AKN_Regie contient ainsi une collection de blueprints dont le principal, AR3_Regie, a été développé en langage blueprint avec l'interface d'édition d'Unreal

Engine, de la même manière qu'un utilisateur non-spécialiste configure un blueprint simplifié du type de la figure 2 pour utiliser AKN_Regie. Tous les nodes utilisés sont mis à jour par défaut à chaque nouvelle version du moteur ce qui assure une consistance du plugin au fil de l'évolution du moteur. Bien que difficilement lisible, la figure 3 permet de visualiser une petite partie de l'agencement des nodes dans AR3_Regie nécessaire pour faire fonctionner le blueprint de la figure 2 destiné à l'utilisateur non-spécialiste en informatique.

3. L'ARCHITECTURE D'AKN_REGIE

Sans entrer trop précisément dans les caractéristiques de la programmation C++, on peut introduire succinctement la notion fondamentale de classe, qui permet de compartimenter les éléments de programmation et de générer ce qu'on appelle des enfants (ou children) qui bénéficient des fonctionnalités de la classe parent en permettant une réorganisation et l'implémentation de nouvelles spécifications. On peut notamment « cacher » du code pour en simplifier l'utilisation. Ce principe de parentage relie le blueprint parent AR3_Regie (fig. 3) à Regie_Cuesheet (fig. 2).

Chaque blueprint est une classe C++ qui doit être placée dans le level, sous forme d'instance, pour exécuter ses instructions. On en distingue deux catégories selon la modalité de présence dans un level. Soit on le place directement dans le level, on parlera alors d'Actor, soit on l'ajoute à un Actor déjà existant, on parlera alors d'Actor Component ou Component.

Ainsi l'utilisateur d'AKN_Regie programme dans le blueprint enfant Regie_Cuesheet, que l'on appellera désormais AR3_Regie_Child pour indiquer son parentage avec le blueprint principal AR3_Regie dont nous allons maintenant expliquer les fonctionnalités et l'architecture. AR3 signifie qu'il s'agit de la troisième version de développement de l'outil **A**KN_**R**egie.

3.1. Génération des avatars et positionnement de goals

Dans la construction d'un jeu vidéo, il y a différentes manières d'instancier les éléments utiles à la navigation active dans le level : ils peuvent y être directement placés à l'aide de l'interface d'édition principale, mais on peut aussi programmer la génération d'éléments (opération encore appelée spawn) à tout moment du jeu, et notamment au moment du lancement en indiquant alors à quels endroits ils devront être générés. Généralement dans un jeu vidéo, les objets 3D qui constituent le décor fixe de la scène sont placés directement dans le level. En revanche, l'avatar du joueur est généré au démarrage et placé sur une position de départ (player start).

Nous procédons de même avec l'outil AKN_Regie en construisant directement dans le level le décor de la scène virtuelle et nous donnons à l'utilisateur la possibilité de faire une distribution de personnages (ou encore casting) à partir d'une liste d'avatars prédéfinis et équipés pour être manipulés par le blueprint principal AR3_Regie. Ce casting d'avatars est alors généré au lancement du jeu. Nous demandons aussi à l'utilisateur de placer dans le level les endroits successifs où il positionnera les avatars du casting. Nous utilisons ainsi un blueprint de classe SkeletalMeshActor, que l'on appelle GoalAvatar (GA) qui contient par défaut un maillage de Robot, ajustable pour chaque instance placée dans le level (figure 4). L'ensemble des GoalAvatars dans un level correspond en quelque sorte aux placements successifs du casting des avatars, encore appelé blocking dans le monde anglo-saxon. Nous parlerons d'offset pour décrire l'opération de repositionnement des éléments générés.

Le même processus s'applique pour le placement d'accessoires que l'on souhaite manipuler dans le décor avec le positionnement de blueprints de classe TargetPointActor et appelés GoalProps (GP) et pour les positionnements de la caméra qui rendra les vues souhaitées sur le level avec des blueprints de type CineCameraActor et appelé GoalCameras (GC).

