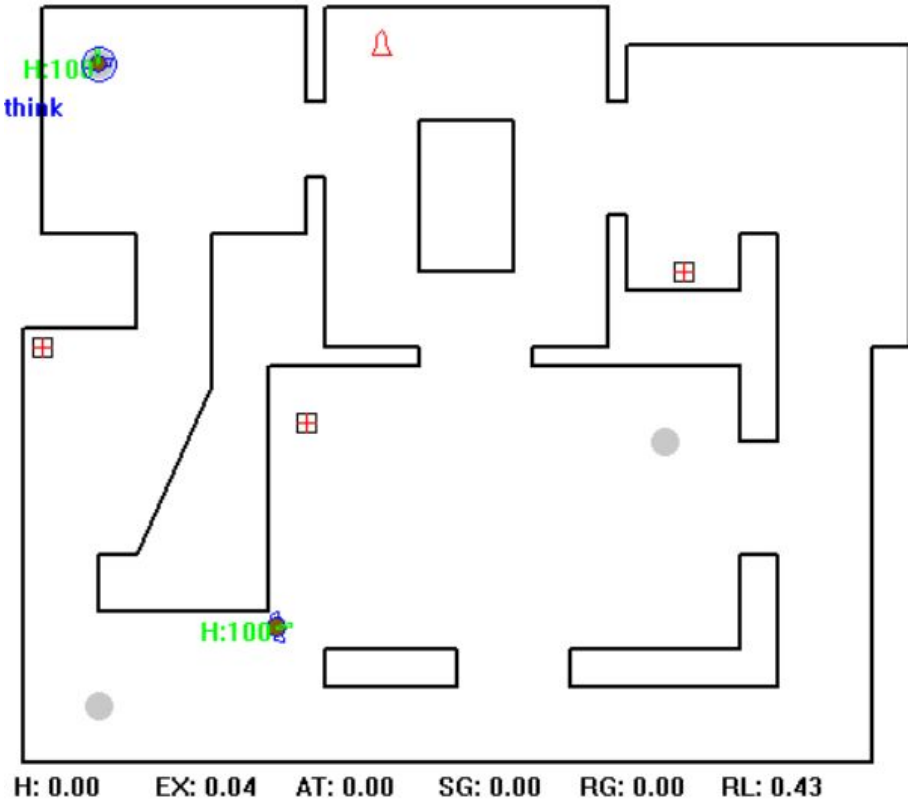
Mini-projet Raven

Noémy Artigouha
Guilhem Cichocki
Elise Combette
Mehdi Fakihani
Damien Maveyraud
Loïc Mouton



I - Détails des livrables **3**

II - Travaux effectués **3**

II.I - Sélection du lance-roquettes par logique floue 3

II.II - Visée altérée par logique floue 3

II.III - Introduction d'un joueur humain 4

II.IV - Bot apprenant 4

II.V - Stratégie de défense 5

II.VI - Création d'équipes d'agents 5

II.VII - La grenade 5

I - Détails des livrables

Trois exécutables sont fournis avec le rapport de ce projet :

- RavenWithNeuronNetwork : lance le jeu Raven avec un joueur humain et un bot. Le réseau de neurone est activé et le bot est apprenant.
- RavenWithoutNeuronNetwork : lance le jeu Raven avec un joueur humain et un bot. Le bot possède une intelligence pré-définie.
- RavenTeams : met en scène l'affrontement de 2 équipes avec des stratégies différentes.

II - Travaux effectués

II.I - Sélection du lance-roquettes par logique floue

Nous avons ajouté des règles concernant la logique floue dirigeant la sélection de l'arme "Lance-Roquettes". Les trois variables floues "Distance à la cible", "Etat des munitions" et "Durabilité" possèdent désormais 5 différents ensembles.

II.II - Visée altérée par logique floue

Afin de simuler une visée réaliste de la part des bots, nous l'avons altérée en utilisant les outils de logique floue fournis. Les bots auront désormais davantage de difficulté à atteindre leur cible dans certaines situations, en fonction de la distance les séparant, de leur vitesse au moment du tir et du temps pendant lequel la cible est demeurée visible.

