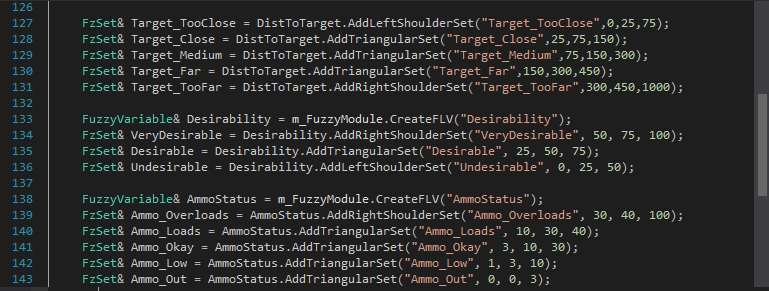
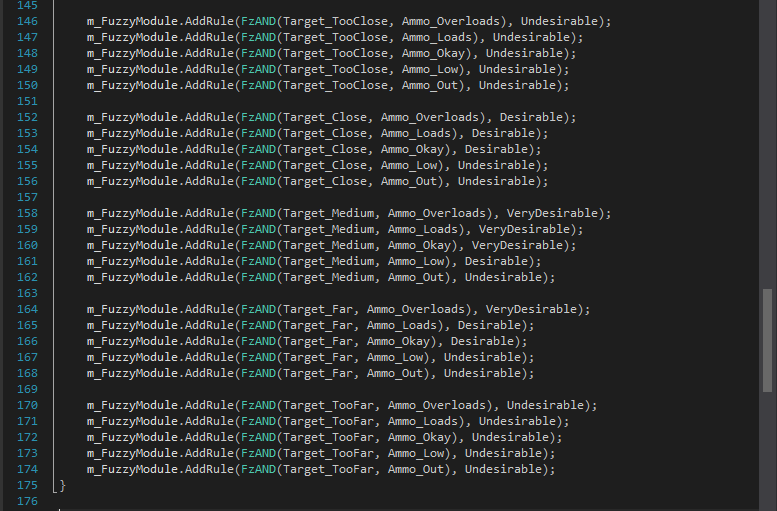
**Point A : Ajout de règles dans le système de sélection du «rocket launcher»**

Image A1 : Définition des ensembles flous

Comme le montre l’image A1, deux ensembles flous ont été rajoutés pour les variables de distance et de munition. L’objectif des nouveaux ensembles est de renforcer les situations où l’arme n’est vraiment pas désirable et les moments où elle devrait être considérée.

Pour la variable de distance, les deux ensembles supplémentaires sont «Target\_TooClose» et «Target\_TooFar ». Ces deux ensembles définissent les distances où l’arme n’est vraiment pas bonne à être utilisée. De cette façon, les ensembles « Target\_Close » et « Target\_Far » regroupent des distances pour lesquelles cette arme peut être un bon choix bien que pas idéale contrairement aux distances avant l’ajout des deux nouveaux ensembles.

Pour la variable de munition, les ensembles «Ammo\_Out» et «Ammo\_Overloads» sont ceux qui ont été rajoutées. Le «rocket launcher» est une arme dont l’intérêt varie beaucoup selon la distance des bots. Ces deux ensembles visent à augmenter l’importance du statut de munition par rapport à la distance. Ainsi, un bot ayant vidé ses munitions ou presque ne devrait pas considérer cette arme même s’il est à une excellente distance pour s’en servir. Dans le cas inverse, le bot devrait chercher à trouver un moyen de se servir de cette arme.

