SCIENCES SORBONNE UNIVERSITÉ

Année Universitaire 2019-2020
M1 SAR

Détection d'intention et du focus d'attention de l'utilisateur en Réalité Virtuelle

Étudiant :
Hugo Dorey
Thibault Roussel

Numéro étudiant :
3671230
3520128

Encadrant :
Sinan Haliyo

Juin 2020

Table des matières

Table des figures

Partie 1

Introduction

Ce rapport a pour but de présenter le travail réalisé dans le cadre du projet proposé par M.Haliyo intitulé : "Détection d'intention et du focus d'attention de l'utilisateur en Réalité Virtuelle".

La prédiction d'intention est un des défis majeurs à relever dans le cadre des interactions humains/robots. En effet, être capable de prédire les futures actions des différents objets et personnes entourant par exemple une voiture autonome est nécessaire si ce secteur d'activité veut pouvoir se développer, principalement pour des raisons de sécurité.

Dans le cadre de la réalité virtuelle, être capable de prédire les futures actions de l'utilisateur signifie pouvoir pré-calculer les éléments avec lesquels il risque d'interagir, et donc d'éviter une perte de l'immersion par un chargement. L'objectif de ce projet est d'étudier les différentes solutions techniques permettant de prévoir avec quels éléments d'une scène virtuelle notre utilisateur a des chances d'interagir.

Nous commencerons par définir le cadre de ce projet en définissant des objectifs à atteindre concernant notre travail, avant de réaliser un état de l'art des différentes technologies disponibles capables de répondre à nos besoins et ainsi choisir lesquelles utiliser. Nous finirons par présenter nos propositions de résolutions avant de conclure.

Partie 2

Cadre du projet

Ces dernières années ont vu un rapide essor de la réalité virtuelle dans de nombreux secteurs d'activité, en raison des récentes avancées dans ce domaine et de sa démocratisation auprès du grand public.

Néanmoins, malgré les progrès conséquents de cette technologie sur le plan de l'immersion visuelle et auditive, les interactions physiques possibles restent très limitées en comparaison des possibilités d'interactions d'un être humain avec son environnement. Une personne se déplaçant dans une scène virtuelle n'aura que peu ou pas de retour haptique sur ses actions, ce qui brisera son immersion.

Plusieurs solutions ont été proposées afin de résoudre ce problème, par exemple, l'utilisation d'une interface physique mobile permet à l'utilisateur d'interagir de manière réelle avec des objets virtuels. Parmi ces propositions, nous retrouvons le cas du dispositif proposée par l'ISIR sur la figure 2.1[1]. Dans ce dernier, le dispositif B, possédant différents éléments d'interactions C, se déplace autour de la structure A afin de se positionner pour être en raccord avec ce que l'utilisateur voit dans la scène virtuelle. Ce déplacement s'effectue grâce aux rails E et G permettant respectivement des déplacements suivant X et Y et au wagon D qui transporte le dispositif B.

Cependant, l'utilisation d'une telle interface soulève plusieurs problèmes, en particulier sa capacité à prédire à l'avance les intentions de l'utilisateur afin de se positionner au bon endroit au bon moment. Notre projet a pour but de proposer des solutions à ce problème avec l'aide d'une simulation informatique tournant sous le logiciel Unity. Cette simulation visible sur la figure 2.2 comporte un humain sous la forme d'un cône, ainsi que plusieurs objets d'intérêts dont nous connaissons la position. Le dispositif physique est quant à lui représenté par une boule roulant sur le sol (un proxy), dont l'objectif est d'atteindre sa cible (un des objets d'intérêts) avant l'utilisateur.

FIGURE 2.1 – Montage physique étudié [1]

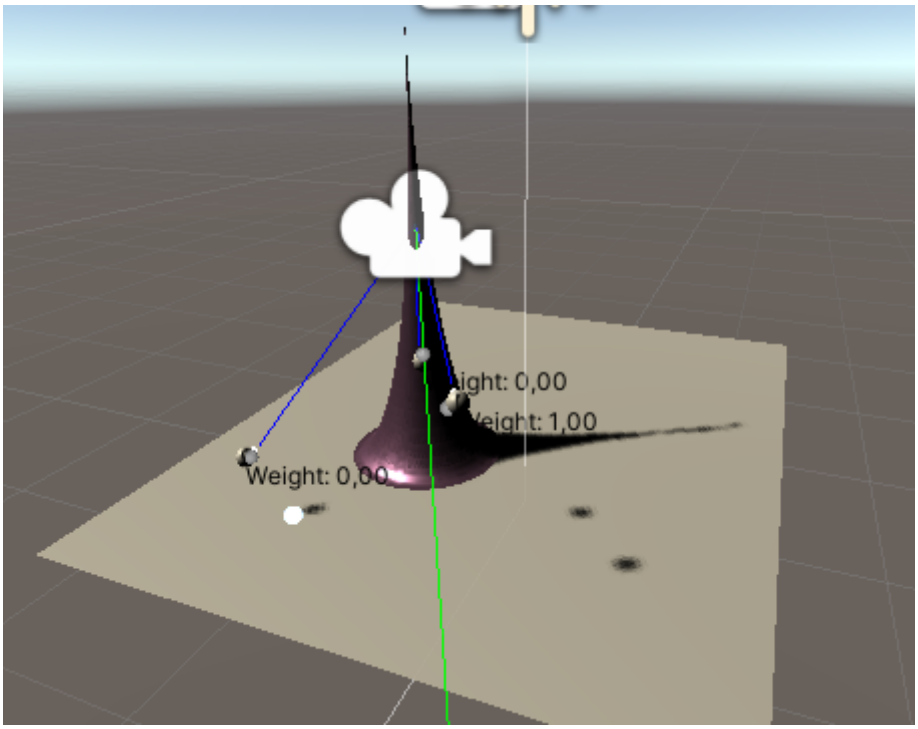

FIGURE 2.2 – Scène Unity représentant le montage étudié

Partie 3

État de l'art

Notre projet s'oriente autour de deux sujets majeurs : la réalité virtuelle et la prédiction d'intention. Dans cette partie, nous ferons un état de l'art de chaque sujet avant de déterminer les enjeux principaux du projet et les technologies à disposition pour le réaliser.

3.1 Réalité Virtuelle

La réalité virtuelle est une technologie permettant à un utilisateur de s'immerger dans un environnement artificiel et d'interagir avec des éléments de cet environnement[8]. L'immersion repose sur l'utilisation de stimulus visuels et auditifs afin de donner l'illusion à l'utilisateur de se trouver physiquement dans le monde virtuel.

Ces stimulus sont fournis par un casque constitué de deux écrans de résolution élevée diffusant une image par oeil, légèrement décalée afin de permettre une vision en 3D et ainsi simuler la vision humaine. Plusieurs capteurs sont présents dans le casque afin de percevoir les mouvements de la tête de l'utilisateur, et ainsi adapter le retour visuel au déplacement de la tête de l'utilisateur.

Les déplacements dans l'espace de l'utilisateur peuvent être simulés de plusieurs façons. Par exemple, le casque HTC-Vive [8] utilise des capteurs externes permettant de faire correspondre les mouvements et déplacements réels de l'utilisateur dans le monde virtuel. L'utilisateur peut donc se déplacer naturellement dans le monde virtuel, comme dans le monde réel. Néanmoins, ces déplacements sont limités dans une zone restreinte, en raison des contraintes physiques des capteurs et du casque. D'autres casques gèrent ces mouvements à l'aide de joysticks, ne permettant donc pas à l'utilisateur de se déplacer physiquement. Des appareils externes comme des tapis roulants sont en développement afin de résoudre le problème des limitations physiques des casques [8].

