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

```
126
127     FzSet& Target_TooClose = DistToTarget.AddLeftShoulderSet("Target_TooClose",0,25,75);
128     FzSet& Target_Close = DistToTarget.AddTriangularSet("Target_Close",25,75,150);
129     FzSet& Target_Medium = DistToTarget.AddTriangularSet("Target_Medium",75,150,300);
130     FzSet& Target_Far = DistToTarget.AddTriangularSet("Target_Far",150,300,450);
131     FzSet& Target_TooFar = DistToTarget.AddRightShoulderSet("Target_TooFar",300,450,1000);
132
133     FuzzyVariable& Desirability = m_FuzzyModule.CreateFLV("Desirability");
134     FzSet& VeryDesirable = Desirability.AddRightShoulderSet("VeryDesirable", 50, 75, 100);
135     FzSet& Desirable = Desirability.AddTriangularSet("Desirable", 25, 50, 75);
136     FzSet& Undesirable = Desirability.AddLeftShoulderSet("Undesirable", 0, 25, 50);
137
138     FuzzyVariable& AmmoStatus = m_FuzzyModule.CreateFLV("AmmoStatus");
139     FzSet& Ammo_Overloads = AmmoStatus.AddRightShoulderSet("Ammo_Overloads", 30, 40, 100);
140     FzSet& Ammo_Loads = AmmoStatus.AddTriangularSet("Ammo_Loads", 10, 30, 40);
141     FzSet& Ammo_Okay = AmmoStatus.AddTriangularSet("Ammo_Okay", 3, 10, 30);
142     FzSet& Ammo_Low = AmmoStatus.AddTriangularSet("Ammo_Low", 1, 3, 10);
143     FzSet& Ammo_Out = AmmoStatus.AddTriangularSet("Ammo_Out", 0, 0, 3);
```

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.

```
145
146     m_FuzzyModule.AddRule(FzAND(Target_TooClose, Ammo_Overloads), Undesirable);
147     m_FuzzyModule.AddRule(FzAND(Target_TooClose, Ammo_Loads), Undesirable);
148     m_FuzzyModule.AddRule(FzAND(Target_TooClose, Ammo_Okay), Undesirable);
149     m_FuzzyModule.AddRule(FzAND(Target_TooClose, Ammo_Low), Undesirable);
150     m_FuzzyModule.AddRule(FzAND(Target_TooClose, Ammo_Out), Undesirable);
151
152     m_FuzzyModule.AddRule(FzAND(Target_Close, Ammo_Overloads), Desirable);
153     m_FuzzyModule.AddRule(FzAND(Target_Close, Ammo_Loads), Desirable);
154     m_FuzzyModule.AddRule(FzAND(Target_Close, Ammo_Okay), Desirable);
155     m_FuzzyModule.AddRule(FzAND(Target_Close, Ammo_Low), Undesirable);
156     m_FuzzyModule.AddRule(FzAND(Target_Close, Ammo_Out), Undesirable);
157
158     m_FuzzyModule.AddRule(FzAND(Target_Medium, Ammo_Overloads), VeryDesirable);
159     m_FuzzyModule.AddRule(FzAND(Target_Medium, Ammo_Loads), VeryDesirable);
160     m_FuzzyModule.AddRule(FzAND(Target_Medium, Ammo_Okay), VeryDesirable);
161     m_FuzzyModule.AddRule(FzAND(Target_Medium, Ammo_Low), Desirable);
162     m_FuzzyModule.AddRule(FzAND(Target_Medium, Ammo_Out), Undesirable);
163
164     m_FuzzyModule.AddRule(FzAND(Target_Far, Ammo_Overloads), VeryDesirable);
165     m_FuzzyModule.AddRule(FzAND(Target_Far, Ammo_Loads), Desirable);
166     m_FuzzyModule.AddRule(FzAND(Target_Far, Ammo_Okay), Desirable);
167     m_FuzzyModule.AddRule(FzAND(Target_Far, Ammo_Low), Undesirable);
168     m_FuzzyModule.AddRule(FzAND(Target_Far, Ammo_Out), Undesirable);
169
170     m_FuzzyModule.AddRule(FzAND(Target_TooFar, Ammo_Overloads), Undesirable);
171     m_FuzzyModule.AddRule(FzAND(Target_TooFar, Ammo_Loads), Undesirable);
172     m_FuzzyModule.AddRule(FzAND(Target_TooFar, Ammo_Okay), Undesirable);
173     m_FuzzyModule.AddRule(FzAND(Target_TooFar, Ammo_Low), Undesirable);
174     m_FuzzyModule.AddRule(FzAND(Target_TooFar, Ammo_Out), Undesirable);
175 }
176
```