 Image A2 : Définition des règles floues pour les ensembles

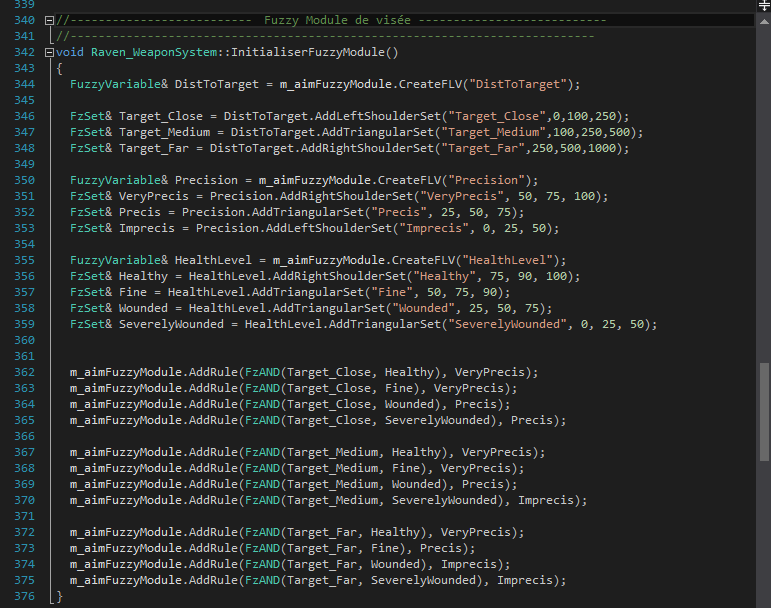
Pour cette deuxième image, les règles pour représenter la désirabilité de l’arme sont définies. Les blocs de cinq lignes regroupent toutes les règles de chacun des ensembles de distance. Le premier et le dernier bloc concernent les règles lorsque la distance n’est vraiment pas bonne pour cette arme. On le voit facilement par le fait que le résultat est toujours «Undesirable» peut importe le statut de munition. On voit aussi que le résultat est aussi le même chaque fois que le statut de munition est «Ammo\_Out».

La différence importante dans les résultats se situe dans les blocs des ensembles «Target\_Close» et «Target\_Far». À l’origine, l’arme était généralement non-désirable lorsque la distance correspondait à ces ensembles. Cependant, la modification de leur frontière avec l’ajout des deux nouveaux ensembles de distance permet de rendre momentanément l’arme plus intéressante. En effet, si le bot a suffisamment de munition de cette arme et qu’aucune autre arme ne paraît plus intéressante, il devrait la privilégier malgré que sa distance ne soit pas parfaitement idéale. En amenant cette idée plus loin, il serait même possible d’encourager le bot à changer sa distance avec la cible tout en lui tirant dessus pour que sa distance devienne idéale.

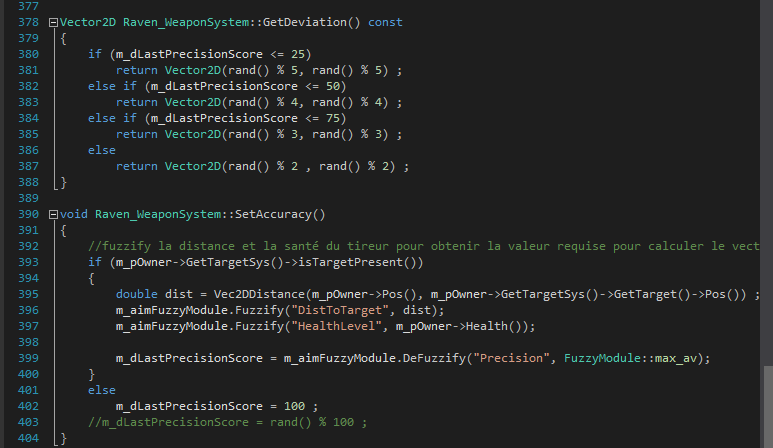
**Partie B : Modification du système de visée en incorporant la logique floue**

Pour ce point, le concept de modification du système de visée vise à introduire le niveau de santé pour déterminer le degré de précision du bot. Il est évidemment plus dur de manier une arme ou faire des tâches avec la même efficacité plus on a subi des blessures importantes.