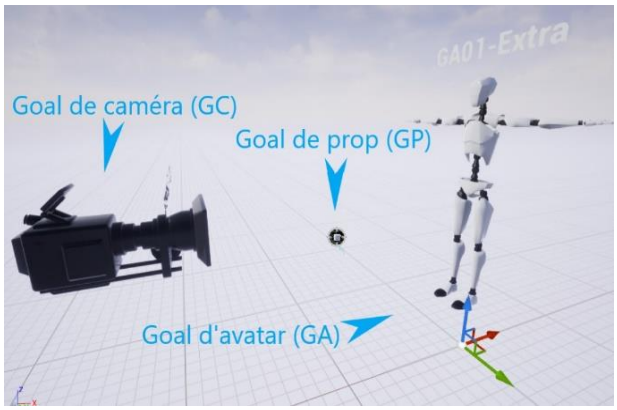

Figure 4. Les 3 types de goals dans le level 3D

L'utilisateur d'AKN_Regie doit donc construire un décor dans un level, y glisser une instance unique du blueprint AR3_Regie_child et les instances des trois types de goals pour toutes les positions souhaitées des avatars, des props et de la caméra. Les goals sont donc essentiellement des marqueurs qui ont l'apparence des objets dont ils indiquent les positions dans le level [13].

3.2. Le blueprint principal AR3_Regie

Les principales fonctionnalités d'AR3_Regie sont les suivantes :

- gestion des périphériques extérieurs,
- génération des avatars et des props,
- mise en place d'une architecture de nodes permettant de transmettre des événements,
- lancement et exécution de cues contenant les effets à appliquer sur les éléments générés.

La gestion des périphériques extérieurs s'effectue en utilisant des blueprints de type Actor ou Component et contenant la programmation spécifique à chaque périphérique. Pour la récupération des signaux MIDI émis par les périphériques connectés à l'ordinateur, il faut notamment activer le plugin MIDI Device Support qui fait partie de la collection des Built-In plugins présents par défaut dans le moteur d'Unreal Engine. Concernant le

signal MIDI, nous nous sommes concentrés pour le moment sur le contrôleur NanoKontrol fabriqué par Korg. Les gamepads utilisées avec AKN_Regie sont des manettes de jeu Microsoft Xbox directement reconnues par Unreal Engine selon un protocole différent. Il peut y en avoir 4 reconnues simultanément sur un ordinateur.

Les Components et les Actors ainsi générés envoient tous les événements émis par les contrôleurs MIDI, les gamepads (Actor AKN_GamePd_Input) et le clavier (Actor AKN_Keyboard_Input) sur trois principaux EventDispatchers nommés AKN_NanoK_Regie, AKN_Gamepad_Regie et AKN_Keyboard_Regie, qui en assureront la transmission à travers toute l'architecture d'AKN_Regie et dont nous expliquerons le principe dans la section suivante.

AR3_Regie se charge ensuite de générer les avatars et les props du casting choisi et il interprète la configuration programmée des périphériques pour effectuer deux tâches principales :

- activer la progression dans l'exécution des cues en avant ou en arrière en suivant la liste des cues (cuelist) et en partant de la cue d'initialisation ; le contrôle est réalisé avec des touches du clavier ou d'un NanoKontrol spécifiquement programmées ;
- transmettre des mouvements de translation avant ou arrière ou de rotation horaire ou antihoraire avec le clavier ou les boutons d'une gamepad sur un avatar ou un prop.

3.3. Les EventDispatchers dans Unreal Engine

Un node de type EventDispatcher est un élément de programmation fréquemment utilisé dans Unreal Engine et garantissant une optimisation de la transmission des événements (event) en temps réel dans la perspective d'un jeu vidéo nécessairement interactif.

