UNIVERSITÉ DE SHERBROOKE

Battle for Pangora

Rapport IA – Projet intégrateur en jeux vidéo

Équipe : Violet Murder

Table des matières

1 IA finale dans le jeu

Cette section a pour but de présenter l'IA finale dans le jeu Battle for Pangora. On parlera donc ici du système de Squad mis en place, des différentes manoeuvres possibles pour une Squad et des comportements des agents d'une Squad.

1.1 Le système de Squad

Le système de Squad a beaucoup évolué durant le projet et nous sommes parvenu à l'architecture suivante :

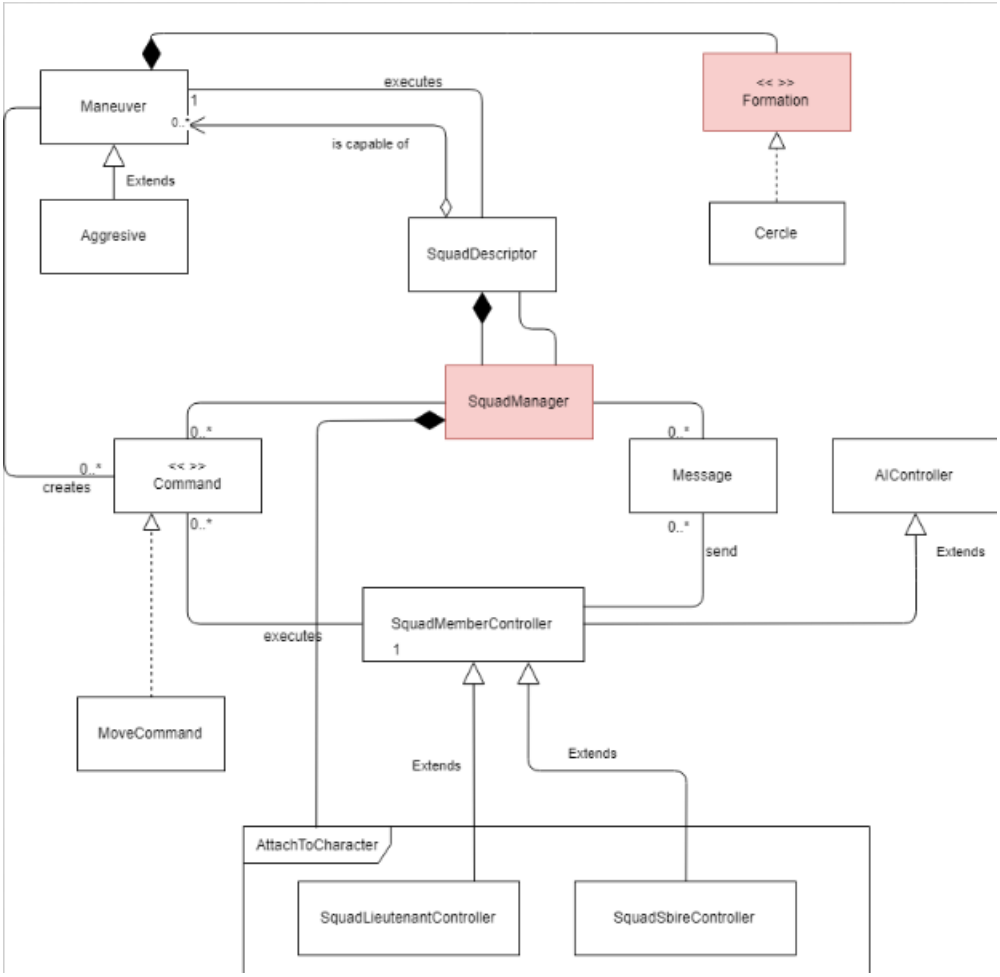


FIGURE 1 – Architecture finale de Squad

1.1.1 Description des différents modules du système

Il y a deux types de cerveaux dans une Squad, l'un s'occupant des décisions à grande échelle, c'est le SquadManager, et l'autre s'occupant des décisions tactiques, c'est la Maneuver.

Dans une Squad, le **SquadManager** correspond au cerveau dans le sens où il a la possibilité de changer la Maneuver actuelle de la Squad où les décisions tactiques sont prises. C'est également le canal de réception des messages.

La **Maneuver** s'occupe de distribuer à tous ses membres des actions à effectuer appellées commandes en fonctions de l'analyse de son environnement et des Messages reçus.

Les **Messages** peuvent aussi bien être envoyés depuis les **SquadMemberController**, que depuis toute autre endroit du système (joueur mort, déclenchement d'évènement, capture de point de contrôle, ...). Ils servent a prévenir le SquadManager des changements autour de lui et à l'informer des modifications. Le SquadManager peut alors prendre des décisions en fonction des messages qu'il reçoit. La plupart du temps, ces messages seront retransmis à la Maneuver pour qu'elle prenne ses propres décisions.