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.

```
339
340 //--------------------------  Fuzzy Module de visée --------------------------
341 //----------------------------------------------------------------------------
342 void Raven_WeaponSystem::InitialiserFuzzyModule()
343 {
344   FuzzyVariable& DistToTarget = m_aimFuzzyModule.CreateFLV("DistToTarget");
345
346   FzSet& Target_Close = DistToTarget.AddLeftShoulderSet("Target_Close",0,100,250);
347   FzSet& Target_Medium = DistToTarget.AddTriangularSet("Target_Medium",100,250,500);
348   FzSet& Target_Far = DistToTarget.AddRightShoulderSet("Target_Far",250,500,1000);
349
350   FuzzyVariable& Precision = m_aimFuzzyModule.CreateFLV("Precision");
351   FzSet& VeryPrecis = Precision.AddRightShoulderSet("VeryPrecis", 50, 75, 100);
352   FzSet& Precis = Precision.AddTriangularSet("Precis", 25, 50, 75);
353   FzSet& Imprecis = Precision.AddLeftShoulderSet("Imprecis", 0, 25, 50);
354
355   FuzzyVariable& HealthLevel = m_aimFuzzyModule.CreateFLV("HealthLevel");
356   FzSet& Healthy = HealthLevel.AddRightShoulderSet("Healthy", 75, 90, 100);
357   FzSet& Fine = HealthLevel.AddTriangularSet("Fine", 50, 75, 90);
358   FzSet& Wounded = HealthLevel.AddTriangularSet("Wounded", 25, 50, 75);
359   FzSet& SeverelyWounded = HealthLevel.AddTriangularSet("SeverelyWounded", 0, 25, 50);
360
361
362   m_aimFuzzyModule.AddRule(FzAND(Target_Close, Healthy), VeryPrecis);
363   m_aimFuzzyModule.AddRule(FzAND(Target_Close, Fine), VeryPrecis);
364   m_aimFuzzyModule.AddRule(FzAND(Target_Close, Wounded), Precis);
365   m_aimFuzzyModule.AddRule(FzAND(Target_Close, SeverelyWounded), Precis);
366
367   m_aimFuzzyModule.AddRule(FzAND(Target_Medium, Healthy), VeryPrecis);
368   m_aimFuzzyModule.AddRule(FzAND(Target_Medium, Fine), VeryPrecis);
369   m_aimFuzzyModule.AddRule(FzAND(Target_Medium, Wounded), Precis);
370   m_aimFuzzyModule.AddRule(FzAND(Target_Medium, SeverelyWounded), Imprecis);
371
372   m_aimFuzzyModule.AddRule(FzAND(Target_Far, Healthy), VeryPrecis);
373   m_aimFuzzyModule.AddRule(FzAND(Target_Far, Fine), Precis);
374   m_aimFuzzyModule.AddRule(FzAND(Target_Far, Wounded), Imprecis);
375   m_aimFuzzyModule.AddRule(FzAND(Target_Far, SeverelyWounded), Imprecis);
376 }
```