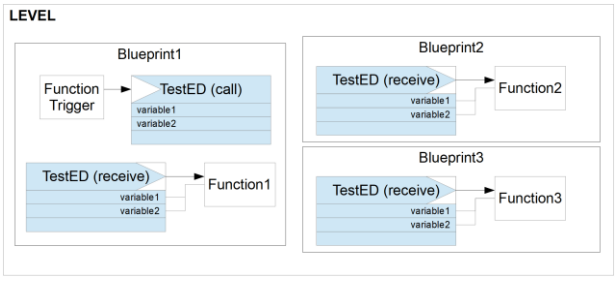

Figure 4. Fonctionnalités des EventDispatchers

La figure 4 schématise l'utilisation d'un exemple d'EventDispatcher nommé TestED et implémenté dans le Blueprint1. Le node FunctionTrigger déclenche le node d'appel de TestED (call) en transmettant deux variables et un event. Cela déclenche l'action du node de réception de TestED (receive) qui transmet instantanément l'event et les deux variables associées à tous les blueprints qui le possèdent, en l'occurrence le Blueprint1 mais aussi les Blueprint2 et Blueprint3 distincts mais également placés dans le level. Grâce aux EventDispatchers, la transmission d'events et de leurs variables de tout type associées est assurée dans les blueprints qui les contiennent, mais aussi à travers le level dans tous les blueprints qui en implémentent un receive.

4. PROGRAMMATION D'AR3_REGIE

4.1. Transmission des events

La figure 5 montre avec un exemple comment AR3_Regie transmet les informations pour contrôler le mouvement d'un avatar ou le déclenchement d'une cue.

L'objectif souhaité et programmé dans la rubrique Devices d'AR3_Regie_Child (le blueprint enfant Regie_Cuesheet accessible aux non-programmeurs, cf. figure 2A) est de translater vers l'avant (forward) Avatar1, choisi dans le casting, avec le thumbstick gauche sur l'axe Y de la Gamepad1. Toutes les opérations suivantes sont réalisées à chaque tick du moteur, qui représente l'unité temporelle d'exécution de tous les calculs et dont le cadencement est généralement supérieur à 30 frames par seconde (fps).

La première étape consiste à récupérer l'information du périphérique extérieur, en l'occurrence la gamepad, lorsqu'on effectue la manipulation souhaitée avec le thumbstick. Le blueprint AKN_GamePd_Input (cité en section 3.2) transmet à chaque tick un event associé à toutes les valeurs qu'il reçoit, donc notamment la valeur d'inclinaison du thumbstick de gauche sur l'axe des Y de la gamepad choisie, à savoir Gamepad1. Ces informations sont envoyées dans le blueprint AR3_Regie avec l'EventDispatcher AKN_Gamepad_Regie (mode call) qui transmet la gamepad active, la partie utilisée et la valeur transmise (1 si le thumbstick gauche est complètement poussé).

La seconde étape consiste à interpréter les informations envoyées avec la manière dont la section Devices d'AR3_Regie_Child a été programmée. Les informations d'AKN_Gamepad_Regie (en mode receive) sont filtrées par la configuration Devices qui sélectionne l'avatar (Avatar1) et le type de mouvement (forward) et déclenche l'EventDispatcher AR3_Move qui transmet l'avatar et les informations de mouvement correspondants. Grâce à cet EventDispatcher, chaque élément généré par AR3_Regie pourra lire l'information, et agir en conséquence s'il est concerné, comme nous le verrons dans la section suivante.

