Projet 4INFO

Sport-Santé et numérique : Application intégrant une reconnaissance prédictive de geste pour lutter contre la sédentarisation

Stay In Motion

Rapport de conception

Auteurs :
Firmin Cadot
Teddy Gesbert
Léandre Le Bizec

Encadrants :
Éric Anquetil, professeur à l'INSA et chercheur à l'Irisa
William Mocaër, doctorant à l'IRISA
Richard Kulpa, maître de conférence à Rennes 2 et chercheur à l'Irisa

En collaboration avec les étudiants de DIGISPORT :
Guillaume Cobat
Marie-Aurélie Castel
Kilian Bertholon

2022-2023

Table des matières

1 Introduction

La réalisation de l'application "Stay In Motion" (SIM) et du démonstrateur associé au moteur de reconnaissance de gestes 3D pour la recherche s'inscrit dans le cadre de notre seconde année au sein du département informatique de l'INSA de Rennes. Toujours dans le but de nous préparer au travail d'ingénieur, il a pour objectif de nous confronter à des problématiques techniques et des problématiques de gestion de projet en nous mettant au défi de mener à bien chaque étape du processus jusqu'à sa concrétisation.

Le projet que nous menons tient son origine d'un problème d'enjeu majeur : le manque d'activité physique chez une partie de la jeune population. En effet, selon l'ANSES, deux tiers des jeunes de 11-17 ans présentent un risque sanitaire associé à la sédentarité et à l'inactivité physique[3]. C'est une problématique qui tend à se détériorer d'année en année. Parallèlement à cette baisse de la pratique du sport, l'utilisation du numérique ne cesse de croître[8]. De plus, ces dernières années, on peut observer un attrait tout particulier pour l'intelligence artificielle qui est désormais utilisée dans beaucoup de nos objets numériques du quotidien. Ainsi, le fait d'imaginer une application où l'utilisateur pourrait effectuer des séances de sport, en lien avec un environnement virtuel, et qui utilise un système d'intelligence artificielle renforçant l'interaction en temps réel, pourrait être considéré comme attrayant pour un jeune public et pourrait ainsi leur donner goût à l'effort physique.

L'application que nous cherchons à développer s'inscrit dans un cadre précis. En 2018/2019 a été réalisé le projet "DERG3D"[10] qui est une application servant de démonstrateur au moteur de reconnaissance de geste de Yacine Boulahia [6]. S'en est suivi le projet "R3G" [5] en 2020/2021, qui est une suite logicielle plus complète proposant des outils de visualisation plus avancées, ainsi qu'un démonstrateur adapté au nouveau moteur de reconnaissance de gestes [16]. L'année dernière, en 2021/2022, s'est formé le projet "MOP" [4] qui est une preuve de concept de l'application visant à lutter contre le manque d'activité physique chez une partie de la jeune population. Nous avons alors deux objectifs :

- Développer la preuve de concept "MOP" à l'état d'application : "SIM" ;
- À partir d'une brique logicielle de R3G, créer un démonstrateur destiné à la recherche.

Ce contexte est explicité en figure 1.

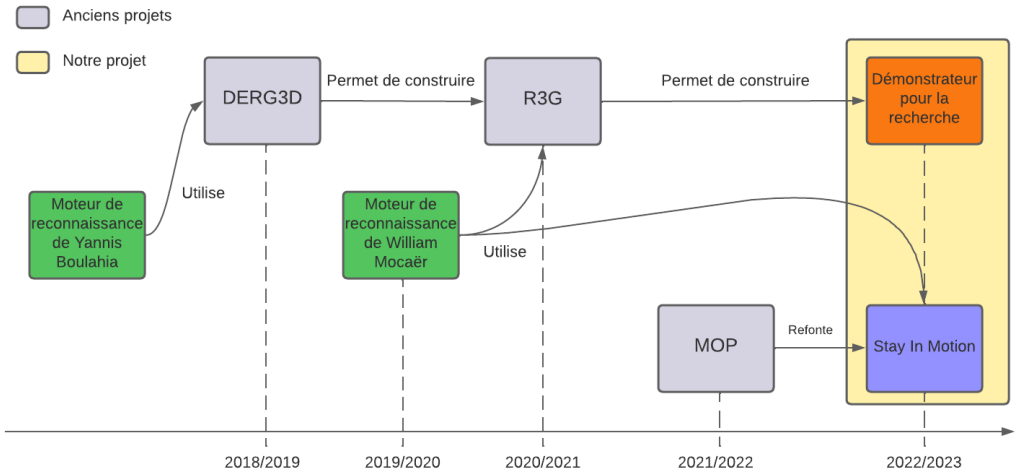

FIGURE 1 – Contexte de développement de notre projet

Le démonstrateur associé au moteur de reconnaissance de gestes 3D à pour objectif d'assister l'équipe IntuiDoc afin d'améliorer le système de reconnaissance de gestes. Pour cela, le démonstrateur permettra de tester de manière interactive la capacité du moteur à décider rapidement du geste qui sera effectué par l'utilisateur. L'évolution de la prise de décision devra être visuelle afin de permettre d'en comprendre le plus rapidement possible tous ses aspects. Le démonstrateur devra être robuste et capable d'utiliser coorectement les resultats du moteur. Nous pourrons également expérimenter sa qualité et sa précision. Cette partie du projet sera donc un outil spécifique à la recherche.

Les similarités de conception entre SIM et le démonstrateur permettent à notre équipe de traiter ces deux objectifs en parallèle. Toutefois, il existe également de nombreuses différences de conception. Notre but à travers ce rapport est de présenter comment nous allons exploiter les bases et architectures logicielles à notre disposition et de quelle manière nous allons les améliorer afin de répondre au cahier des charges.

2 Développement et Architecture sous Unity

Avant de présenter les architectures logicielles sur lesquelles nous allons travailler, il est important de présenter notre environnement de développement, Unity, ainsi que les patrons de conceptions qui régiront nos productions. Ces prérequis sont nécessaires pour bien comprendre les architectures que nous présenterons par la suite.

2.1 Le patron de conception « Entités-Composants »

L'architecture « Entité-Composant » est une approche de conception dans laquelle les éléments qui composent une application sont d'abord organisés en une hiérarchie (tel que présenté en figure 2), puis les fonctionnalités et les données de chaque élément sont définies. En termes de programmation, une entité est un objet avec une liste de composants. Cette architecture aide à structurer le code de manière efficace, mais peut devenir complexe pour les grands projets en raison du nombre important d'entités et de composants. Il est donc important d'avoir une stratégie claire pour éviter la confusion. C'est l'objectif du modèle MVC (Model, View, Controller) en général et du modèle AMVCC (Application, Model, View, Controller, Component) que nous allons expliquer en section 2.3.

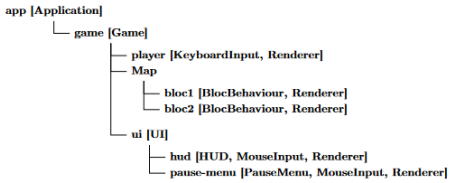

FIGURE 2 – Exemple d'une hiérarchie entités-composants

2.2 Unity et « Entités-Composants »

Unity, un moteur de développement d'application d'environnement 2D et 3D populaire, utilise le modèle Entité-Composant pour organiser les objets des environnements et leur comportement. Dans Unity, les objets sont appelés entités et leur comportement est défini par des composants. Chaque composant représente une fonctionnalité particulière associée à l'objet, comme la position, la physique de l'objet, l'audio, la vidéo, etc. Les composants peuvent être ajoutés ou retirés d'une entité à tout moment pour changer son comportement. De cette façon, Unity peut créer de nombreux objets d'environnement différents à partir d'un nombre limité de composants. Les développeurs peuvent également créer de nouveaux composants personnalisés pour ajouter de nouvelles fonctionnalités aux objets. En utilisant le modèle Entité-Composant, Unity peut diviser les objets en partie plus petit et plus gérables pour faciliter le développement et la maintenance d'une application.

2.3 Modèle AMVCC

Dans le développement pour Unity, le modèle basique Model-View-Controller (MVC) est souvent utilisé avec des modifications pour s'adapter à différentes situations rencontrées dans ce framework. Ce modèle se base sur trois modules qui ont chacun une responsabilité différente au sein l'application[1]. Pour faire face aux défis rencontrés avec le MVC standard, le modèle AMVCC (Application-Modèle-Vue-Contrôleur-Composant) a été introduit. Cela inclut une application qui est le seul point d'entrée du projet et des composants qui sont des scripts réutilisables. Le comportement de ce modèle, représenté par la figure 3, est le suivant :

Les modèles sont responsables de gérer les données et l'état de l'application, ainsi que de les sérialiser, désérialiser et convertir les types. Ils notifient les contrôleurs des progrès de chaque opération, mais n'ont jamais accès aux vues bien qu'ils les mettent à jour indirectement.