Il existe également plusieurs techniques qui donnent la possibilité à l'utilisateur de manipuler des éléments de la simulation virtuelle. La plupart des casques à l'heure actuelle utilisent des manettes conçues afin de pouvoir interagir de manière intuitive avec la simulation, tout en gérant les déplacements dans cette dernière. D'autres systèmes plus poussés sont en développement, comme notamment des gants pouvant transmettre un retour haptique à l'utilisateur, ou encore des interfaces physiques mouvantes[1].

Cependant, la réalité virtuelle souffre encore de plusieurs limitations : les casques les plus sophistiqués nécessitent pour la plupart d'être connectés physiquement à un ordinateur suffisamment puissant, limitant ainsi les déplacements physiques et obligeant l'utilisateur à se doter d'une installation coûteuse. De plus, de nombreuses personnes ne peuvent pas l'utiliser de manière prolongée sans souffrir de vertiges et de maux de tête en raison du motion sickness.

3.2 Prédiction d'intention

Différents algorithmes sont utilisés pour la prédiction d'intention. Dans cette partie, nous présenterons les deux technologies utilisées les plus intéressantes au vu de ce que nous souhaitons réaliser, à savoir de la prédiction d'intention en réalité virtuelle. Ainsi, nous présenterons deux technologies utilisant le machine learning : le HMM et les réseaux de neurones récurrents.

3.2.1 Prédiction par HMM (Hidden Markov Model)

Les HMM ont été plusieurs fois utilisés pour la reconnaissance d'intention d'un utilisateur, et ce pour différents types de tâches, comme la prédiction d'intention pour l'optimisation de trajectoire [9] ou la téléopération[10]. Il existe différentes approches de cette méthode, que nous allons décrire ici.

Double-layer Hidden Markov Model

On peut utiliser un Double-Layer HMM [2] pour la reconnaissance de l'intention et la prédiction du comportement d'un conducteur sur la route, à partir de données recueillies sur l'utilisateur et la voiture.

Ce modèle est composé de deux couches : la couche inférieure est constituée de plusieurs multi-dimensional Gaussian HMM (MGHMM), qui permettent de représenter plusieurs types de réactions du conducteur à court terme, dans un seul environnement de travail. La couche supérieure, quant à elle, est composée de plusieurs multi-dimensional discrete HMM (MDHMM), qui permettent d'indiquer les intentions à long terme de l'utilisateur dans un environnement de travail combinant plusieurs paramètres.

La couche inférieure est connectée à la supérieure par ses résultats inférentiels. L'analyse est organisée selon le principe suivant :

Pour faciliter la reconnaissance, la conduite à long terme est divisée en plusieurs chaînes de réactions simples à court terme. La couche inférieure va donc reconnaître les différents comportements de l'utilisateur à court terme, puis établira les modèles de conduite à court terme les plus probables pour chaque paramètre étudié. Les résultats seront ensuite transférés à la couche supérieure, qui déterminera la conduite à long terme la plus probable.

Cette configuration est représentée plus en détails dans le schéma de la figure 3.1.

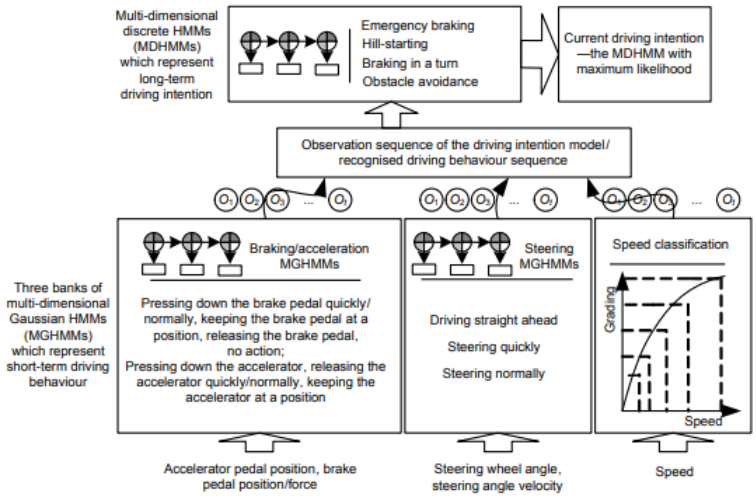

FIGURE 3.1 – Schéma représentant la configuration du HMM à deux couches[2]

Layered HMM (LHMM)

L'utilisation de Layered HMM a déjà été étudiée [3] pour la reconnaissance d'intention. Un LHMM est constitué de plusieurs HMM tournant en parallèle à n'importe quel niveau de la hiérarchie de cette méthode. Chaque HMM correspond à un concept précis. Cette méthode utilise deux types de HMM : les HMM dit gestem-level servent à reconnaître certaines séquences de mouvement simples, tandis que les task-level HMM servent à reconnaître la tache complète. Un LHMM est donc constitué de N niveaux de HMM où le HMM au niveau N+1 utilisent les informations obtenues à partir des HMM du niveau N.

On peut considérer cette méthode comme une généralisation du Double-Layer HMM.

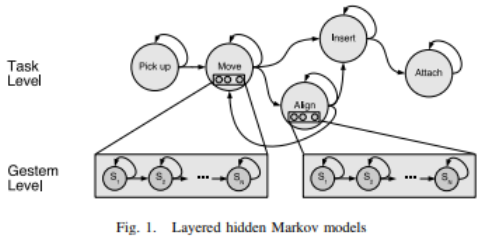

Fig. 1. Layered hidden Markov models

FIGURE 3.2 – Schéma représentant la configuration du LHMM[3]

Cette méthode présente plusieurs avantages par rapport à une seule grande couche de HMM où tous les modèles et concepts sont rassemblés : elle souffre moins du sur-apprentissage, les quantités de données d'entraînement nécessaires sont significativement plus faibles à performance finale égale. Enfin, les couches inférieures peuvent être facilement réentrainées sans avoir à toucher aux couches supérieures.

3.2.2 Prédiction par RNN (Recurrent neural network)

Déjà étudié pour les véhicules autonomes, ce modèle de réseau de neurones prend en compte dans son modèle décisionnel des patterns temporels. Pour dire de manière sim-

plifié, le réseau de neurones RNN est capable de prendre en compte le contexte d'une information (ce qui est arrivé avant cette dernière) pour en prédire la suite. Le principal problème de ce réseau tient sur la rétention des dépendances à long terme : les réseaux de neurones RNN classique ont du mal à prendre en compte les informations dans des longues séquences de données [4].

Heureusement, une dérivation de ce réseau fut créée : le LSTM. abréviation de Long short-term memory, la particularité de ce réseau est sa capacité à retenir les dépendances d'informations sur le long terme[4]. Il est notamment adapté à la prédiction de mouvement grâce à sa structure.