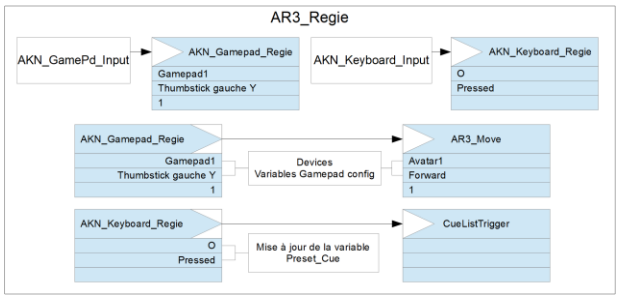

Figure 5. Architecture du blueprint AR3_Regie

La progression dans la conduite s'effectue d'une manière similaire. Imaginons que les variables keyboard de Devices (figure 2A) soient configurées pour que les

touches O et P du clavier fassent respectivement avancer et reculer dans la cuelist. Lorsqu'on appuie sur le clavier, le blueprint AKN_Keyboard_Input repère la lettre appuyée, en l'occurence O, et transmet l'information à l'EventDispatcher AKN_Keyboard_Regie. Le node receive de l'EventDispatcher interroge la configuration Devices pour savoir s'il faut incrémenter ou décrémenter la variable Preset-Cue qui indique la position dans la cuelist et déclenche l'event CueListTrigger avec la nouvelle valeur de Preset-Cue (n+1). Le déclenchement de cet event permet alors d'exécuter la cue adéquate parmi toutes celles programmées dans le blueprint AR3_Regie_Child, comme nous allons l'expliquer.

4.2. Fonctionnement et programmation de la Cuesheet

L'ensemble des opérations réalisées dans la figure 5 se déroulent au niveau du blueprint parent AR3_Regie auquel l'utilisateur n'a pas accès. Ce dernier ne peut changer que les éléments à sa disposition dans le blueprint enfant AR3_Regie_Child, dont la figure 6 schématise la programmation des cues et la configuration des variables, et qui correspond exactement à la présentation graphique de la figure 2.

On connecte en série à un receive de l'event CueListTrigger les nodes de cue (en l'occurrence les deux cues 10 et 20) qui contiennent elles-mêmes des nodes de type Set. Ces derniers consistent à envoyer des ordres aux éléments générés par AR3_Regie. Nous avons pris l'exemple d'instruction concernant le positionnement dans le level avec l'EventDispatcher AR3_Offset. Les nodes SetAvatar, SetCamera et SetProp utilisent un call d'AR3_Offset en envoyant comme informations le nom de l'élément généré (Avatar1 par exemple) et le goal souhaité (GA01 par exemple). Un receive de cet EventDispatcher placé dans chaque élément généré récupère les instructions, comme nous allons le voir dans la section suivante.

L'initialisation au lancement du jeu se fait sur la Cue10, et si on appuie sur O, la variable Preset-Cue passe à la valeur 2 indiquant que la cue active est désormais Cue20. Le déclenchement de l'event CueListTrigger interroge toutes les cues programmées et n'exécute que le node correspondant à la cue active, grâce à un filtre sur la variable Preset-Cue.

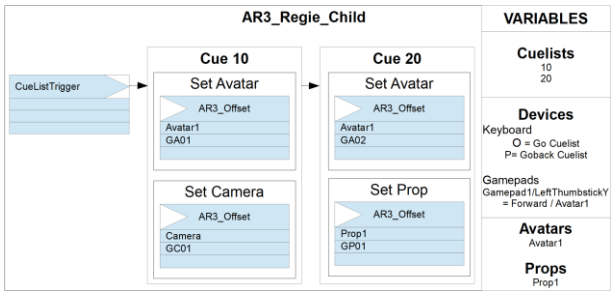

Figure 6. Schéma du blueprint AR3_Regie_Child

Dans l'exemple, la Cue10 envoie l'Avatar1 sur le GoalAvatar GA01 et la caméra sur le GoalCamera GC01. La Cue20 change la position d'Avatar1 sur le GA02 et positionne l'accessoire Prop1 sur le GoalProp GP01. Dans le tableau de configuration des variables à droite on constate que la gamepad peut aussi translater Avatar1 vers l'avant avec le thumbstick Y gauche de Gamepad1. Le casting est limité à Avatar1 et Prop1.