Les vues peuvent accéder aux données des modèles pour afficher l'état du jeu, mais ne doivent jamais les modifier. Elles n'ont aucune donnée importante pour l'application. Elles peuvent dans certains cas interagir directement avec le contrôleur (par exemple en cas de clic de l'utilisateur), mais dans notre projet, cela ne sera que peu le cas puisque le contrôleur recevra principalement en entrée les données venues des capteurs de gestes ou du moteur de reconnaissance de gestes.

Les contrôleurs peuvent utiliser et mettre à jour les données des modèles, mais n'ont aucune donnée importante pour l'application.

Le patron AMVCC est choisi pour son architecture structurée et la facilité d'ajouter de nouvelles fonctionnalités (composants), un exemple de cette structure se trouve en figure 4.

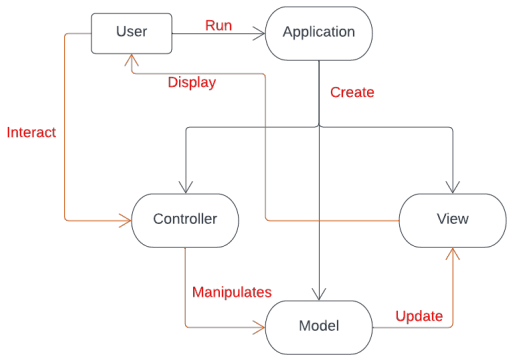

FIGURE 3 – Aperçu de l'architecture Entité-Composant au sein d'une scène unity

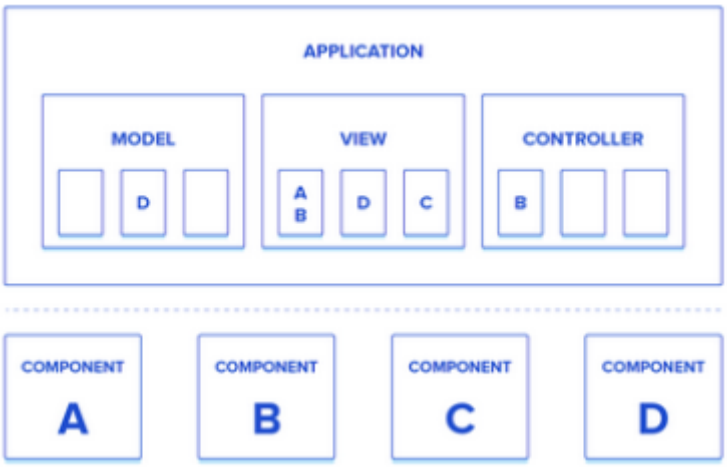

FIGURE 4 – Aperçu de l'architecture Entité-Composant au sein d'une scène unity

2.4 Modèle générique de nos applications

Nos applications suivront donc toujours le patron de conception AMVCC dont voici l'UML en figure 5

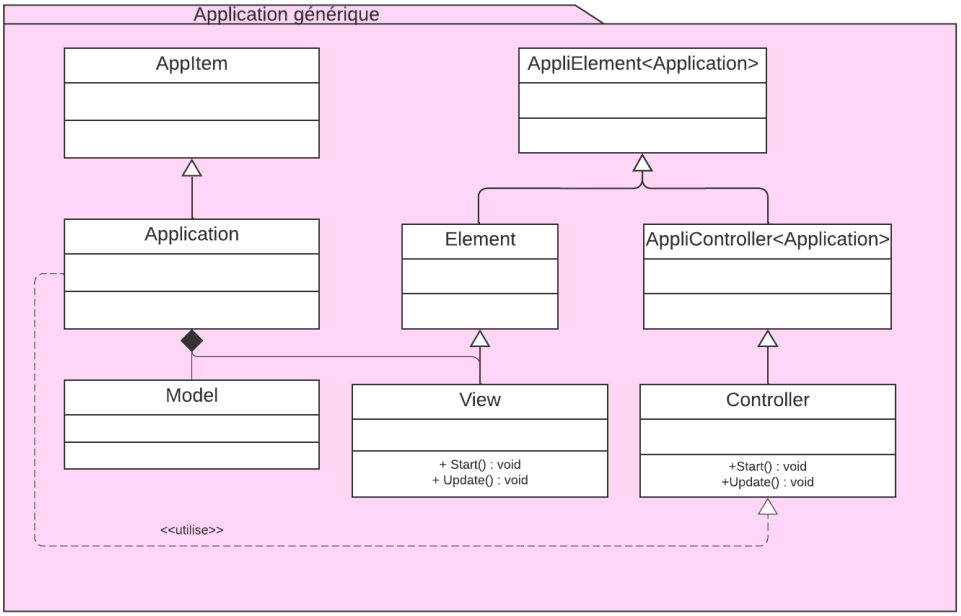

FIGURE 5 – Diagramme UML générique d'une application avec le modèle AMVCC

3 Architecture commune de Stay In Motion et du démonstrateur

La figure 6 permet de comprendre visuellement quelles sont les architectures communes de SIM et du démonstrateur. Les données reçues par les capteurs sont traitées de manière commune et générique par le bloc Gestion des capteurs qui sera représenté en bleu. Ces données peuvent être utilisées de deux manières différentes :

— Elles sont envoyées directement à "SIM", ce qui permet un feedback à l'utilisateur sur par exemple la qualité d'exécution de son geste, le temps d'exécution ou autres informations ne nécessitant pas l'utilisation du moteur de reconnaissance de gestes ;

— Elles sont envoyées au moteur de reconnaissance de gestes et ensuite utilisées dans le démonstrateur ou dans SIM.

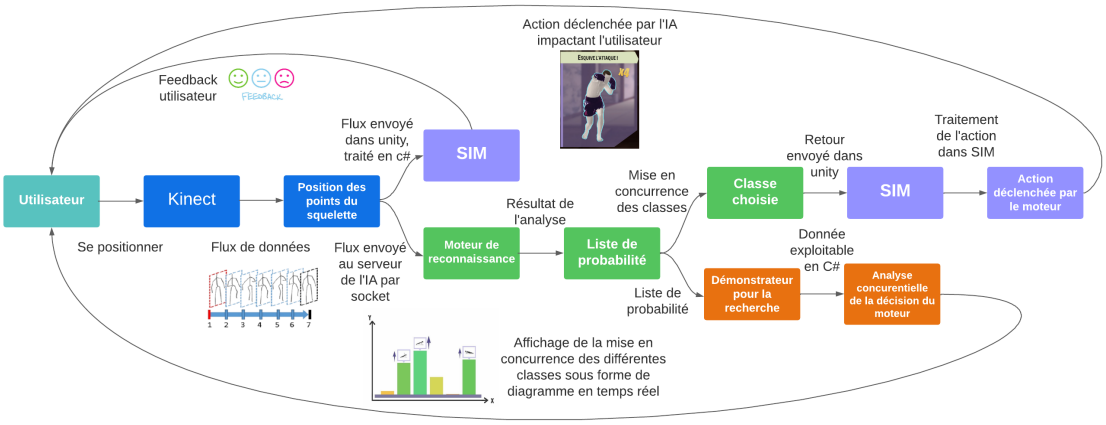

FIGURE 6 – Workflow du projet

Pour gérer cette structure au niveau du code, nous avons fait le choix de séparer chaque module dans des packages afin de conserver une architecture très claire. La figure 7 explicite cette décomposition en package.

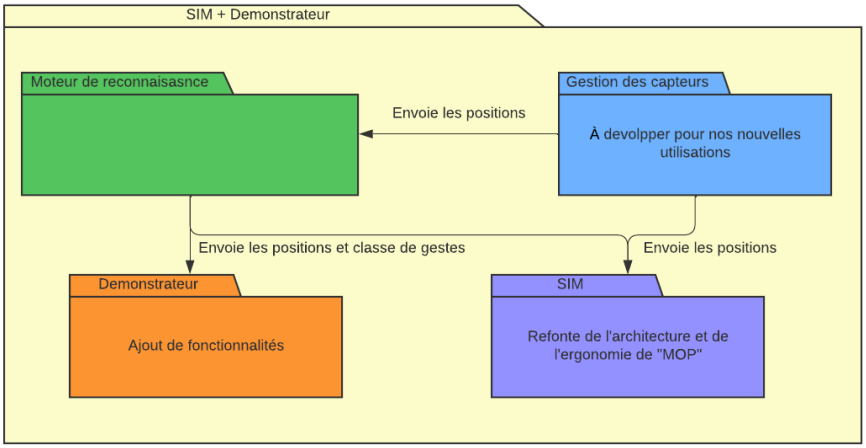

FIGURE 7 – Chaîne de traitement de l'architecture des projets

Au sein de notre projet, nous disposons donc de quatres packages :
— En bleu : Le bloc de gestion des capteurs détaillé en figure 8 ;
— En vert : Le bloc du moteur de reconnaissance de gestes détaillé en figure 9 ;
— En orange : Le bloc du démonstrateur détaillé en figure 17 ;
— En violet : Le bloc de "SIM" détaillé en figure 10.
Ce code couleur sera utilisé tout au long du rapport.