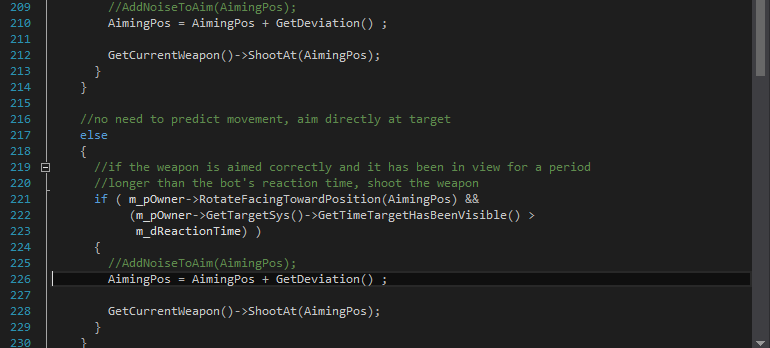
Le module de logique floue servant à définir le degré de précision se décline en 12 règles et deux variables floues, soit la distance et la santé. Le résultat donne le degré de précision qui lui est traduit en une valeur de 0 à 100. Comme on le verra plus tard, cette valeur sert à définir la «force» du vecteur de déviation qui est ajouté au vecteur donnant la trajectoire de tir. C’est ainsi que la précision du tir est affectée.

Image B1 : Initialisation du module de logique floue

Comme le montre l’image précédente, la santé et la distance influe conjointement sur le degré de précision. Ainsi, plus on bot est à la fois loin et blessé, moins il est précis. À l’inverse, plus il est près et en santé, plus il est précis.

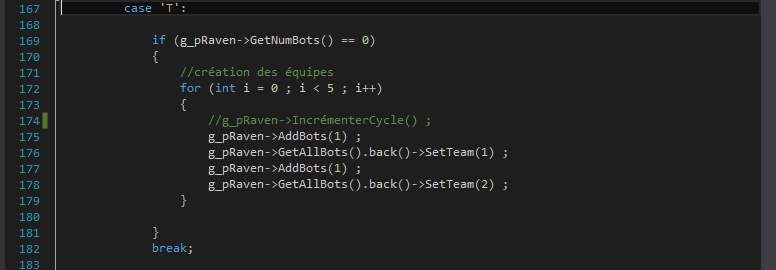
Image B2 : Système permettant d’obtenir le vecteur de déviation

L’image précédente montre la manière d’obtenir la valeur du vecteur de déviation. Le module de logique floue retourne la valeur de 0 à 100. Celle-ci est ensuite utilisée dans la méthode «GetDeviation». Les valeurs de X et Y du vecteur sont définies de manière totalement aléatoire, mais la marge de valeurs possibles devient plus grande moins le bot est précis. Ainsi, le vecteur de déviation affectera davantage celui de trajectoire.

Image B3 : Appel de la méthode donnant le vecteur de déviation

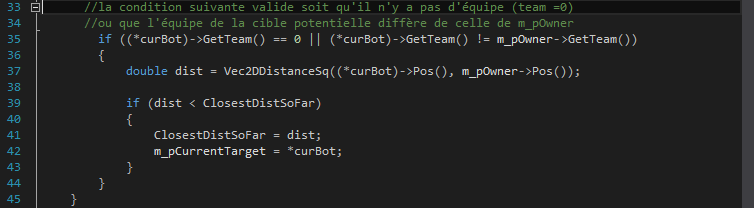
Dans l’image précédente, on voit les appels à la méthode «GetDeviation» pour obtenir le vecteur de déviation dans les lignes 210 et 226. Ce code se trouve dans la méthode «TakeAimAndShoot». Le degré de précision est défini avant dans l’update du bot avant d’appeler la méthode de tir.