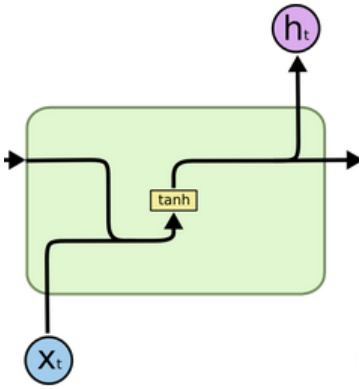

FIGURE 3.3 – Structure basique d'une cellule du réseau de neurone RNN [4]

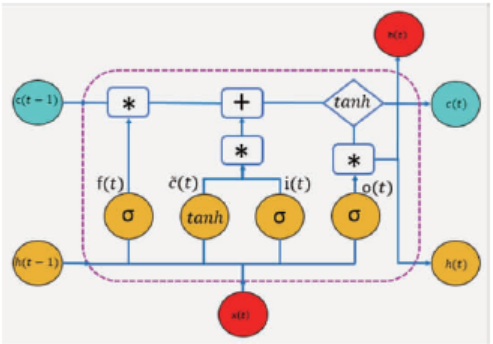

FIGURE 3.4 – Structure basique d'une cellule du réseau de neurone LSTM [5]

Comme nous pouvons le voir sur les figures 3.3 et 3.4, la structure des cellules en LSTM possèdent des couches supplémentaires comparées aux modèles RNN classiques. Ces couches ajoutées permettent un contrôle sur les informations entrantes dans la cellule (alors que dans le cas d'un RNN classique, nous sommes sur un "tout ou rien"), ce qui permet un meilleur affinement des données à prendre en compte, et donc une meilleure précision [11] [12].

3.2.3 Conclusion sur le choix des technologies à utiliser

Maintenant que nous avons vu les principales technologies utilisées pour la détection d'intention, nous devons choisir lesquelles de ces derniers conviennent à notre projet.

Les structures en réseau de neurones récurrents sont plus performantes dans les études de comportements au fil du temps [12]. De plus, cette technologie possède des toolkits sous Unity, ce qui simplifierait son implémentation dans le projet existant. Notre choix s'est donc porté sur cette technologie.

Partie 4

Travail réalisé

Dans cette partie, nous proposerons différentes solutions permettant de prédire les actions d'un utilisateur. Pour cela, nous utiliserons des jeux de données fourni par Mme Bouzbib.

Ces solutions utilisent les différentes données auxquelles nous avons accès lors de la simulation, à savoir : la position de l'utilisateur, les positions des différents objets d'intention, ainsi que le vecteur représentant la direction du regard de l'utilisateur. Nous discuterons de leurs avantages et de leurs limites.

4.1 Approche probabiliste

Nous avons réalisé dans un premier temps une approche probabiliste afin de résoudre notre problème. Définir en fonction de critères choisis une probabilité de réaliser une action permet en première approche d'appréhender certaines subtilités qui auraient pu ne pas être détectées autrement.

Pour cela, nous avons mis en place plusieurs algorithmes de détection d'intention se basant sur un système de scores, déterminés à partir des informations auxquelles nous avons accès.

4.1.1 Algorithme de détection simple

Notre premier algorithme repose sur un principe simple. A partir du vecteur représentant la vision de l'utilisateur et des positions des différents objets, nous déterminons les angles θ_i représentant la position de l'objet dans le champ de vision de l'utilisateur, puis nous déterminons des scores pour chaque objet en fonction de la valeur de ces angles.

Nous établissons les scores de la manière suivante : nous attribuons un score pour chaque zone du champ de vision humain entre 0° et 60°, car comme nous pouvons le voir sur la figure 4.2, il s'agit de la zone du focus de l'attention chez l'être humain. Ainsi, si un des angles déterminés précédemment est compris entre 60° et 30°, le premier score de cet objet sera égal à 0.2. De 30° à 20°, il est égal à 0.4 ; entre 20° et 10° : 0.6 ; entre 10°

et 5° : 0.8 ; et enfin le score atteint sa valeur maximale de 1 entre 5° et 0°.

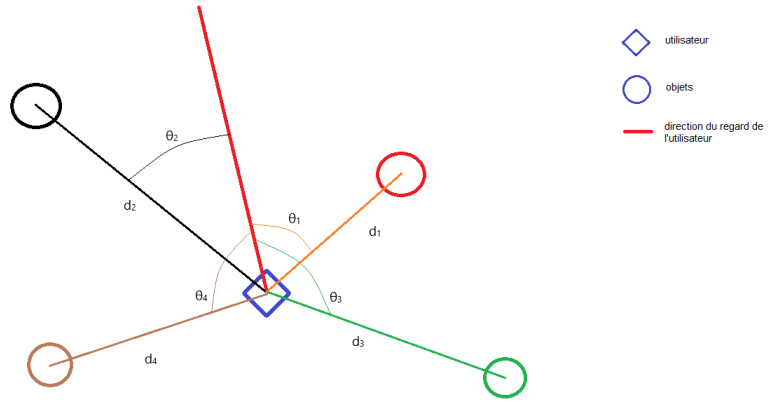

FIGURE 4.1 – Représentation du fonctionnement de notre premier algorithme.

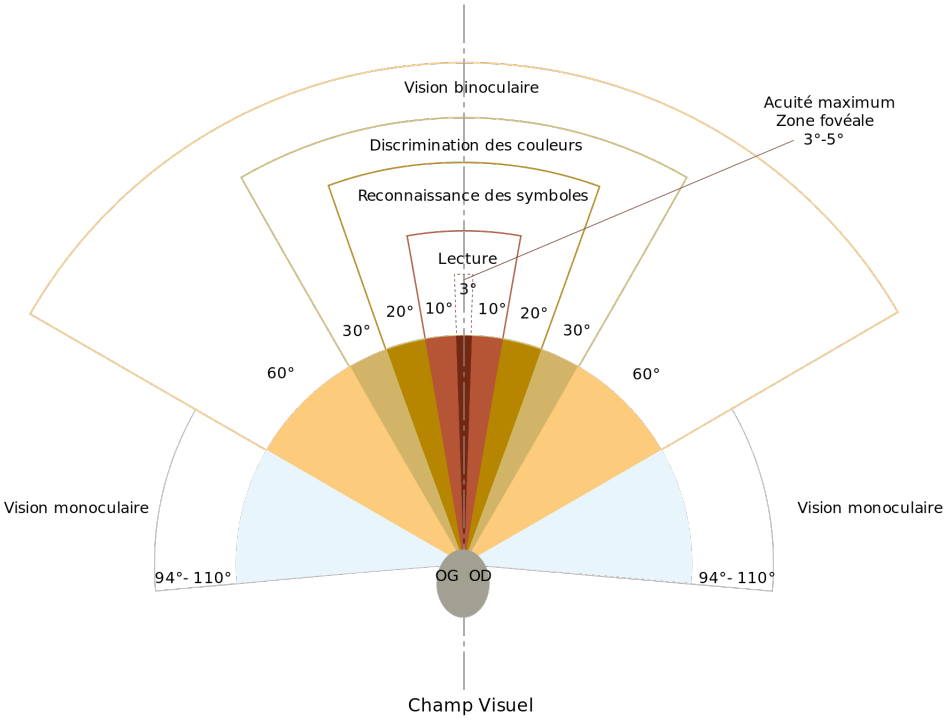

FIGURE 4.2 – Représentation de la vision Humaine [6]

Dans un second temps, nous déterminons un deuxième score dépendant de la distance d_i entre l'utilisateur et chaque objet. Il est calculé de la manière suivante :

$$S_{distance} = \frac{1}{d_i + 1} \tag{4.1}$$

Après avoir calculé les deux scores de chaque objet, nous obtenons le score final en réalisant la moyenne entre les deux précédents. Afin d'affiner notre modèle, nous appliquons des coefficients à chacun de nos scores, afin de représenter leur importance respective. Nous avons essayé plusieurs coefficients afin d'obtenir le meilleur résultat possible. la formule utilisée est donc :

$$S_{final} = \frac{k_1 S_{distance} + k_2 S_{angle}}{k_1 + k_2} \tag{4.2}$$

Finalement, l'objet ayant le score final le plus élevé est donc reconnu comme celui ayant le plus de chance d'interagir avec l'utilisateur.
Nous avons testé cet algorithme sur une base de données comprenant les coordonnées d'un utilisateur et de trois cibles, et l'évolution de l'orientation de son regard et de sa position au cours du temps. Notre objectif était que l'algorithme reconnaisse la bonne cible parmi les trois possibles, que nous avons choisie au préalable. La simulation tourne sur un programme codé en Python. Nous avons également cherché à optimiser les coefficients k_1 et k_2.