4.3. Pilotage des avatars et des props

Pour conclure cette introduction succincte à l'architecture d'AKN_Regie, on peut visualiser dans la figure 7 ce qui se passe après avoir configuré les variables Devices et Avatars-props, programmé les cues dans AR3_Regie_Child, placé le blueprint dans un level, lancé le jeu et appuyé sur la touche O qui déclenche le passage à la cue suivante d'une cuelist qui comporte deux éléments Cue10 et Cue20.

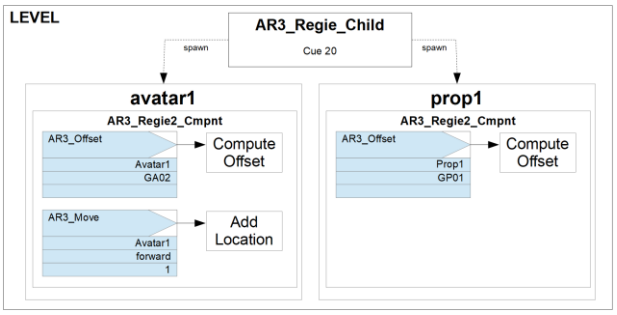

Figure 7. AR3_Regie_Child joué dans le level

Lorsque le jeu est lancé, AR3_Regie_Child génère dans le level Avatar1 et Prop1 présents dans le casting. Ces deux blueprints possèdent chacun le blueprint Component AR3_Regie2_Cmpnt qui contient, entre autres, les receive d'EventDispatchers comme AR3_Move et AR3_Offset d'AR3_Regie. La figure 7 montre ainsi les nodes actifs lorsque la Cue20 est déclenchée et que la Gamepad1 est utilisée : Avatar1 reconnaît les messages d'offset et de move qui lui sont destinés et les exécute respectivement dans les nodes ComputeOffset et AddLocation, et de même pour l'offset de Prop1.

5. DISCUSSION

Les éléments de programmation qui viennent d'être décrits correspondent aux fonctionnalités originelles d'AKN_Regie concernant le contrôle d'avatars pilotés en temps réel par des mocapteurs dans le dispositif AvatarStaging. Le calcul d'offset pour placer un avatar sur un goal a nécessité un développement spécifique lié à la nature des données de capture de mouvement, qui ont pu être ajustées en temps réel en blueprint.

5.1. Le calcul de l'offset en blueprint

La figure 8 reproduit le schéma manuscrit originellement dessiné pour traduire géométriquement le repositionnement d'un avatar mocapté dans un level. Les besoins artistiques de ce repositionnement ont été documentés dans [11] et l'impact des ajustements sur la mise en scène dans [9].

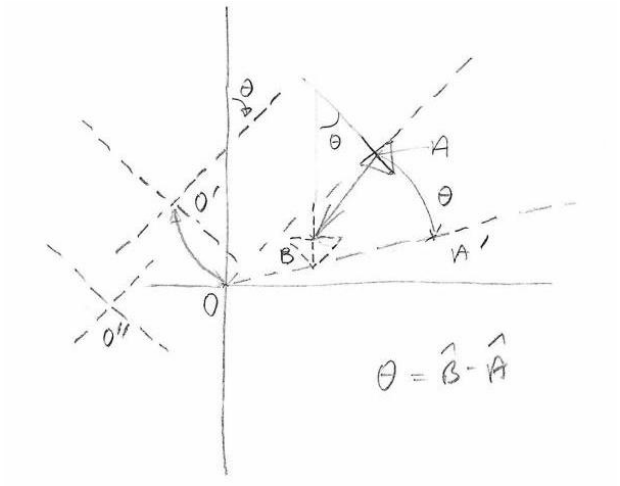

Figure 8. Transformation géométrique du référentiel d'un avatar.