Le module de gestion des capteurs envoie les données de position récupérées par le périphérique de capture au module "SIM" ou au module de reconnaissance de geste.
— Le module "SIM" récupère la position brute et rend des feedbacks à l'utilisateur comme expliqué à la figure 6 ;
— Le module de reconnaissance de gestes renvoie l'information traitée afin de permettre les différentes actions expliquées dans la figure 6 :
  — au démonstrateur ;
  — à "SIM".

3.1 Généricité de l'acquisition de donnée

Pour récupérer les données des capteurs, il faut passer par la classe DeviceManager. Le DeviceManager communique avec les capteurs grâce à son attribut DeviceInfo. L'organisation est décrite dans la figure 8. Les informations sont extraites des capteurs au moyen de l'interface IDeviceInfoTaker qui permet de s'abstraire des particularités des capteurs. Chaque capteur nécessite sa propre implémentation de cette interface. Nous avons à notre disposition la Kinect V1, la Kinect V2, le Leap Motion, la Kinect Azure et un simulateur de la Kinect Azure. Les implémentations relatives aux Kinect ont déjà été implémentées. Ces implémentations sont représentées sur la figure 8 par les classes <NomDuCapteur>InfoTaker. Les méthodes définies dans l'interface IDeviceInfoTaker prennent en argument un objet JointsKnowApp qui correspond à une énumération des

articulations connus et utilisé par les applications. Les informations concernant ces articulations doivent être fournies par tous les capteurs afin d'assurer le bon fonctionnement des applications avec chaque capteur. Ainsi, la couche de généricité apportée par le DeviceManager nous permet d'interroger tous les capteurs dont nous disposons et les potentiels futurs capteurs. La classe DeviceManager respecte le patron de conception Singleton. Il est donc possible de récupérer le capteur dans n'importe quel module du projet.

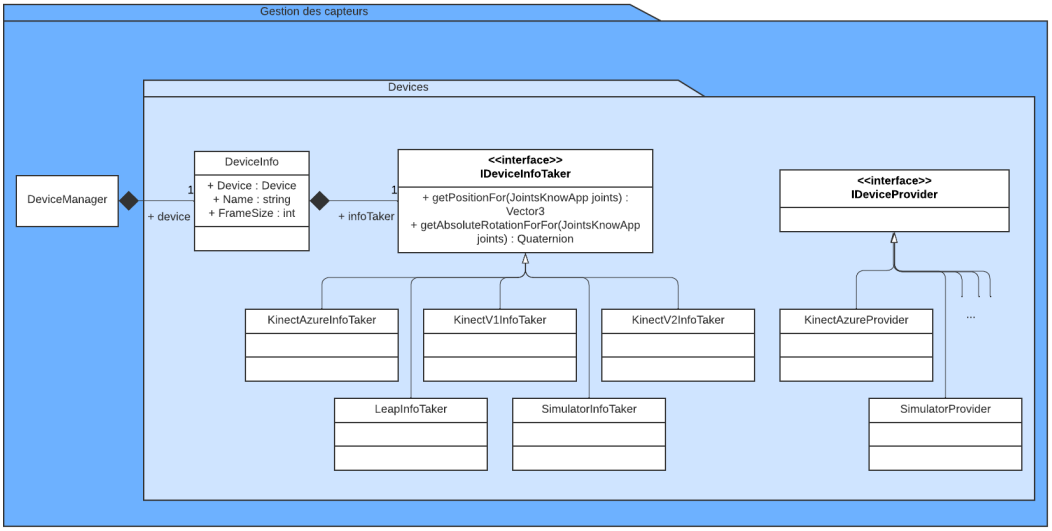

FIGURE 8 – Diagramme de classes du module de gestion des capteurs

Les providers ont pour rôle d'assurer le suivi du tracking pour un capteur spécifique, ainsi que d'afficher le squelette de l'utilisateur. Les providers sont à fournir au Controller. Le Controller prend en paramètre tous les capteurs référencés, et associe à ces derniers des gameObject comprenant notamment les scripts providers.

Cette architecture du module de la gestion des capteurs permet de répondre à la problématique de généricité présentée dans le tableau 1.

TABLE 1 – Cahier des charges

N°	Intitulé	Critère	Priorité
Refonte de l'application côté développeur			1
N°1.1	Généricité des capteurs	Architecture permettant d'ajouter de manière normée un nouvel appareil de capture.	1
Généricité			3
N°8.2	Généricité des capteurs	Notre démonstrateur doit pouvoir afficher fonctionnellement l'utilisateur et les résultats de l'IA en temps réel peu importe le capteur de mouvement utilisé.	3.1

3.2 Génération des avatars 3D liés aux mouvements dans nos applications

En parallèle de la refonte de l'architecture de "SIM", la première étape de notre phase de conception commune aux deux versions est de remplacer les squelettes, générés par le suivi de corps d'un utilisateur, par des avatars 3D. En effet, un utilisateur utilisant "SIM" doit pouvoir se visualiser et visualiser un coach. Initialement, le squelette généré par le suivi de corps par le provider du capteur est affiché pour l'utilisateur et le coach, mais cette solution n'est pas très esthétique. Un avatar 3D permet d'améliorer l'immersion et l'ergonomie de l'application. Il faut également que la solution proposée soit générique quant aux appareils de captures utilisés.

Récupérer le mouvement du squelette afin de l'appliquer à l'avatar de manière générique Pour ce faire, il est possible de récupérer la rotation absolue des articulations captées sur le squelette et de mettre à jour cette rotation sur l'avatar mappé au squelette. Afin de rendre l'avatar visible dans l'application compatible avec tous les capteurs, nous récupérerons cette rotation en passant par le DeviceManager. La méthode getAbsoluteRotaionFor() présente sur la figure 8 dans la classe abstraite IDeviceInfoTaker permet cela. Ce sera donc à nous de créer cette méthode et de l'implémenter pour les différents capteurs.

Ce qui nous permet de répondre aux points du cahier des charges reporté dans le tableau 2.

Table 2 – Cahier des charges

N°	Intitulé	Critère	Priorité
Refonte ergonomique de l'application			2
N°2.1	Coach virtuel	Avatar 3D montrant le mouvement à faire.	2.1
N°2.2	Modèle 3D	Avatar 3D représentant l'utilisateur sportif.	2.1

3.3 Communication avec le moteur de reconnaissance de gestes

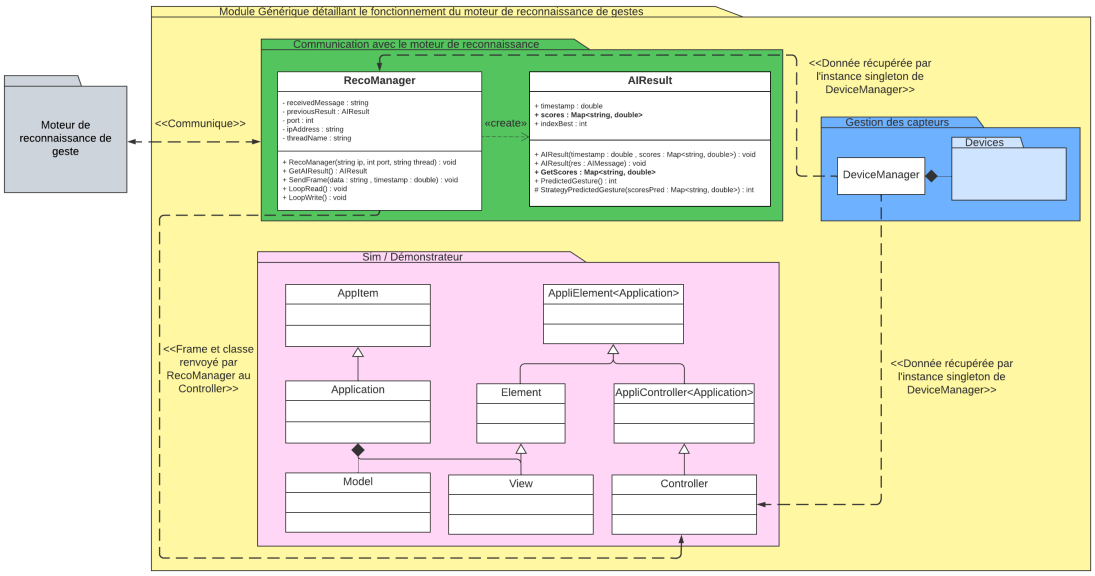

FIGURE 9 – Diagramme de classes du module permettant de communiquer avec le moteur de reconnaissance de gestes

4 Application Stay in Motion

SIM permet de mettre en place des séances d'activités physiques. Une séance comportera plusieurs exercices prédéfinis, accompagnés par un entraîneur virtuel au sein d'un environnement virtuel projeté sur un écran. Pendant ces exercices, il y a un retour direct sous la forme d'un avatar représentant l'utilisateur en réalisant ses gestes, ainsi que des conseils pour exécuter le mouvement de la bonne manière et divers retours sur sa réalisation. Les exercices peuvent être catégorisés en 3 types selon la manière dont la performance de l'utilisateur est mesurée :

- Type «collision», qui utilise la position dans l'espace de l'utilisateur ;
- Type «angle», qui utilise les angles des articulations de l'utilisateur ;
- Type «IA», qui utilise le moteur de reconnaissance de gestes.