La cible 0 est la cible désirée durant toute la durée de la simulation.

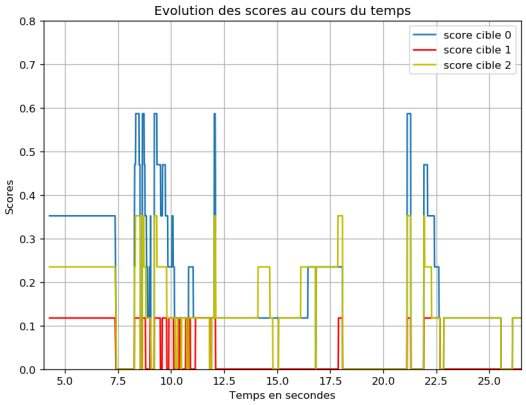

FIGURE 4.3 – Application de l'algorithme de détection simple, pour $k_1 = 1$ et $k_2 = 2$

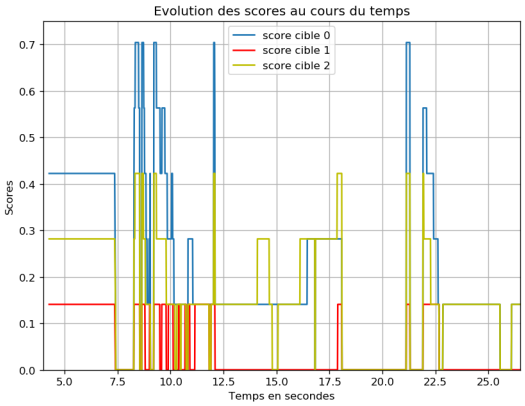

FIGURE 4.4 – Application de l'algorithme de détection simple, pour $k_1 = 1$ et $k_2 = 4$

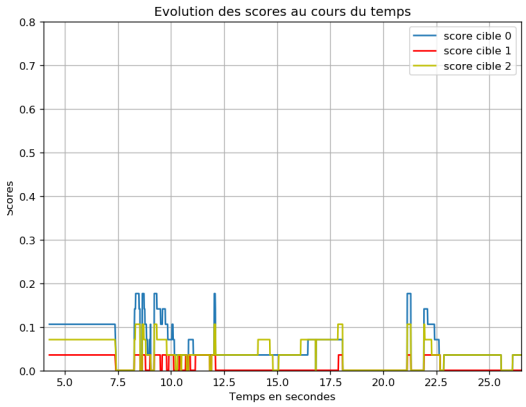

FIGURE 4.5 – Application de l'algorithme de détection simple, pour $k_1 = 4$ et $k_2 = 1$

Nous pouvons constater que l'algorithme n'arrive pas à prédire la bonne cible sur la totalité de la simulation. En effet, à plusieurs moments, l'algorithme identifie la mauvaise cible ou n'arrive pas à les différencier, notamment entre 12.5 secondes et 14 secondes.

Ces problèmes de reconnaissance sont provoqués par plusieurs facteurs : cette méthode présente une forte dépendance aux déplacements de l'utilisateur ainsi qu'au changement de l'orientation de son regard. De plus, des cibles situées à des emplacements proches se verront attribuer des scores très proches, ce qui empêche une bonne reconnaissance de l'intention de l'utilisateur. Nous pouvons donc en conclure que cet algorithme ne discrimine pas suffisamment les cibles potentielles.

Nous pouvons également remarquer, concernant l'optimisation des coefficients k_1 et k_2, qu'une trop forte influence de la distance par rapport à la vision provoque une forte diminution de l'amplitude des courbes 4.5, ce qui se traduit par des différences de scores faibles, et donc des résultats finaux peu fiables.

Nous avons pensé à des améliorations supplémentaires pour cet algorithme, en lui donnant la possibilité d'augmenter ou de diminuer des scores en fonction de la taille et de la couleur de chaque objet. En effet, un utilisateur découvrant une scène sera attiré différemment par les objets de cette dernière selon s'ils lui attirent plus ou moins le regard : entre un objet de 30 cm et un de 3, c'est souvent le premier que l'on remarque en premier et un objet orange sur fond vert est plus visible qu'un vert. Nous n'avons malheureusement pas pu tester ces améliorations, car ces informations n'étaient pas présentes dans notre base de données.

4.1.2 Algorithme du Voisin

Dans ce second algorithme, nous avons appliqué une autre méthode afin de déterminer la future interaction de l'utilisateur, tout en essayant d'éviter le problème de l'algorithme précédent, concernant la reconnaissance de cibles proches les unes des autres. Nous calculons les angles θ_i entre la vue de l'utilisateur et chaque objet, mais nous ne préoccupons cette fois-ci uniquement d'une cible dont l'angle se situe entre 10° et 0°. Cette restriction très forte du champ de vision nous permet d'éliminer un maximum de cibles lors de la première phase de l'algorithme, notre objectif étant de n'en conserver qu'une seule.

Si un objet se situe dans cette zone du champ de vision, le score maximum lui est automatiquement attribué. Les scores des autres objets sont ensuite déterminés en fonction de la distance d_i entre eux et l'objet ayant le score maximum en suivant la formule 4.1 utilisée dans le précédent algorithme.

Cette technique se base sur le fait qu'un utilisateur aura tendance à manipuler des objets proches de celui qui l'intéresse, et ainsi nous permet de reconnaître facilement des zones d'intérêts dans une simulation. Une représentation du fonctionnement de cet algorithme est visible sur la figure 4.6.

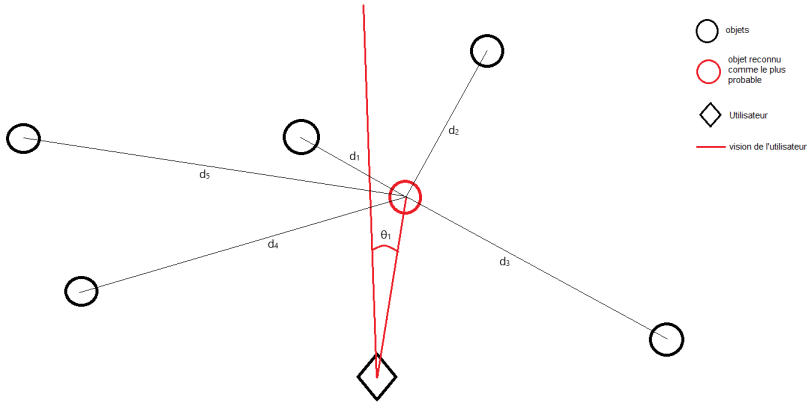

FIGURE 4.6 – Représentation du fonctionnement de notre deuxième algorithme.

Pour tester cet algorithme, nous utilisons un script Python et les données précédemment utilisées pour le premier algorithme. Les résultats sont visibles sur la figure 4.7(la cible désirée est la cible 0).