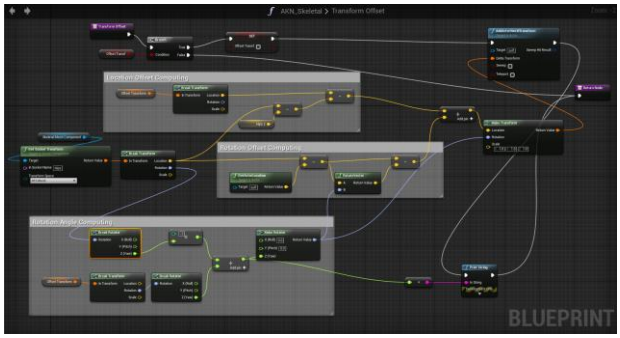

Figure 9. Transcription blueprint du calcul d'offset par changement de référentiel

Lorsqu'on récupère les données de capture de mouvement d'un avatar situé en A, les informations de positions et de rotations des articulations du squelette de l'avatar sont calculées par rapport à une origine O qui reste fixe. Repositionner l'avatar indiqué par une flèche en A sur une nouvelle position indiquée par la flèche en pointillés en B consiste en pratique à appliquer une rotation d'angle θ sur le point A qui vient en A', puis une translation $\overrightarrow{A'B}$. Cela revient à faire un changement de référentiel de O en O'' avec une rotation d'angle θ.

Les nodes en mode blueprint de la figure 9 permettent de calculer cette transformation d'origine. Il suffit alors d'implémenter ce calcul dans le node Compute Offset de la figure 7 pour repositionner précisément tout avatar mocapté en temps réel dans un level. Ce calcul sert aussi pour faire tourner l'avatar sur lui-même.

5.2. LiveLink et Salient-Idle Player

La seconde version d'AKN_Regie a intégré la fonctionnalité LiveLink apparue dans Unreal Engine à partir de la version 4.18. Cette fonctionnalité facilite entre autres la relation avec les logiciels tiers de capture de mouvement tels que Motive (pour le système optique Optitrack) ou Axis Studio (pour les systèmes inertiels Perception Neuron), ainsi qu'avec le logiciel Motion Builder (développé par Autodesk) pour les enjeux de motion retargeting. Ces développements ont conduit à créer des bibliothèques d'avatars prêts à l'emploi avec AKN_Regie et à offrir une méthode pour équiper tout avatar importable dans Unreal Engine et le rendre contrôlable avec AKN_Regie.

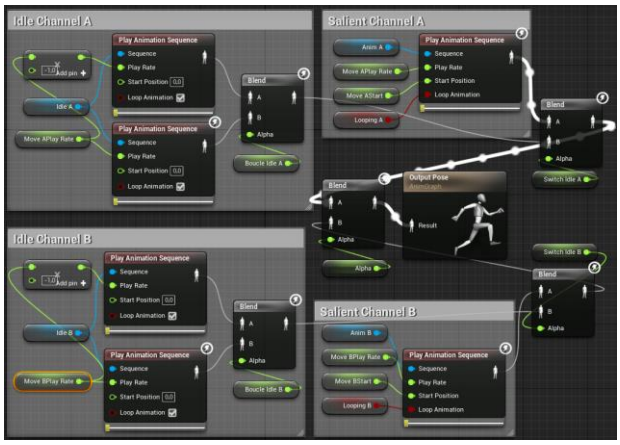

Figure 10. Nodes de mixage du Salient-Idle Player

La troisième version d'AKN_Regie a intégré le développement d'un player spécifique d'animations, le Salient-Idle Player (figure 10), qui permet de contrôler finement des avatars sans recourir à la capture de mouvement en temps réel mais en agençant des animations préenregistrées selon une approche combinant théâtre et jeu vidéo [7].

5.3. Perspectives

Le titre de l'article fait référence à la direction d'avatar, plutôt qu'au contrôle. Il nous semble en effet que la combinaison de la capture de mouvement et d'un ajustement avec une gamepad permet à un metteur en scène utilisant AKN_Regie de diriger un avatar en collaboration avec un mocapteur dans un contexte de prévisualisation in situ [12]. En effet, le metteur en scène visualise au moment de la capture des mouvements un résultat qui sera quasiment identique à celui offert au public pendant les représentations. L'ajout progressif de fonctionnalités comme celles du Salient-Idle Player a permis d'en enrichir sensiblement les usages.