Pour introduire cette fonctionnalité, un module de logique floue a été introduit dans ***Raven_WeaponSystem***. Une variable floue "Accuracy" sert à ajouter un flou plus ou moins important dans la méthode *AddNoiseToAim*. Les règles définissant l'état d'Accuracy sont définies sur ce modèle dans une méthode *InitializeFuzzyModule()* :

```
// Définition des règles
m_FuzzyModule.AddRule(FzAND(Target_Close, ShooterSpeed_Slow, TimeInSight_Short), Accuracy_Good);
m_FuzzyModule.AddRule(FzAND(Target_Close, ShooterSpeed_Slow, TimeInSight_Medium), Accuracy_Good);
m_FuzzyModule.AddRule(FzAND(Target_Close, ShooterSpeed_Slow, TimeInSight_Long), Accuracy_Good);
m_FuzzyModule.AddRule(FzAND(Target_Close, ShooterSpeed_Medium, TimeInSight_Short), Accuracy_Medium);
m_FuzzyModule.AddRule(FzAND(Target_Close, ShooterSpeed_Medium, TimeInSight_Medium), Accuracy_Good);
m_FuzzyModule.AddRule(FzAND(Target_Close, ShooterSpeed_Medium, TimeInSight_Long), Accuracy_Good);
m_FuzzyModule.AddRule(FzAND(Target_Close, ShooterSpeed_Fast, TimeInSight_Short), Accuracy_Medium);
m_FuzzyModule.AddRule(FzAND(Target_Close, ShooterSpeed_Fast, TimeInSight_Medium), Accuracy_Medium);
m_FuzzyModule.AddRule(FzAND(Target_Close, ShooterSpeed_Fast, TimeInSight_Long), Accuracy_Good);
```

II.III - Introduction d'un joueur humain

Un bot humain est intégré au jeu. Il peut se déplacer à l'aide des touches **ZQSD** et tirer avec le **clic gauche** de la souris. Il peut également choisir son arme à l'aide de la **molette de la souris**. Ce bot humain est créé dès le début du jeu dans la fonction AddBots() de la classe Raven_Game. Si vous ne souhaitez pas de bot humain, il suffit de commenter la ligne CreateHumanBot(rb); dans la fonction AddBots().
Quand un bot humain est créé, les autres bots ne sont pas contrôlables. Quand il n'y a pas de bots humains, les autres sont contrôlables avec les commandes de base. Déplacement avec le clic droit de la souris et changement d'armes avec le pavé numérique.

II.IV - Bot apprenant

Nous avons pris l'initiative de recoder notre propre réseau de neurones pour ce projet. Nous avons ainsi découvert plus en profondeur le fonctionnement de cet algorithme. Une fonction nous permet de récupérer un certain nombre de variables pendant une partie au sein du projet Raven. Pour obtenir ces informations dans un fichier, il faut mettre la variable **getHumanData** à **true** dans le *main.cpp* et n'avoir que **2 bots** sur la map dont un **humain**. Le fichier résultant sert d'entraînement à notre réseau. Il possède les données suivantes :

- Santé du bot humain et de l'autre bot
- Savoir si l'autre bot est visible pour le bot humain
- La distance qui sépare les 2 bots
- La direction dans laquelle regarde le bot humain
- La position du bot humain et de l'autre bot
- Le type d'arme utilisé par le bot humain
- Les munitions disponibles pour cette arme
- Savoir si le bot humain a tiré ou non

Un second projet nous permet d'entraîner le réseau et de récupérer l'état final des neurones dans un fichier. Nous avons choisis de découpler cette fonctionnalité du jeu Raven pour ne pas avoir un apprentissage à chaque lancement du jeu. Le réseau que nous avons conçu a pour but de déterminer si le bot apprenant doit tirer ou non dans la situation donnée en entrée.