Les deux premiers types d'exercices présentent une même structure. En effet, pour ces deux types, un traitement direct de la position est effectué : dans le type collision le système s'intéressera aux positions de différentes articulations alors que pour le type angle il s'intéressera aux angles entre plusieurs articulations.
L'exercice de type IA qui utilisera le moteur de reconnaissance de gestes nécessitera une structure différente. C'est pourquoi nous séparerons ces deux structures dans la présentation.

4.1 Exercice type «Collision» et «Angle»

Étant donné que les exercices de type «angle» et «collision» présentent une même structure, nous détaillerons ici le fonctionnement d'un exercice de squat qui est donc de type «angle», l'ensemble de ces exercices fonctionnant de la même manière.

4.1.1 Architecture d'un exercice type Collision ou Angle

Chaque exercice récupère les données des capteurs, les formate et adapte son affichage à ces données. L'architecture utilisée suit le modèle AMVCC défini précédemment. Un exercice est donc composé d'une classe Application qui contient le modèle, et la vue (cf. Fig10). Le Controller peut lui modifier des données de la vue ou du modèle en héritant de la classe AppliController<Application>. Le modèle contient l'ensemble des données utiles à l'exercice : un compteur du nombre de squat, l'angle actuel, le dernier angle reçu, l'état de position (Relevé, baissé avec angle correct, en train de se baisser, etc.) et un dictionnaire associant un état de position à un certain feedback. La vue quant à elle peut accéder aux données du modèle puisqu'elle hérite de la classe Element qui elle-même hérite de la classe AppliElement<Application>.

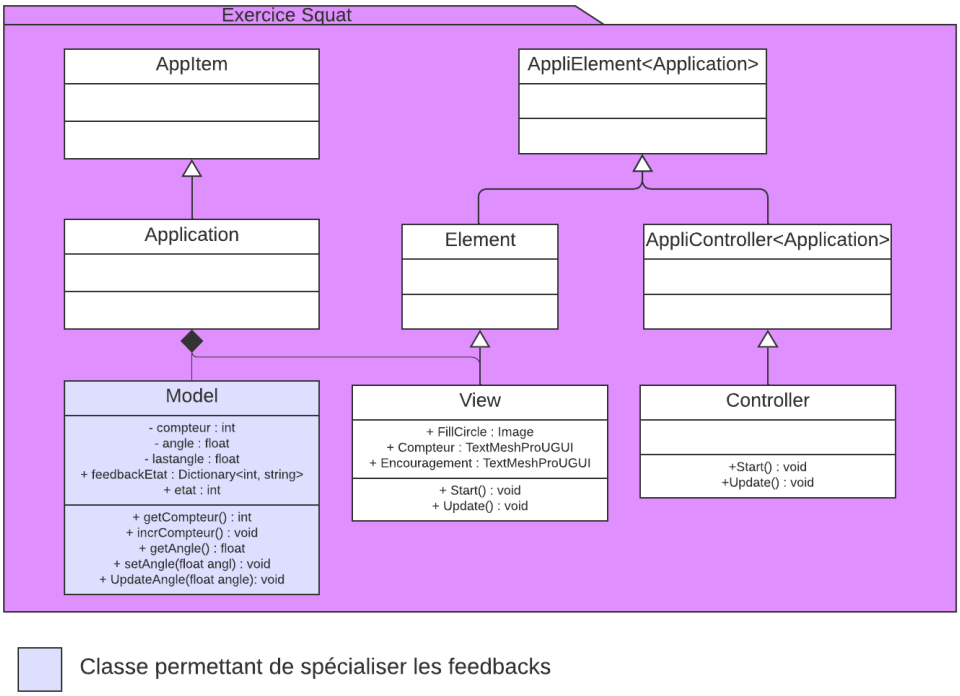

FIGURE 10 – Diagramme de classes décrivant l'architecture d'un exercice (ici un exercice de squat)

4.1.2 Comportement d'un exercice type Collision ou Angle

Une fois la scène lancée, les classes Contrôleur et View sont créées par l'application (via héritage) qui va créer dans sa méthode Awake un modèle11. La classe DeviceManager, responsable de récupérer les données des capteurs, est elle aussi créée via un provider. Ensuite pour chaque frame reçu provenant du capteur de position, le DeviceManager récupère la position de certaines articulations précises. Ces positions sont ensuite récupérées par le contrôleur afin d'effectuer un calcul d'angle, puis vont mettre à jour la valeur de cet angle dans le modèle. Cela va permettre de mettre à jour l'ensemble des attributs du modèle, que va récupérer la vue afin d'actualiser les composants des entités de la scène. L'utilisateur va donc recevoir un retour direct de son geste effectué.

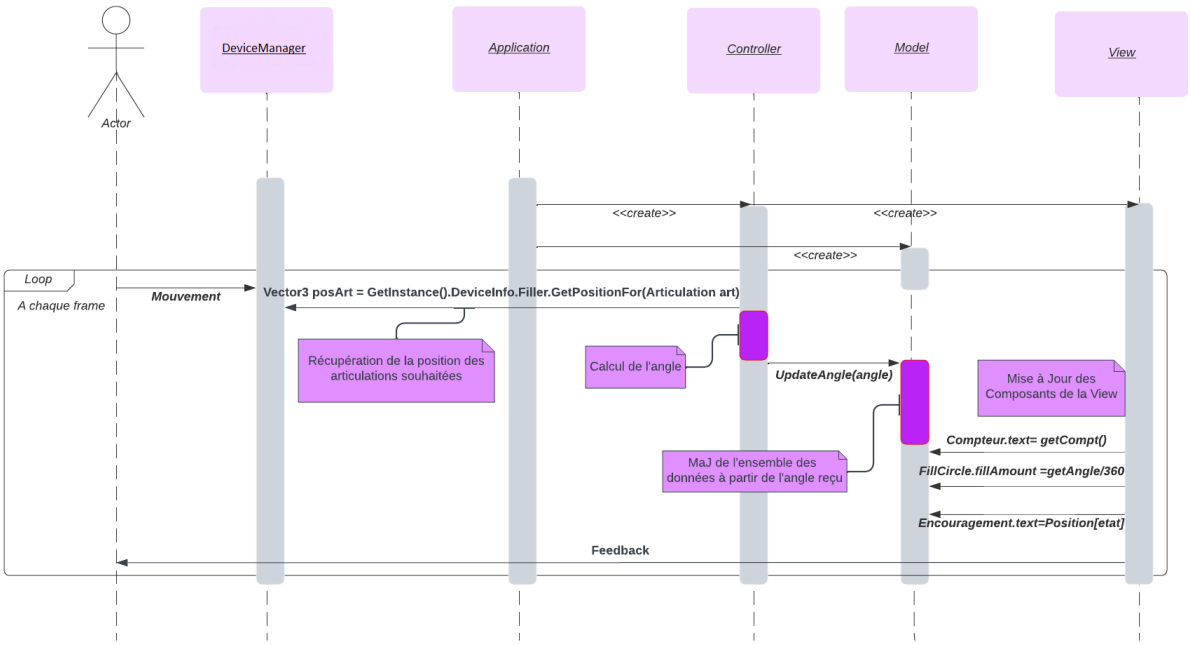

FIGURE 11 – Diagramme de séquence d'un exercice de squat

Cette implémentation nous permet ainsi de répondre aux points du cahier des charges reporté dans le tableau 3.

TABLE 3 – Cahier des charges

N°	Intitulé	Critère	Priorité
Refonte de l'application côté développeur			1
N°2.4	Timer	Indication permettant de connaître le temps restant d'un exercice ou d'un temps de repos.	2.2
N°2.5	Barre de progression	Indication permettant de connaître le score de l'utilisateur sportif.	2.2
N°2.6	Nombre de répétitions	Indication permettant de connaître le nombre de répétitions de l'utilisateur sportif.	2.2
N°2.7	Environnement	Mise en place du fond et des couleurs.	2.2

4.2 Exercice type IA

4.2.1 Architecture d'un exercice type IA

L'architecture d'un exercice de type IA sera légèrement différente. En effet, ici le système s'intéresse au geste reconnu par le moteur de reconnaissance de geste et non pas au mouvement propre de l'utilisateur. Un exercice de type IA est un exercice qui se présentera sous la forme d'un jeu, où une action de jeu sera générée lorsque le moteur de reconnaissance reconnaîtra un certain geste.