On peut noter que le passage d'une utilisation du blueprint AR3_Regie_Child, sans connaissance informatique approfondie, au développement du blueprint parent AR3_Regie permettant de mettre en place de nouvelles fonctionnalités, demande beaucoup d'efforts [6]. C'est pourtant une évolution qui nous semble nécessaire dans la pratique d'AKN_Regie pour bénéficier de toutes ses potentialités créatives.

La prochaine étape de développement consiste à prendre en compte la capture de mouvement faciale et à intégrer les outils d'intelligence artificielle d'Unreal Engine tels que les automates à états finis et les arbres comportementaux afin de doter les avatars d'éléments d'autonomie scénique.

6. REFERENCES

[1] Didascalie.net, AKN_Regie, http://avatarstaging.eu (consulté le 22 janvier 2023)

[2] Epic Games, Unreal Engine, http://unrealengine.com (consulté le 22 janvier 2023)

[3] Epic Games, Blueprint, https://docs.unrealengine.com/5.0/en-US/blueprints-visual-scripting-in-unreal-engine/ (consulté le 22 janvier 2023)

[4] Gagneré G. « Du théâtre à l'informatique : bascule dans un nouveau monde », *Actes des Journées d'Informatique Théâtrale*, février 2020, Grenoble

[5] Gagneré G. « *The Shadow* », *Proceedings of the 7th International Conference on Movement and Computing (MOCO '20)*, Association for Computing Machinery, New York, NY, USA, 2020

[6] Gagneré G. « AKN_Regie : une passerelle entre arts numériques et spectacle vivant », *Actes du colloque Frontières Numériques 2023*, 5ème édition, Hammamet, juin 2023, Saleh I., Szoniecky S., Ghemina M. (dir.), Ed. Europia Production, 2023

[7] Gagneré G., Mays T., Ternova A. « How a Hyper-actor directs Avatars in Virtual Shadow Theater », *Proceedings of the 7th International Conference on Movement and Computing (MOCO '20)*, Association for Computing Machinery, New York, USA, 2020

[8] Gagneré G., Plessiet C. « Experiencing avatar direction in low cost theatrical mixed reality setup », *Proceedings of the 5th International Conference on Movement and Computing (MOCO '18)*, Association for Computing Machinery, New York, USA, 2018

[9] Gagneré G., Plessiet C. « Espace virtuel interconnecté et Théâtre (2). Influences sur le jeu scénique », *Revue : Internet des objets* , Numéro 1, Volume : 3, ISTE OpenScience, 2019

[10] Gagneré G., Plessiet C. « Quand le jeu vidéo est le catalyseur d'expérimentations théâtrales (2014-2019) », *Le jeu vidéo au carrefour de l'histoire, des arts et des médias*, dir. C. Devès, Lyon, Les Éditions du CRHI, 2023

[11] Gagneré G., Plessiet C., Sohier R. « Intercon-nected virtual space and Theater. Practice as research on theater stage in the era of the network » in *Challenges of the Internet of Things. Technology, Use, Ethics*, Volume 7 (edited by I. Saleh, M. Ammi, S. Szoniecky), Wiley, 2018

[12] Plessiet C., Gagneré G., « Mises en scène spectaculaires et interactives de l'avatar. Deux approches en recherche-création utilisant des technologies de prévisualisation en temps réel », in *Hybrid* [En ligne], 9 | 2022, mis en ligne le 30 novembre 2022

[13] Ternova A., Gagneré G. « Le potentiel créatif du plugin AKN_Regie dans un contexte théâtral », in *Actes des Journées d'Informatique Théâtrale* – octobre 2022, Lyon, 2023