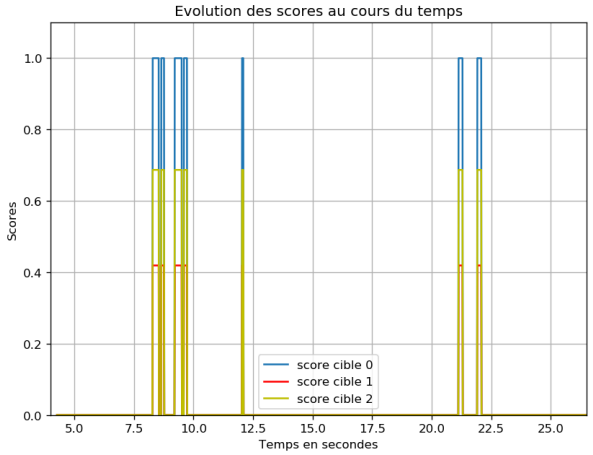

FIGURE 4.7 – Évolution des scores en simulation avec l'algorithme du voisin

Nous observons que l'algorithme n'effectue pas de mauvaises prédictions, contraire-

ment à la méthode précédente. L'algorithme arrive donc à prévoir correctement même pour des objets se situant dans des positions proches. Cependant, ce dernier ne possède pas de "mémoire" et ses conditions de reconnaissance sont très strictes, ce qui explique les passages à vide observés : l'utilisateur ne devait probablement pas regarder la cible à ces instants-là.

Néanmoins, les deux algorithmes que nous venons de décrire ne prennent pas en compte la notion de temps : en effet, un objet observé pendant 0,5 seconde sera considéré de la même manière qu'un autre observé pendant 10 secondes. Pour résoudre ce problème ainsi que les autres que nous avons rencontrés, nous proposons un dernier algorithme.

4.1.3 Algorithme de reconnaissance par activation

Pour cet algorithme, nous avons utilisé une méthode plus complexe que pour les deux précédents algorithmes, reposant sur une prise en compte du temps, et un principe d'activation.

Cette méthode considère qu'un utilisateur aura tendance à regarder de manière prolongée l'élément qui l'intéresse, tout en conservant la possibilité de réaliser une discrimination et une reconnaissance rapide des éléments à proximité.

Le programme opère de la façon suivante : comme pour l'algorithme du plus proche voisin, nous nous préoccupons d'un objet uniquement s'il se situe dans la zone la plus centrale du champ de vision. Néanmoins, cette fois-ci, nous incorporons une notion de temps dans notre algorithme, absente dans les deux autres.

En effet, si un objet se situe dans la zone du champ de vision qui nous intéresse, ce dernier ne se verra pas automatiquement attribué le score maximum. Nous appellerons cet objet O_1. Si O_1 reste dans le champ de vision pendant une période prolongée, son score augmentera au cours du temps jusqu'à atteindre la valeur maximale. Le score augmente de la façon suivante : si l'utilisateur reste fixe, le score augmentera d'une valeur fixe F_1 à chaque itération (toutes les 1/100èmes de secondes dans notre base de données) ; si l'utilisateur se rapproche de l'objet, tel que la distance entre lui et O_1 au temps t+1 est inférieure aux 2/3 de la distance au temps t, alors le score augmentera plus fortement.

Les scores des autres objets à proximité d' O_1 sont déterminés de manière similaire à celle de l'algorithme du voisin. Ainsi, le score des autres objets évolue conjointement avec celui d' O_1. Ces scores sont conservés en mémoire, permettant ainsi d'éviter les périodes de vide de l'algorithme précédent.

$$S = \frac{S_1}{d_i + 1} \tag{4.3}$$

Avec S_1 le score de O_1, à l'instant t, et S le score de l'objet testé.

Si, au cours de la simulation, le score d'un objet dépasse la valeur seuil de 0.5, cet objet est "activé". Cela signifie que si l'utilisateur change la direction de son regard, et qu'une cible activée se situe dans la zone qui nous intéresse, elle est automatiquement

reconnue comme nouvelle cible la plus probable et obtient immédiatement le score maximum. Les scores des autres objets sont instantanément mis à jour selon la formule 4.3 ci-dessus. Cette technique nous permet ainsi d'opérer une transition rapide entre des objets proches, tout en éliminant facilement les éléments les moins probables.

Nous avons, comme pour les deux autres méthodes, essayé cet algorithme sur une base de données afin de vérifier son fonctionnement et comparer son efficacité avec celle des autres algorithmes. Nous avons également cherché à optimiser F_1 au cours du temps, afin d'avoir le meilleur compromis entre la rapidité de l'algorithme et le phénomène physique. Pour cela, nous avons essayé les valeurs suivantes dans plusieurs tests : F_1= 0.01, 0.0075 et 0.005. Enfin, un deuxième test sur une base de données spéciales fut effectué afin de vérifier le bon fonctionnement du principe d'activation.
Lors des premiers essais, la cible désirée était la cible 0.

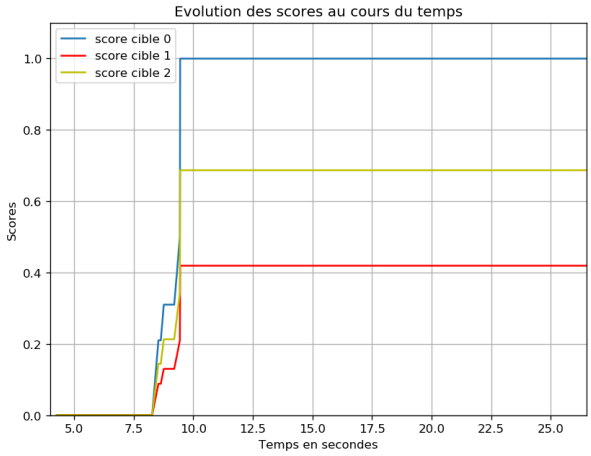

FIGURE 4.8 – Évolution des scores en simulation avec l'algorithme par activation. F_1=0.01

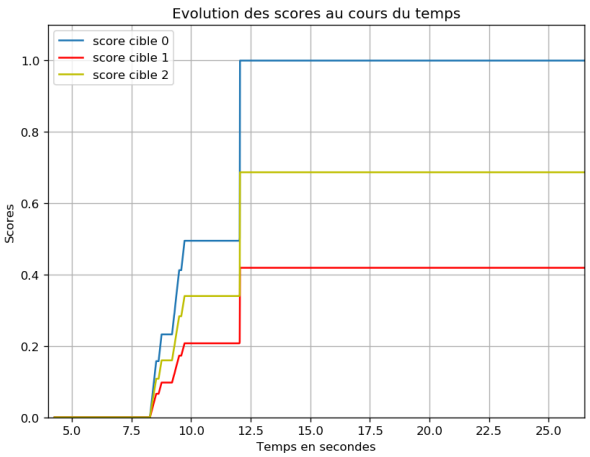

FIGURE 4.9 – Évolution des scores en simulation avec l'algorithme par activation. F_1=0.0075

Nous pouvons constater que l'algorithme reconnaît bien la cible désirée, et ce à chaque instant, contrairement à nos autres algorithmes. Un délai est néanmoins présent, la re-

connaissance de l'intention de l'utilisateur n'est donc pas immédiate.