**Partie F : Création d’équipes dans la partie Raven**

Image F1 : Case dans le main pour créer les équipes

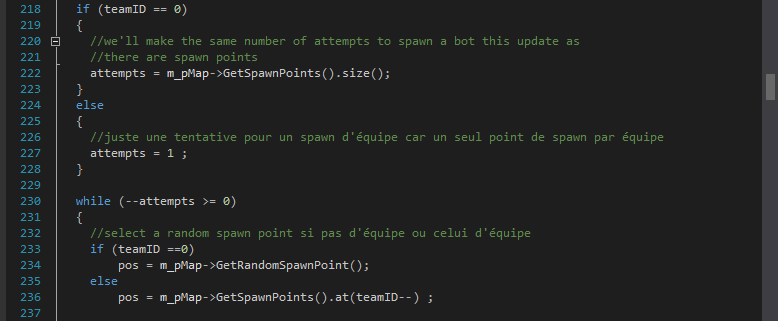
L’image précédente montre les actions entreprises lorsque le joueur clique sur le bouton ‘T’ pour démarrer une partie d’équipe. Cependant, j’ai choisi de mettre la condition que tous les bots présents avant doivent être supprimés. En effet, le match d’équipe implique que les bots ne « respawns » pas afin de savoir quelle équipe aura survécue au combat. Aucun bot neutre ne doit interférer dans la version actuelle du combat d’équipe.

La méthode « SetTeam » a été rajoutée dans la classe du Bot afin de permettre de modifier la valeur de l’équipe à laquelle il appartient. La valeur 0 représente l’absence d’équipe (chacun pour soi). C’est la valeur par défaut telle qu’assignée dans le constructeur.

 Image F2 : Système de ciblage

L’image F2 montre la condition supplémentaire dans la méthode donnant une cible pour les bots. Cette condition sert à s’assurer qu’un bot ne prend pas directement pour cible un allié.

Cependant, il n’y a pas de système présentement qui permet d’éviter que les bots se tirent accidentellement dessus. Par exemple, lorsqu’un bot allié se trouve entre la cible et le bot, celui se fera très certainement tirer dessus à la fois par l’ennemi et son allié en même temps !

Image F3 : Incorporation d’un bot d’équipe

L’image F3 montre les modifications dans la méthode « AttemptToAddBot » dans la classe « Raven\_Game ». L’idée est de faire apparaître tous les bots d’une équipe dans un seul «spawn zone». Ainsi, le nombre de tentative est réduit à un pour trouver une zone libre et la zone choisie est toujours la même. Ceci permet aux coéquipiers d’être ensemble au début du match d’équipe.

**Difficultés rencontrées**

Pour la partie A, cette partie fut assez simple à compléter. Aucune difficulté particulière n’a été rencontrée. Il fallait surtout étudier le code et comprendre le fonctionnement afin de pouvoir y apporter adéquatement les changements voulus.

Dans la partie B, le défi fut un peu plus grand dans la mesure où il ne suffisait pas de modifier un module de logique floue existant. Il fallait en créer un. Cependant, fort de l’expérience acquise en travaillant sur le point A, les difficultés furent assez simples à surmonter. Il a tout de même été nécessaire de faire quelques essais avant d’aboutir à un résultat qui fonctionne. Le plus dur n’était pas de définir les règles, mais de concevoir le système qui fait l’appel du module et qui utilise la valeur de retour pour en faire quelque chose de significatif. Dans le cadre de ce travail, on s’est contenté d’un simple mécanisme aléatoire se servant de la fonction « rand ». La valeur du module de logique floue ne fait qu’influencer l’écart de valeur possible avec cette fonction.

Par rapport au point F, le défi fut davantage significatif malgré la relative simplicité de la tâche demandée. En effet, le système permettant d’ajouter des bots est déjà bien établi et contourner ce système pour créer une équipe entière à la fois n’a pas donné des bons résultats. On souhaitait faire que les membres d’équipe apparaissent au même endroit. Le meilleur moyen fut de modifier le système existant plutôt que d’en créer une version spécifique pour les équipes. Le mécanisme pour que les bots connaissent leur équipe fut quant à lui facile. L’ajout d’un simple attribut et de deux méthodes ont suffit à obtenir ce qu’il fallait.