Cet exercice sera créé en suivant l'UML présenter à la figure 9 puis en spécialisant la classe Model de l'exercice afin de l'adapter à l'utilisation du moteur de reconnaissance de gestes

4.2.2 Comportement d'un exercice type IA

Pour cet exercice, le client, établissant la connexion Serveur IA - Application, va recevoir directement les données des capteurs récupérés par le device manager12. Celui-ci va ensuite envoyer ces données au serveur IA qui va les traiter et transmettre au client sa réponse. Le client va enfin transmettre au contrôleur cette réponse sous la forme d'une action de jeu (comme par exemple le déplacement d'un avatar).
L'application garde le même comportement par la suite, seules les données d'entrées du contrôleur changent.

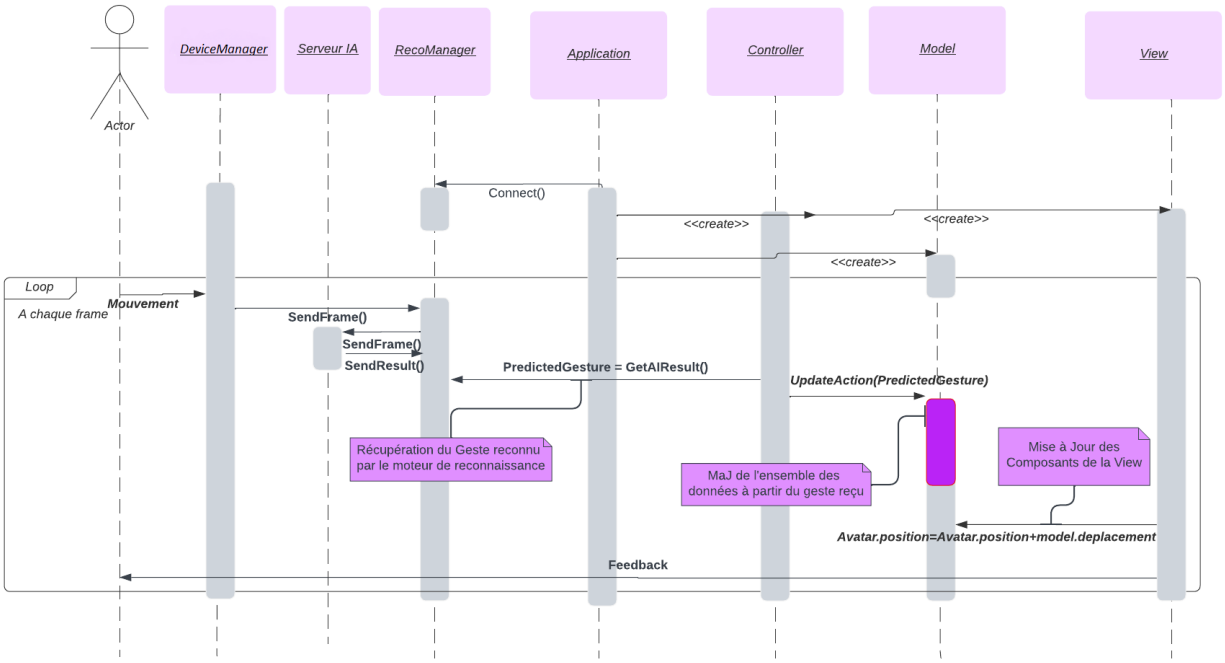

FIGURE 12 – Diagramme de séquence d'un exercice type IA

Cette implémentation nous permet ainsi de répondre aux points du cahier des charges reporté dans le tableau 4.

TABLE 4 – Cahier des charges

N°	Intitulé	Critère	Priorité
Ajout d'un exercice implémentant l'IA			5
N°3.3	Nombre de coups réussis	Nombre de coups reconnus par l'IA.	5.2
N°3.4	Nombre de coups qui ont touché	Nombre de coups reconnus par l'IA n'ayant pas déclenché la réaction du coach virtuel à temps.	5.2

4.3 Structure de l'application SIM

L'application "SIM" doit être le plus ergonomique possible. Il doit y avoir un menu permettant de naviguer entre les différents modes de l'application initialement prévue et les exercices doivent s'enchaîner rapidement afin de ne pas couper une séance de sport.

La classe sceneManager présentée en figure 13 permet de charger les différentes scènes de notre projet. Chaque scène représente un exercice. Pour ajouter un nouvel exercice il suffit de :

— Créer une nouvelle scène en suivant le modèle précédent de l'exercice squat ;
— Ajouter un lien entre la nouvelle scène créée et le SceneManager.

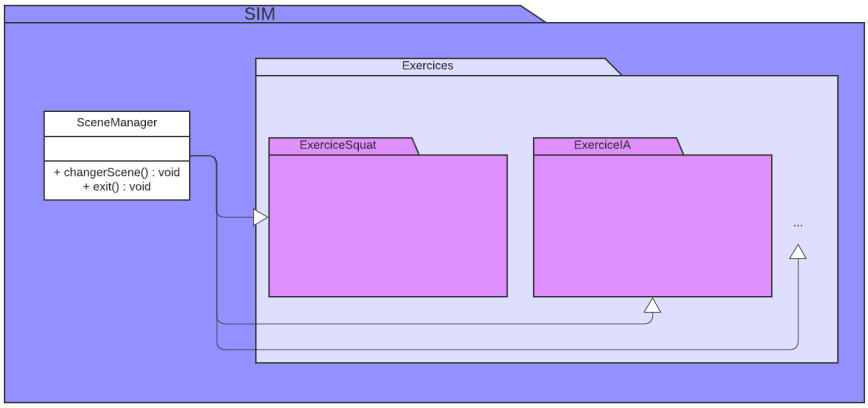

FIGURE 13 – Diagramme UML de la structure de "SIM" avec le sceneManager

Tout cela nous permet ainsi de répondre aux points du cahier des charges reporté dans le tableau 5.

TABLE 5 – Cahier des charges

N°	Intitulé	Critère	Priorité
Refonte de l'application côté développeur			1
N°2.3	Avancement de la séance	Indication permettant de connaître l'avancement de la séance.	2.2

5 Démonstrateur associé au moteur de reconnaissance de gestes 3D

Pour rappel, le moteur de reconnaissance de geste qui sera liée à notre démonstrateur recevra en permanence des frames grâce aux dispositifs de capture de mouvement. Après le traitement de ces frames par le réseau de neurones, le démonstrateur affichera à l'écran les scores associés à chaque geste calculé par le classifieur en temps réel.

5.1 Présentation du projet R3G et des améliorations futures

Le projet R3G est composé de cinq applications, dissociables en deux blocs. Le premier bloc (en bleu sur la figure 14) capture et analyse les gestes de l'utilisateur, ce qui permet de créer une base de données et de visualiser les performances du dispositif. Le développement de celui-ci a été réalisé sur Unity. Les gestes de l'utilisateur sont alors étudiés par l'IA et leur case respective sur l'interface passe au vert clair lorsque le système a suffisamment confiance pour prendre sa décision. Le second bloc (en rouge sur la figure 14) analyse les données recueillies après que celles-ci aient été annotées, puis évalue l'efficacité de l'IA.

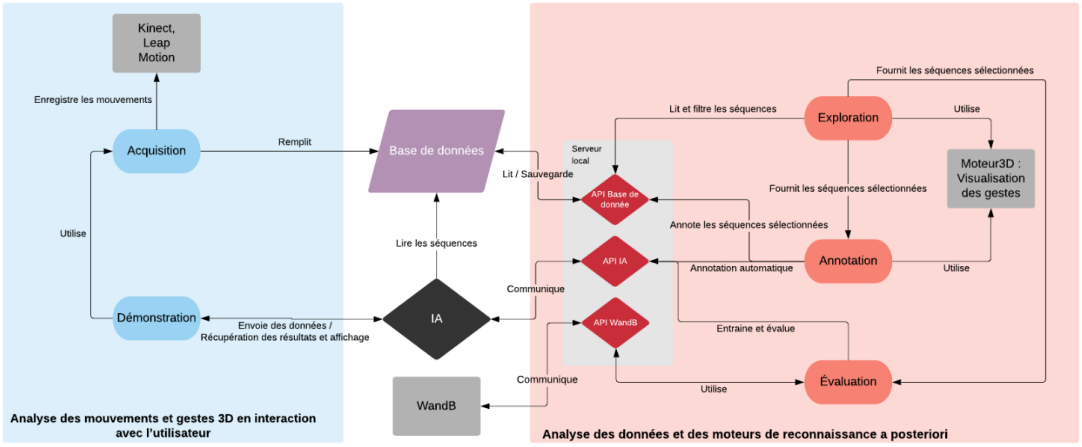

FIGURE 14 – Diagramme de l'architecture globale de la suite R3G décrivant les interactions entre les cinq modules, l'IA et la base de données [5]

Pour notre projet, nous allons nous focaliser uniquement sur l'architecture du premier bloc (en bleu sur la figure 14) de R3G qui se compose du module de démonstration et d'acquisition. La figure 15 décrit les différents composants de notre nouveau démonstrateur au niveau macroscopique. Les composants en jaune correspondent aux modules déjà existant sur le projet R3G qui vont nous être utiles par la suite. Les composants en gris correspondent à ceux qui ne vont pas être utiles pour notre démonstrateur. En particulier, il va s'agir du composant dédier à la création et la personnification du profil utilisateur avant la séance de démonstrateur. Cette fonctionnalité n'est plus pertinente

dans le cadre de la recherche de notre client. Les composants en vert correspondent aux nouveaux composants de notre démonstrateur ou à ceux déjà existant sur R3G qui vont être modifiés et améliorés afin de répondre aux nouvelles spécifications fonctionnelles. Nous verrons les détails structurels des améliorations qu'amènent ces nouvelles fonctionnalités.