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.

```
377
378 Vector2D Raven_WeaponSystem::GetDeviation() const
379 {
380     if (m_dLastPrecisionScore <= 25)
381         return Vector2D(rand() % 5, rand() % 5) ;
382     else if (m_dLastPrecisionScore <= 50)
383         return Vector2D(rand() % 4, rand() % 4) ;
384     else if (m_dLastPrecisionScore <= 75)
385         return Vector2D(rand() % 3, rand() % 3) ;
386     else
387         return Vector2D(rand() % 2 , rand() % 2) ;
388 }
389
390 void Raven_WeaponSystem::SetAccuracy()
391 {
392     //fuzzify la distance et la santé du tireur pour obtenir la valeur requise pour calculer le vect
393     if (m_pOwner->GetTargetSys()->isTargetPresent())
394     {
395         double dist = Vec2DDistance(m_pOwner->Pos(), m_pOwner->GetTargetSys()->GetTarget()->Pos()) ;
396         m_aimFuzzyModule.Fuzzify("DistToTarget", dist);
397         m_aimFuzzyModule.Fuzzify("HealthLevel", m_pOwner->Health());
398
399         m_dLastPrecisionScore = m_aimFuzzyModule.DeFuzzify("Precision", FuzzyModule::max_av);
400     }
401     else
402         m_dLastPrecisionScore = 100 ;
403     //m_dLastPrecisionScore = rand() % 100 ;
404 }
```

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.

```
209         //AddNoiseToAim(AimingPos);
210         AimingPos = AimingPos + GetDeviation() ;
211
212         GetCurrentWeapon()->ShootAt(AimingPos);
213       }
214     }
215
216     //no need to predict movement, aim directly at target
217     else
218     {
219       //if the weapon is aimed correctly and it has been in view for a period
220       //longer than the bot's reaction time, shoot the weapon
221       if ( m_pOwner->RotateFacingTowardPosition(AimingPos) &&
222            (m_pOwner->GetTargetSys()->GetTimeTargetHasBeenVisible() >
223             m_dReactionTime) )
224       {
225         //AddNoiseToAim(AimingPos);
226         AimingPos = AimingPos + GetDeviation() ;
227
228         GetCurrentWeapon()->ShootAt(AimingPos);
229       }
230     }
```

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

```
167         case 'T':
168
169             if (g_pRaven->GetNumBots() == 0)
170             {
171                 //création des équipes
172                 for (int i = 0 ; i < 5 ; i++)
173                 {
174                     //g_pRaven->IncrémenterCycle() ;
175                     g_pRaven->AddBots(1) ;
176                     g_pRaven->GetAllBots().back()->SetTeam(1) ;
177                     g_pRaven->AddBots(1) ;
178                     g_pRaven->GetAllBots().back()->SetTeam(2) ;
179                 }
180
181             }
182             break;
183
```

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.

```
33      //la condition suivante valide soit qu'il n'y a pas d'équipe (team =0)
34      //ou que l'équipe de la cible potentielle diffère de celle de m_pOwner
35        if ((*curBot)->GetTeam() == 0 || (*curBot)->GetTeam() != m_pOwner->GetTeam())
36        {
37            double dist = Vec2DDistanceSq((*curBot)->Pos(), m_pOwner->Pos());
38
39            if (dist < ClosestDistSoFar)
40            {
41                ClosestDistSoFar = dist;
42                m_pCurrentTarget = *curBot;
43            }
44        }
45    }
```

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 !

```
218   if (teamID == 0)
219   {
220     //we'll make the same number of attempts to spawn a bot this update as
221     //there are spawn points
222     attempts = m_pMap->GetSpawnPoints().size();
223   }
224   else
225   {
226     //juste une tentative pour un spawn d'équipe car un seul point de spawn par équipe
227     attempts = 1 ;
228   }
229
230   while (--attempts >= 0)
231   {
232     //select a random spawn point si pas d'équipe ou celui d'équipe
233     if (teamID ==0)
234         pos = m_pMap->GetRandomSpawnPoint();
235     else
236         pos = m_pMap->GetSpawnPoints().at(teamID--) ;
237
```

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.