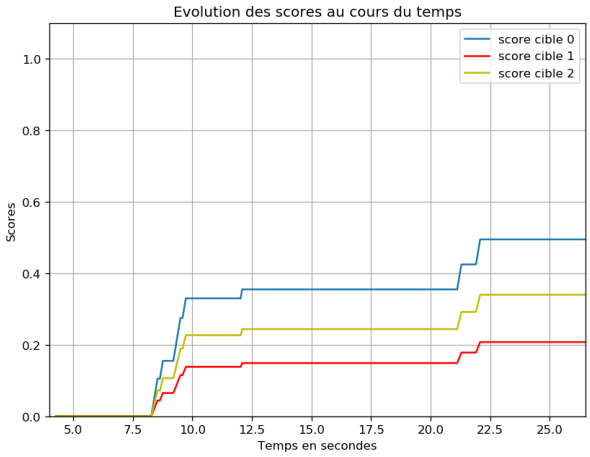

FIGURE 4.10 – Évolution des scores en simulation avec l'algorithme par activation. F_1=0.005

Dans notre seconde série de tests, nous avons voulu vérifier que notre programme était bien capable de rapidement s'adapter en cas de changement soudain de cible. Nous avons donc repris la base de données de nos autres tests, mais en la modifiant pour que la cible désirée entre 15 et 21 secondes soit la cible 2. Après ce passage, la cible choisie est de nouveau la cible 0.

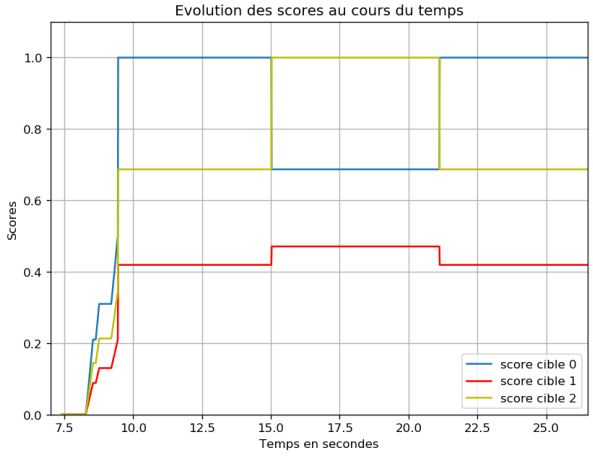

FIGURE 4.11 – Évolution des scores en simulation avec l'algorithme par activation sur la base spéciale. F_1=0.01

Nous constatons bien sur les graphiques 4.11, 4.12 et 4.13 que l'algorithme a réussi à s'adapter rapidement au changement de cible, et nous pouvons en conclure que le système d'activation fonctionne bien, en tout cas dans des applications simples. En terme de coefficient, nous pensons que mettre $F_1 = 0.0075$ semble être un bon compromis entre réactivité et précision en ce qui concerne la recherche de cible.

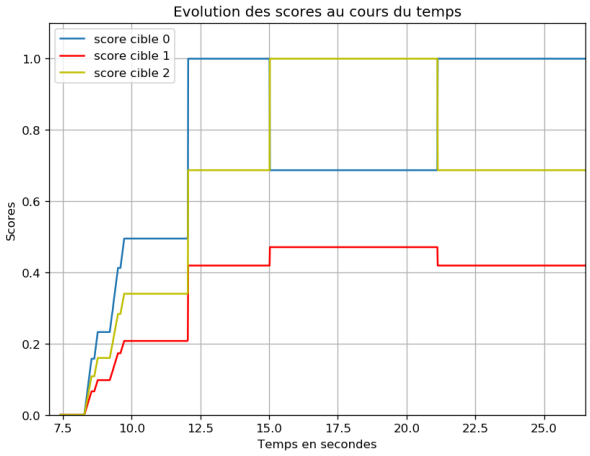

FIGURE 4.12 – Évolution des scores en simulation avec l'algorithme par activation sur la base spéciale. F_1=0.0075

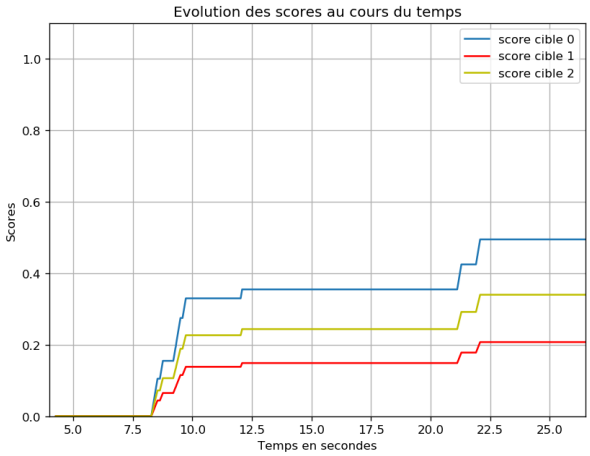

FIGURE 4.13 – Évolution des scores en simulation avec l'algorithme par activation sur la base spéciale. F_1=0.005

4.2 Approche à l'aide des réseaux de neurones

Comme montré précédemment, l'approche probabiliste possède ces limitations, et nous avons donc décidé de nous intéresser à une implémentation d'un réseau de neurones sous Unity. Nous commencerons par présenter la technologie utilisée avant de réaliser un compte-rendu de ce que nous sommes parvenu à mettre en place.

4.2.1 Technologie utilisée

Pour implémenter notre réseau de neurones, nous utilisons le toolkit ML-Agents, qui permet de simplifier le développement d'intelligence artificielle utilisant le machine learning sous Unity. Il s'agit d'un toolkit contenant une API python et autorisant l'utilisation des librairies existantes sous Unity, et qui nous permet de créer des "agents" qui peuvent apprendre à accomplir des objectifs sous contraintes à l'aide des réseaux de neurones.

Pour définir les objectifs à accomplir, un système de récompense et de malus est utilisé par les agents. Par exemple, prenons la figure 4.14 : nous avons un agent représenté par la balle qui doit rejoindre sa cible représentée par le cube en roulant sur une plateforme. Nous pourrions entraîner cet agent en lui disant que réduire sa distance avec le cube entraîne une récompense de 1, que réduire cette distance à 0 ou 0.1 vaut 10, que tomber de la plateforme équivaut à une pénalité de -10... La pertinence des critères que nous choisissons permettra une meilleure efficacité lors de l'apprentissage.

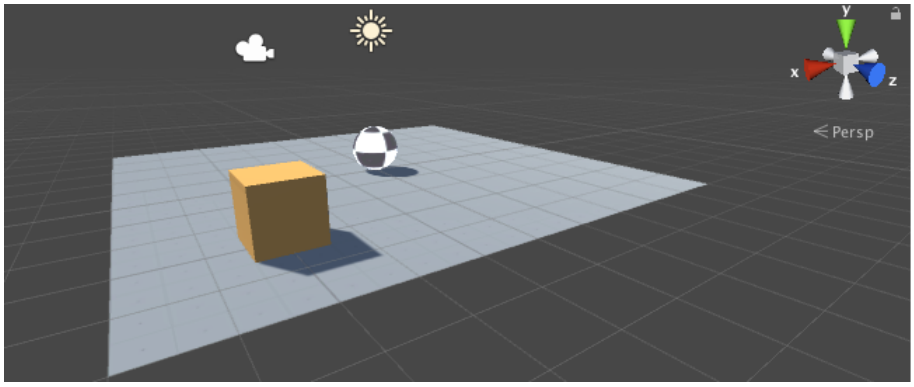

FIGURE 4.14 – Exemple de projet par ML-agent [7]

Pour prendre des décisions, l'agent doit posséder en entrée différentes données que nous pouvons choisir, telle que la position d'un objet, sa vitesse, sa distance par rapport à l'agent... Il est ainsi possible de contrôler les données entrant dans ce dernier et donc son apprentissage. Trois méthodes d'observation peuvent être distinguées :

— l'observation visuelle : consiste à donner à l'agent une "caméra" lui permettant de voir son environnement. Cette méthode est principalement utilisée pour le réseau de neurones convolutionnels (CNN). Un exemple de ce que voit l'agent est disponible sur la figure 4.15.