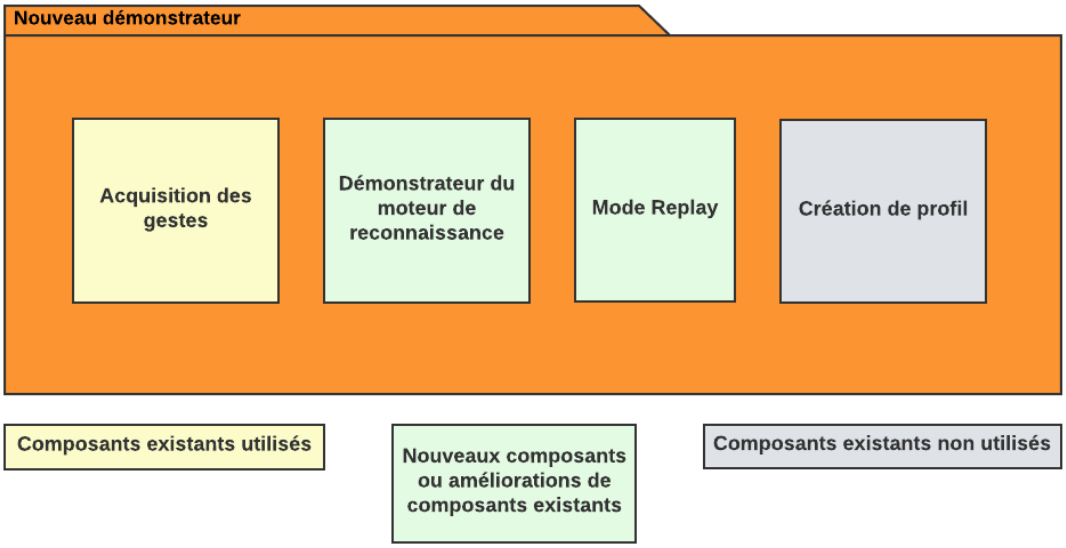

FIGURE 15 – Vision macroscopique de l'ensemble des modules d'analyse et d'acquisition des données en temps réel du nouveau démonstrateur

5.1.1 Architecture et comportement du nouveau démonstrateur

L'architecture suivante décrit les améliorations et modifications de conception de l'architecture du projet R3G. Celles-ci ont été réfléchies pour obtenir les fonctionnalités décrites aux points du cahier des charges reporté dans la table 6.

TABLE 6 – Cahier des charges

N°	Intitulé	Critère	Priorité
Affichage dynamique des gestes en compétition			4
N°6.1	Représentation des résultats	L'utilisateur encadrant peut visualiser les résultats de l'IA grâce à l'affichage des histogrammes correspondant aux différents gestes en compétition.	4
N°6.2	Seuil d'activation des histogrammes	Sélection entre les différents gestes de la base de données effectuée avant ou pendant l'affichage de l'histogramme grâce à un filtre.	4

Le démonstrateur a besoin de capturer un flux de gestes et de le communiquer au moteur de reconnaissance pour ensuite pour ensuite pouvoir afficher les résultats. Son architecture suit le modèle AMVCC défini précedemment 4. Le diagramme UML du démonstrateur est présenté en figure 17. Il est composé d'une classe DemonstrationApplication qui possède le modèle (DemonstrationModel), la vue (DemonstrationView) du module. Le modèle DemonstrationModel représente les données utilisées par l'application de reconnaissance de gestes 3D. Il contient les devices, les bases de données et les gestes disponibles. Il stocke également des informations sur le temps de départ, la fin de la reconnaissance de geste, les résultats de reconnaissance de gestes. La classe DemonstrationView correspond à la vue de l'application de reconnaissance de geste 3D et les interactions de l'utilisateur avec cette interface graphique. Elle gère également les actions déclenchées par l'utilisateur

La classe DemonstrationController, quant à elle, hérite de la classe abstraite AppliController. Cela lui permet de faire appel au DeviceManager grâce à la méthode InstanciateProviderLaunch() pour gérer la communication avec le moteur de reconnaissance. La classe RecoManager utilise la classe AIResult pour interpréter et afficher les résultats de confiance des classes de geste provenant du moteur de reconnaissance de geste. Cette communication avec le moteur de reconnaissance a été décrite plus en détail dans la partie 3.3.

Pour obtenir la fonctionalité d'affichage des histogrammes, nous avons décidé d'ajouter plusieurs méthodes à cette architecture. La méthode **onHistogramReceived(Map<string, double> scores)** ajouté à la classe DemonstrationController prend en paramètre un tableau associatif scores qui correspond aux différents identifiants de classe associés à leur score de confiance respectif calculé par le moteur de reconnaissance. Cette méthode va recevoir le tableau associatif provenant de la classe AIResult et va mettre à jour le tableau des scores du DemonstrationModel. Ensuite la mise à jour de l'affichage des histogrammes sera réalisée grâce à la méthode UpdateView() de DemonstrationView. La séquence de cette fonctionnalité est décrite dans la partie 4 du diagramme de séquence du démonstrateur présent sur la figure 18. De plus, La figure 16

décrit la maquette de l'interface graphique correspondant à l'implémentation de cette fonctionnalité.

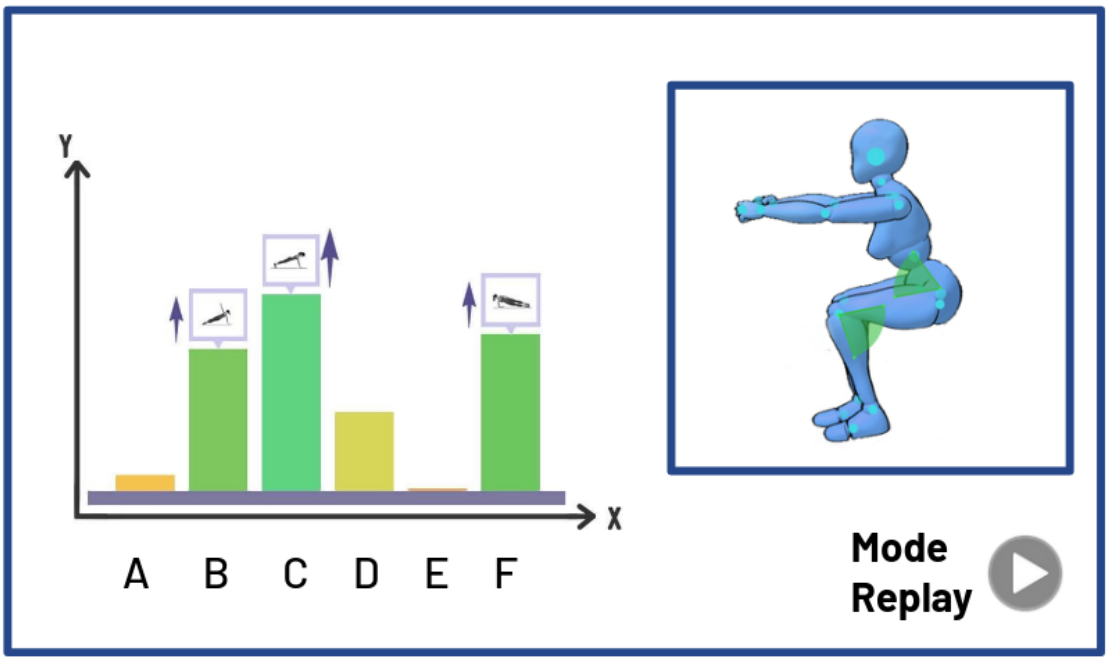

FIGURE 16 – Maquette de la nouvelle interface graphique du démonstrateur [13]

Concernant la fonctionnalité de sélection de filtre, nous souhaitons ajouter une méthode **onFilterSelected(List<string>)** dans la class DemonstrationController. La liste en paramètre de cette fonction correspond à la liste des classes qui ont été sélectionnées suite au filtre effectué par l'utilisateur. Une fois le filtre validé via la méthode **SelectFilter()**, de la classe DemonstrationView, il met à jour la liste scoreSelected de la classe Model et notifie ensuite le DemonstrationController. Le Controller va pouvoir récupérer la sélection du filtre dans la classe Model. De la même manière que la fonctionnalité précedente, la mise à jour de l'affichage des histogrammes filtrées sera réalisé grâce à la méthode UpdateView(). La séquence de cette fonctionnalité est décrite dans la partie 5 du diagramme de séquence du démonstrateur sur la figure 18.