Nous avons constaté qu'entraîner un tel réseau prendrait en réalité bien plus de temps que celui dont nous disposions. Voici les résultats obtenus avec 11 variables d'entrées, 3 couches cachées contenant chacunes 4 neurones et des données collectées sur 30 minutes de jeu : le réseau est sous-entraîné et choisit de tirer dans toutes les situations. Cette stratégie n'est pas complètement surprenante puisque dans le pire des cas il ne touchera pas sa cible. Nous aurions aimé entraîner le réseau avec beaucoup plus de données pour voir un résultat plus impressionnant.

Ce comportement est disponible en mettant la variable booléenne **NeuronIsActivated** à **true** dans le fichier *main.cpp.*

II.V - Stratégie de défense

Nous avons choisi de créer un nouveau but appelé GoalSafeDodge. Désormais, lorsqu'un joueur contrôlé par l'ordinateur se dirige vers une arme, des munitions ou un trousse de soin, un comportement d'esquive est ajouté. Nous nous sommes inspiré du but GoaldDodgeSideToSide en modifiant un peu le comportement de l'agent. Ainsi dans notre cas, l'agent effectuera des esquives en choisissant un côté aléatoire en droite et gauche pour que le mouvement ne soit pas prévisible pour un humain. Il est ainsi difficile de tirer sur un bot se dirigeant vers une trousse de soin.

Ce comportement est présent dans tous les exécutables.

II.VI - Création d'équipes d'agents

Pour pouvoir créer des équipes d'agents Raven, nous avons mis en place des **Raven_TeamManager** constituée de **Raven_Teammate**.

Le comportement des **Raven_Teammate** est le même que celui des **Raven_Bot** à quelques exceptions près; ils possèdent chacun une référence à une **Raven_TeamManager** qui correspond à leur équipe et ils ont pour objectif supplémentaire de rester proche d'un coéquipier. Pour cela il demande régulièrement à leur coéquipier leur position, via envoie de message, puis cherche à se rapprocher de celui qui est le moins loin.

Pour générer des **Raven_Teammate** et peupler l'équipe durant la partie, il suffit de presser la **touche T pour ajouter** un **Raven_Teammate** et la **touche Y pour le supprimer**, à la manière des agents Raven classiques.

Nous avons également mis en place des **Raven_Leader** et **Raven_Follower**. Les **Raven_Leader** sont des **Raven_Bots** particuliers qui sont utiles en tant que meneur d'une équipe. Ils envoient régulièrement des ordres à leurs coéquipiers, via envoie de message. Ces messages sont ignorés par les **Raven_Teammates** mais traités par les **Raven_Follower** qui sont une variante de ces derniers qui acceptent les ordres venant d'un meneur. Ces ordres peuvent être variés mais nous avons seulement implémenté l'ordre "Attaquer cible" qui indique la position d'un bot ennemi à abattre.

II.VII - La grenade

Nous avons ajouté une arme à Raven, cette dernière est la grenade. Elle est disponible dès le début du jeu par tous les bots tout comme l'arme Blaster. Elle possède une quantité infinie de munitions, ainsi les bots peuvent s'en servir comme bon leur semble.

Le comportement de la grenade est semblable à celui de la roquette du lance-roquette, les deux étant des explosifs. Néanmoins, ses attributs sont paramétrables, à l'instar des autres armes, via le script Lua qui se trouve dans le projet.

Du point de vue de l'intelligence artificielle, nous avons intégré l'arme dans le moteur d'inférence utilisé dans le système de logique floue. Pour les règles, nous nous sommes inspirés de celles du lance-roquette. En effet, les deux armes sont très

semblables : nous avons donc pensé qu'elles auraient intérêt à être utilisées dans les mêmes conditions.

Du point de vue fonctionnement, il suffit juste de sélectionner l'arme et de tirer avec pour voir une grenade partir en ligne droite vers l'endroit ciblé. Une fois la cible atteinte, elle explose. Les rebonds n'ont cependant pas été implémentés.