— l'observation par vecteur : principalement utilisée pour les réseaux de neurones récurrents (RNN), elle se base sur l'observation des coordonnées des objets et de leurs vélocités en x, y et z. A noter que pour que cette approche soit efficace, les observations doivent être normalisées.

— l'observation à l'aide de Raycast : consiste à envoyer des "rayons" autour de l'agent lui permettant de détecter des objets s'ils les rencontrent. Cette méthode est illustrée dans la figure 4.16 et est principalement utilisée lorsqu'un agent doit posséder une forte dépendance à son environnement, comme par exemple un agent devant faire une course de karts.

Cela n'est cependant pas suffisant pour l'apprentissage : chaque agent doit avoir un comportement qui, en fonction des informations reçues, va déterminer les actions à réaliser. Il s'agit donc du "cerveau" de notre agent. Ce dernier permet de facilement tester les différents modèles d'apprentissage que nous avons implémentés en les réutilisant sans réapprentissage.

Sur la figure 4.17 est représenté la structure classique d'un programme sous Unity utilisant ML-Agent.

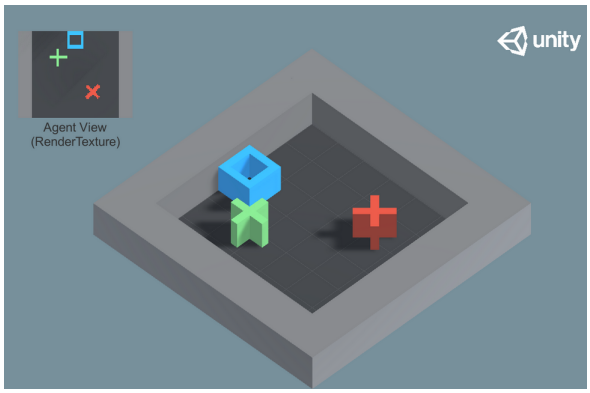

FIGURE 4.15 – Représentation de la vue d'un agent à l'aide des capteurs visuels [7]

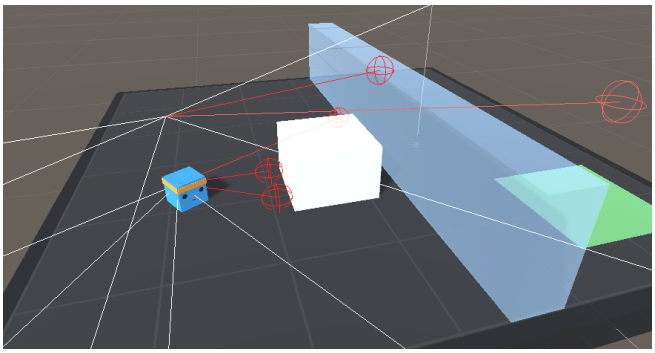

FIGURE 4.16 – Représentation de la vue d'un agent à l'aide de la technologie Raycast [7]

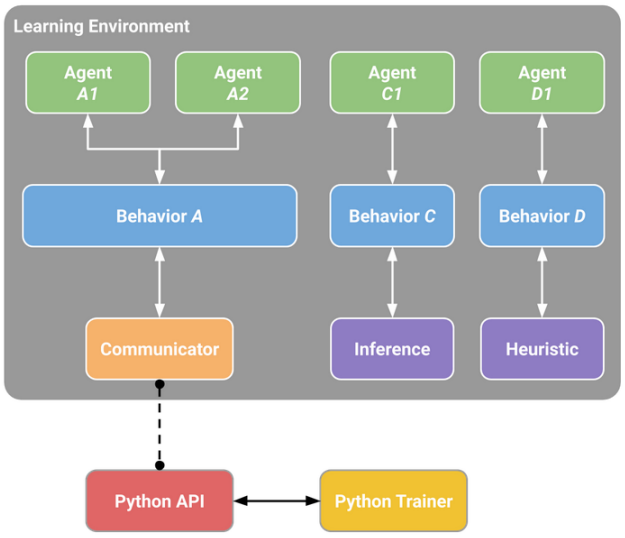

FIGURE 4.17 – Structure d'un programme utilisant ML-agent [7]

4.2.2 Implémentation de la technologie

Maintenant que nous avons expliqué le principe derrière ML-agent, nous allons voir comment ce toolkit a été utilisé dans notre projet pour implémenter un réseau de neurones.

L'agent sera utilisé dans une sphère qui pourra rouler sur le sol. Son but sera de trouver quelle cible est active avant que l'utilisateur ne la touche. Pour cela, l'agent observe les positions des différentes cibles, la position de l'utilisateur et de ses mains, sa position et sa vélocité. Le regard de l'utilisateur aurait été une donnée utile à exploiter mais nous ne savons pas comment la récupérer à l'aide des commandes disponibles dans le cas de l'utilisation d'un réseau de neurones récurrents.

Notre agent sera récompensé lorsqu'il atteindra les coordonnées x et z d'une cible et que cette dernière doit être ramassée par l'utilisateur. Si cette dernière condition n'est pas respectée, l'agent recevra un malus, de même s'il s'éloigne trop des cibles (représenté par le fait que l'agent tombe du plateau sur lequel il roule).

La figure 4.18 représente l'évolution de la récompense moyenne de l'agent au fur et à

mesure de son apprentissage. Comme nous pouvons le constater, cette dernière n'évolue pas comme nous le désirions : l'agent n'apprend pas de ses erreurs et continue de tomber du plateau sans rejoindre la position des cibles. Plusieurs raisons peuvent expliquer ces résultats :

— un mauvais choix dans les récompenses attribuées à l'agent, si bien que ce dernier ne sait pas quoi apprendre.
— une mauvaise attribution des données en entrée, nous n'avons par exemple pas pu prendre en compte la direction du regard de l'utilisateur ou la position de ses mains, ce qui peut jouer sur l'apprentissage.
— un mauvais paramétrage de l'apprentissage, notamment en ce qui concerne les paramètres comme le nombre d'epochs, le learning rate...

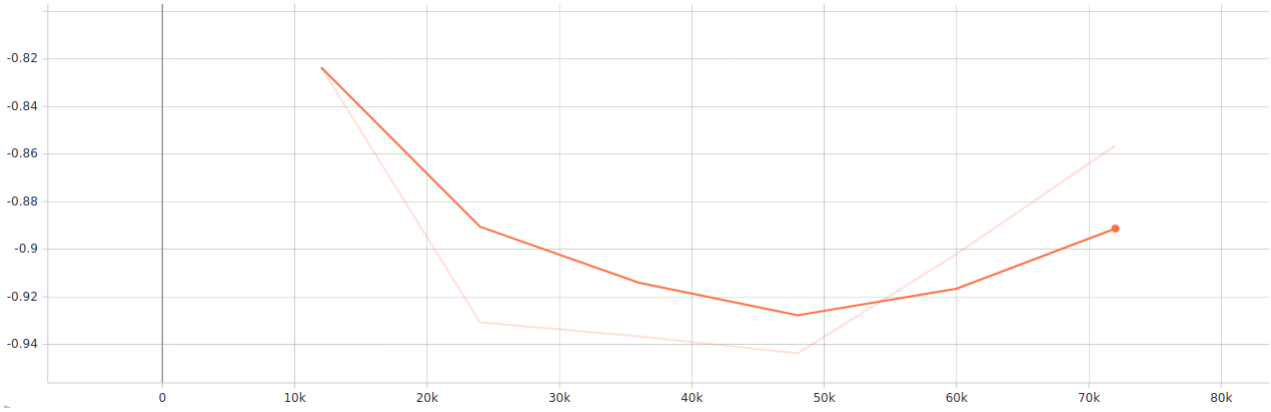

FIGURE 4.18 – Évolution de la récompense moyenne de l'agent au fil du temps

Malgré tout, même si notre agent marchait parfaitement, il posséderait encore un défaut non négligeable : il est en sous-apprentissage. En effet, nous n'avons que trois ou quatre situations disponibles dans les scripts, ce qui n'est pas assez si l'on souhaite réaliser quelque chose de robuste. Pour remédier à ce problème, d'autres tests sont à réaliser afin d'enregistrer de nouvelles situations. Cette méthode est coûteuse en temps car pour avoir un programme suffisamment robuste, cela demanderait un nombre de tests conséquent.