À la fin de l'enregistrement, l'ensemble des données enregistrées lors d'une séance sont sauvegardées dans un répertoire spécifique aux séances de démonstration. Ces données sont sauvegardées au format InkML et permettent d'enrichir les bases de gestes pour l'entraînement du moteur de reconnaissance de geste.

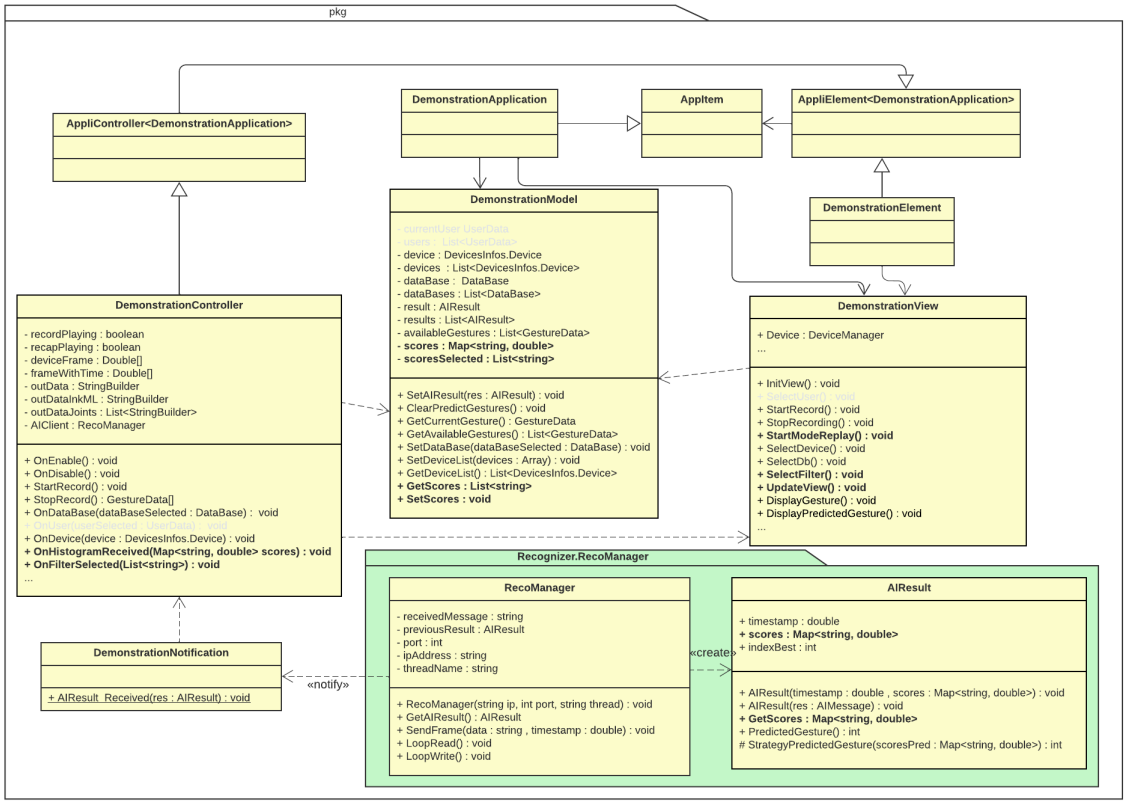

FIGURE 17 – Diagramme UML du nouveau démonstrateur

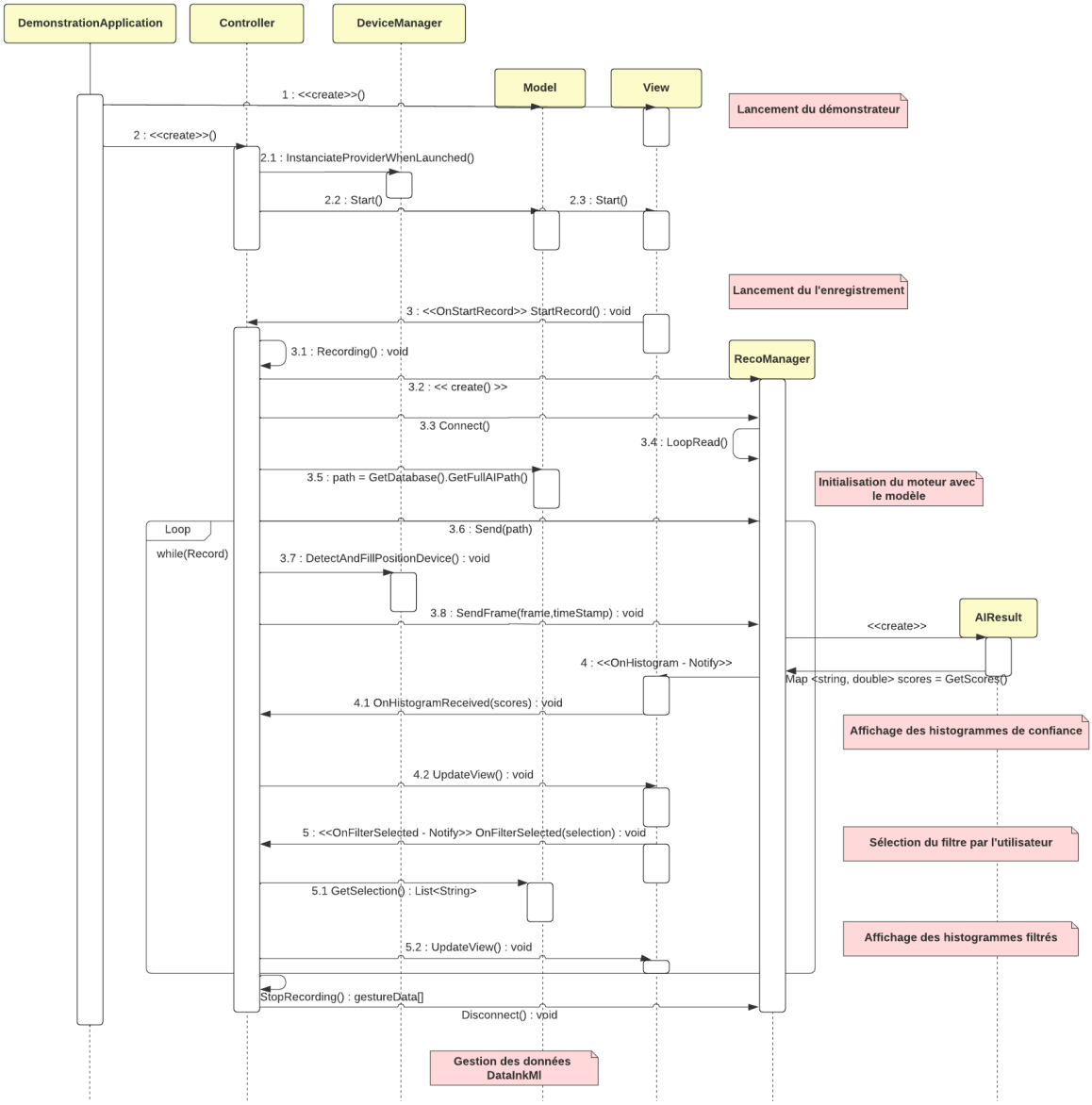

FIGURE 18 – Diagramme de séquence du démonstrateur

5.1.2 Conception de la généricité du démonstrateur

Dans cette partie, nous allons décrire notre réflexion autour de la fonctionnalité décrite en table 7 :

TABLE 7 – Cahier des charges

N°	Intitulé	Critère	Priorité
Généricité			3
N°8.1	Généricité du démonstrateur	Gestion de la variation du nombre de classes de gestes et des différentes natures de gestes.	3.2

Gestion de la variation du nombre de gestes Le démonstrateur doit pouvoir comprendre et analyser la variation du nombre de classes de geste. Pour cela, le démonstrateur devra adapter son affichage en fonction de ce nombre classe. Notre axe de travail se base sur l'idée de rendre les icônes de gestes de plus en plus petites pour faire rentrer le maximum de gestes sur les affichages des histogrammes. Pour faire varier la taille des icônes et des noms de classes, nous allons nous baser sur le concept de design web réactif en l'intégrant à l'interface Unity. Cela peut faire soit en utilisant l'application WebGL sur Unity ou en utilisant des FlexBox. Ces deux méthodes permettront à nos icônes de changer dynamiquement de tailles en fonction du nombre de classes. De plus, la fonctionnalité de sélection de filtre décrit précédemment permettrait déjà de répondre au cas d'un surplus de classe qui rendrait illisible l'affichage des résultats. Ce filtre peut permettre de réduire considérablement le nombre de classes affichées pour utilisateur.

Gestion des différentes natures de gestes Voici des exemples de base de données de gestes sur lesquelles le démonstrateur pourra être utilisé :

- La base Chalearn [11], qui est très statique, possède quasiment que des mouvements de bras ;
- Les bases OAD [14] et PKU-MMD [15] sont très dynamiques : le(s) squelette(s) bougent beaucoup dans toutes les directions.Sur PKU-MMD, il y a certaines séquences qui disposent de deux squelettes ;
- La base DHG [9] est une base où seulement les mains ont été captées ;
- Les bases ILGDB [17] et MTG [7] sont des bases de gestes 2D, qui ont été dessinés sur tablettes.

On observe que ces exemples possèdent des natures très différentes. Cependant ce sont tous des gestes qui devront être par notre démonstrateur. Pour gérer ce problème de généricité, nous pensons implémenter un calcul de boîte englobante du squelette 3D couplé à un calcul de marge basée sur la structure des données renvoyées par le capteur sur les premières de la captation de l'utilisateur. Le calcul de la boîte englobante peut etre implémenté simplement par la méthode CalculateBounds(Vector3[] positions, Matrix4x4 transform) proposé par la documentation Unity [2]. On ajoutera à ces calculs une connaissance a priori de la base choisie c'est-à-dire les données qui sont déjà enregistrées, sur lesquels le moteur de reconnaissance s'est déjà entrainé. Cette combinaison permettra de gérer des cas plus difficiles comme par exemple des gestes 2D où le geste n'est réduit qu'à un seul doigt.

6 Conclusion

Au terme de ce rapport de conception, Nous avons défini des diagrammes décrivant l'organisation des classes et des composants de chaque module ainsi que des diagrammes de séquence pour détailler des exemples de comportement. Ce travail nous permet d'avoir une vision précise sur le développement à effectuer.

Cette étape de conception était primordiale pour obtenir une bonne organisation du code du projet. Elle nous a permis de préciser certains points encore flous et d'appréhender les difficultés de la reprise de plusieurs projets existants. Notre projet possède à présent une base solide sur laquelle reposera le développement. La structure du projet étant déjà en place au terme de ce rapport, nous allons désormais nous concentrer sur la production des exercices afin de proposer le plus rapidement une application testable. Nous nous pencherons ensuite sur le démonstrateur.

7 Annexe

7.1 Retour sur la planification

La date des soutenances finales ayant été avancé de deux semaines, nous avons pris la décision d'abandonner les éléments concernant la version trois de "SIM" que nous avions présenté dans notre rapport de planification [12]. Le reste de planification reste inchangé. La table 8 présente les éléments abandonnés du cahier des charges :

Table 8 – Éléments abandonnés du cahier des charges

Refonte du mode auteur			6
N°5.1	Séance	L'utilisateur encadrant peut visualiser les différentes parties de la séance et configurer chacune d'entre elles.	6
N°5.2	Modification de la séance	L'utilisateur encadrant peut ajouter un exercice ou retirer un exercice à la liste.	6
N°5.3	Ajout d'un exercice	L'utilisateur encadrant peut assigner un exercice de la base à un exercice de la liste.	6
N°5.4	Paramétrage de l'exercice	L'utilisateur encadrant peut paramétrer le nombre de séries, de répétions, le temps de repos et des paramètres avancés en fonction du type d'exercice.	6

Pour rappel, il avait été décidé lors de la planification [12] que les points du cahier des charges présenté en table 9 ne seraient pas traités afin de pouvoir proposé une

planification réalisable :

TABLE 9 – Éléments abandonnés du cahier des charges

Ajout d'exercices dans la base			7
N°4.3	Vidéo	L'utilisateur encadrant peut se filmer pour définir enregistrer les mouvements d'un nouvel exercice.	7
N°4.2	Type d'exercice	L'utilisateur encadrant peut définir le type du nouvel exercice à ajouter à la base et paramétrer l'exercice en fonction de son type.	7
N°4.1	Nom	L'utilisateur encadrant peut définir un nom pour un nouvel exercice.	7
Analyse a posteriori des résultats fournis par l'IA			8
N°7.1	Visualisation d'un geste au ralenti	L'utilisateur encadrant peut visualiser les résultats de l'IA grâce à l'affichage des histogrammes correspondant aux différents gestes en compétition.	8
N°7.2	Navigation dans la séquence de gestes	L'utilisateur peut naviguer à travers la séquence de gestes et revenir sur un instant précis de l'analyse grâce à la timeline.	8

Références

[1] URL : https://en.wikipedia.org/wiki/Model-view-controller. (consulté : 6 février 2023).

[2] URL : https://docs.unity3d.com/ScriptReference/GeometryUtility.CalculateBounds.html.

[3] ANSES. *Inactivité physique et sédentarité chez les jeunes : l'Anses alerte les pouvoirs publics*. 2020. URL : https://www.anses.fr/fr/content/inactivit%C3%A9-physique-et-s%C3%A9dentarit%C3%A9-chez-les-jeunes-l'anses-alerte-les-pouvoirs-publics. (consulté : 4 Novembre 2022).

[4] Mehdi AOUICHA et al. *Rapport de Conception - Move on Progress*. Nov. 2021. URL : https://insarennesfr.sharepoint.com/sites/Projet4info22-23/Documents%20partages/Anciens%20Projets/RapportConception.pdf.

[5] Paul BERTHELOT et al. *Rapport de Conception - Recognition of 3D Gesture*. 2021. URL : https://insarennesfr.sharepoint.com/sites/Projet4info22-23/Documents%20partages/Anciens%20Projets/Concep_R3G_BabelMarie.pdf.

[6] Said Yacine Boulahia. « Reconnaissance en-ligne d'actions 3D par l'analyse des trajectoires du squelette humain ». Theses. INSA de Rennes, juill. 2018. URL : https://theses.hal.science/tel-01857262.

[7] Zhaoxin Chen et al. « Recognize multi-touch gestures by graph modeling and matching ». In : *17th Biennial Conference of the International Graphonomics Society*. Drawing, Handwriting Processing Analysis : New Advances and Challenges. International Graphonomics Society (IGS) and Université des Antilles (UA). Pointe-a-Pitre, Guadeloupe, juin 2015.

[8] Antoine Courmont. *Baromètre du numérique 2021 – Les chiffres des usages numériques en France*. Juill. 2021. URL : https://linc.cnil.fr/fr/barometre-du-numerique-2021-les-chiffres-des-usages-numeriques-en-france. (consulté : 4 Novembre 2022).

[9] Quentin De Smedt et al. « SHREC'17 Track : 3D Hand Gesture Recognition Using a Depth and Skeletal Dataset ». In : *3DOR - 10th Eurographics Workshop on 3D Object Retrieval*. Sous la dir. d'I. Pratikakis, F. Dupont et M. Ovsjanikov. Lyon, France, 2017, p. 1-6. DOI : 10.2312/3dor.20171049. URL : https://hal.archives-ouvertes.fr/hal-01563505.

[10] Aydınlı Emre et al. *Rapport de Conception - DERG3D*. 2018. URL : https://insarennesfr.sharepoint.com/sites/Projet4info22-23/Documents%20partages/Anciens%20Projets/_4info__Projet___DERG3D___Conception.pdf.

[11] Sergio Escalera et al. « Multi-modal Gesture Recognition Challenge 2013 : Dataset and Results ». In : *Proceedings of the 15th ACM on International Conference on Multimodal Interaction*. ICMI '13. ACM, 2013, p. 445-452.

[12] Cadot Firmin et al. *Rapport de Planification - Stay In Motion*. Nov. 2022. URL : https://insarennesfr.sharepoint.com/sites/Projet4info22-23/Documents%20partages/Anciens%20Projets/Planif_StayInMotion_ThierryRoger.pdf.

[13] Cadot Firmin et al. *Rapport de Pré-étude et de spécification fonctionnelle - Stay In Motion*. Nov. 2022. URL : https://insarennesfr.sharepoint.com/sites/Projet4info22-23/Documents%20partages/Anciens%20Projets/Stay_In_MotionSpec.pdf.

[14] Yanghao Li et al. « Online Human Action Detection Using Joint Classification-Regression Recurrent Neural Networks ». In : *Computer Vision – ECCV 2016*. Sous la dir. de Bastian Leibe et al. Cham : Springer International Publishing, 2016, p. 203-220. ISBN : 978-3-319-46478-7.

[15] Chunhui Liu et al. « PKU-MMD : A Large Scale Benchmark for Continuous Multi-Modal Human Action Understanding ». In : *CoRR* abs/1703.07475 (2017).

[16] William Mocaër, Eric Anquetil et Richard Kulpa. « Réseau Convolutif Spatio-Temporel 3D pour la Reconnaissance Précoce de Gestes Manuscrits Non-Segmentés ». In : *RFIAP 2022 - Congrès Reconnaissance des Formes, Image, Apprentissage et Perception*. Vannes, France, juill. 2022, p. 1-9. URL : https://hal.archives-ouvertes.fr/hal-03682604.

[17] N. Renau-Ferrer et al. « The ILGDB database of realistic pen-based gestural commands ». In : *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. 2012, p. 3741-3744.