Nous avons donc réfléchi à une alternative qui pourrait remplacer la méthode précédente : une génération de tests par machine learning. ML-agent permet de générer des agents évoluant en compétition. Notre solution se baserait sur cela pour créer deux agents : l'un devant atteindre un objectif fixé et l'autre devant deviner quel objectif le premier doit atteindre pour ensuite arriver avant lui. Ce premier agent ne disposera pour se repérer que de la technologie Raycast dans un angle limité qui simulera la vision humaine et ne sera récompensé que s'il atteint son objectif avant son agent rival. Avec un script générant de manière aléatoire la position des cibles à atteindre, cela permettrait de générer un grand nombre de test rapidement. Cependant, nous n'avons pas implémenté cette méthode et avons juste réfléchie de manière théorique à son fonctionnement.

Partie 5

Conclusion

En conclusion, les tests de nos algorithmes se sont déroulés sans problèmes particuliers, si nous excluons notre tentative d'implémentation d'un réseau de neurones. Une prise en compte d'autres éléments comme la position des mains de l'utilisateur ou les caractéristiques physiques des objets (taille, couleur...) et une étude de leurs influences respectives aurait été pertinentes mais nous manquions de temps ou d'informations sur l'influence probable de ces paramètres. De même, il aurait été intéressant de tester sur d'autres jeux de données nos algorithmes, mais en considérant ceux à disposition, cela aurait été redondant.

Que pouvons-nous retenir de ce projet ? En terme de connaissances, nous avons appris énormément au cours de ce dernier : nous avons découvert un secteur de recherche entier de l'Intelligence Artificielle, un langage de programmation, un environnement de développement d'un moteur 3D et des technologies de machine learning.

D'un point de vue pratique, nous avons pu nous rendre compte des difficultés existantes quand il s'agit de prédire les mouvements d'une personne avec seulement la direction de son regard et sa position, l'utilisateur pouvant avoir des changements de comportements imprévisibles. Pour avoir des résultats probants, il faut pour l'instant utiliser des systèmes plus intrusifs, comme utiliser l'électroencéphalographie[13] ou bien ne considérer que les cas où l'utilisateur sera coopératif et n'essaiera pas de tromper le programme.

Bibliographie

[1] "CoVR : A Large-Scale Force-Feedback Robotic Interface for Non-Deterministic Scenarios in VR," p. 14.

[2] L. He, C.-f. Zong, and C. Wang, "Driving intention recognition and behaviour prediction based on a double-layer hidden Markov model," *Journal of Zhejiang University SCIENCE C*, vol. 13, pp. 208–217, Mar. 2012.

[3] D. Aarno and D. Kragic, "Layered HMM for Motion Intention Recognition," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Beijing, China), pp. 5130–5135, IEEE, Oct. 2006.

[4] C. Olah, "Understanding LSTM Networks," Aug. 2015.

[5] L. Yan, X. Gao, X. Zhang, and S. Chang, "Human-Robot Collaboration by Intention Recognition using Deep LSTM Neural Network," in *2019 IEEE 8th International Conference on Fluid Power and Mechatronics (FPM)*, (Wuhan, China), pp. 1390–1396, IEEE, Apr. 2019.

[6] Rheto, "Français : Schéma champ de vision humaine," Jan. 2010.

[7] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity : A General Platform for Intelligent Agents," *arXiv :1809.02627 [cs, stat]*, May 2020. arXiv : 1809.02627.

[8] C. Anthes, R. J. Garcia-Hernandez, M. Wiedemann, and D. Kranzlmuller, "State of the art of virtual reality technology," in *2016 IEEE Aerospace Conference*, (Big Sky, MT, USA), pp. 1–19, IEEE, Mar. 2016.

[9] T. Petković, D. Puljiz, I. Marković, and B. Hein, "Human intention estimation based on hidden Markov model motion validation for safe flexible robotized warehouses," *Robotics and Computer-Integrated Manufacturing*, vol. 57, pp. 182–196, June 2019.

[10] N. Stefanov, A. Peer, and M. Buss, "Online intention recognition for computer-assisted teleoperation," in *2010 IEEE International Conference on Robotics and Automation*, (Anchorage, AK), pp. 5334–5339, IEEE, May 2010.

[11] D. J. Phillips, T. A. Wheeler, and M. J. Kochenderfer, "Generalizable intention prediction of human drivers at intersections," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, (Los Angeles, CA, USA), pp. 1665–1670, IEEE, June 2017.

[12] Z. Wang, B. Wang, H. Liu, and Z. Kong, "Recurrent convolutional networks based intention recognition for human-robot collaboration tasks," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, (Banff, AB), pp. 1675–1680, IEEE, Oct. 2017.

[13] D. S. V. Bandara, J. Arata, and K. Kigichi, "Task based motion intention prediction with EEG signals," in *2016 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*, pp. 57–60, Dec. 2016.

[14] T. Nescher and A. Kunz, "Analysis of Short Term Path Prediction of Human Locomotion for Augmented and Virtual Reality Applications," in *2012 International Conference on Cyberworlds*, (Darmstadt, Germany), pp. 15–22, IEEE, Sept. 2012.

[15] W. Wang, R. Li, Y. Chen, and Y. Jia, "Human Intention Prediction in Human-Robot Collaborative Tasks," in *Companion of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, (Chicago IL USA), pp. 279–280, ACM, Mar. 2018.

[16] A. Abuduweili, S. Li, and C. Liu, "Adaptable Human Intention and Trajectory Prediction for Human-Robot Collaboration," *arXiv :1909.05089 [cs]*, Sept. 2019. arXiv : 1909.05089.

[17] L. Cohen, *Interaction naturelle avec une scène virtuelle de micromanipulation*. PhD thesis, Pierre et Marie Curie, May 2015.

[18] H. He, Y. She, J. Xiahou, J. Yao, J. Li, Q. Hong, and Y. Ji, "Real-Time Eye-Gaze Based Interaction for Human Intention Prediction and Emotion Analysis," in *Proceedings of Computer Graphics International 2018 on - CGI 2018*, (Bintan, Island, Indonesia), pp. 185–194, ACM Press, 2018.

[19] O. Dermy, F. Charpillet, and S. Ivaldi, "Multi-modal Intention Prediction with Probabilistic Movement Primitives," in *Human Friendly Robotics* (F. Ficuciello, F. Ruggiero, and A. Finzi, eds.), vol. 7, pp. 181–196, Cham : Springer International Publishing, 2019. Series Title : Springer Proceedings in Advanced Robotics.