Les **SquadMemberController** sont les contrôleurs attachés à chacun des agents d'une Squad. C'est eux qui vont exécuter les commandes reçues.

Les **Commandes** sont créées par le SquadManager et exécutées par les SquadMemeberController.

On a ensuite la partie qui correspond au corps de la Squad : le **SquadDescriptor**. Il regroupe les informations relatives à le position des sbires dans la Squad, l'état de formation actuel, le nombre de membres, ...
On établit ensuite un dialogue entre le SquadDescriptor et le SquadManager.

La **Formation** est un élément représentant les positions relatives à la Squad des membres de cette même Squad.

1.1.2 Quelques remarques

Si nous avions eu plus de temps, nous aurions voulu coder la fusion de Squad. C'est à dire que deux squads qui sont dans la même manoeuvre, suffisamment proches, deviennent une seule et même Squad. Tous les outils étaient disponibles mais il nous aurait fallût un peu plus de temps.
Ce système permet de rajouter des comportements (i.e. manoeuvres) et des formations très facilement.
La mécanique de message suggérée par Carle Côté nous a également permis d'avoir un maximum de contrôle sur notre système et de pouvoir communiquer avec lui depuis l'extérieur.

1.2 Manoeuvre de Squad

Une Maneuver, car il y en a plusieurs en fonction du comportement souhaité, fonctionne comme suit :

— la Maneuver de notre Squad prend une décision en fonction de son environnement

— la Maneuver distribue des ordres à tous les membres de la Squad.
— les membres de la Squad analysent leur environnement et renvoient ce qu'ils jugent pertinent à la Squad sous forme de Message.
— lorsque la Maneuver possède assez de messages suffisamment pertinents, elle prend alors une nouvelle décision.

Concrètement, une Maneuver est un Behavior Tree. Cependant c'est un Behavior Tree particulier car il a la particularité de s'arrêter dans chacun de ses noeuds et de n'en sortir que lorsque son action est soit terminée soit échouée (ce qui est normal) et que l'information nécessaire pour savoir si cette action est terminée ou échouée ne dépend pas de la Maneuver elle-même mais de tout son environnement ! C'est pour cela que nous avons ajouté notre système de Message qui nous permet de changer de noeuds.

Ainsi nous n'avons que deux choses à faire dans les noeuds du notre Behavior Tree. Premièrement coder son comportement. Deuxièmement coder tous ses cas d'arrêt.

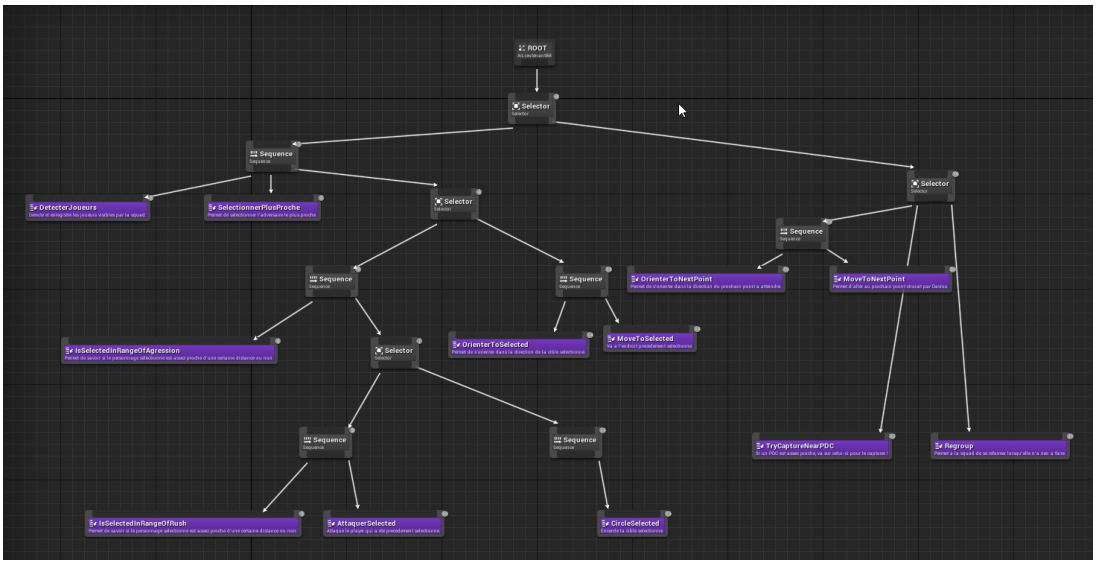
Une chose que nous avons remarqué en codant les cas d'arrêt est que le comportement de notre BehaviorTree ressemblait fortement à une machine à états. Cependant la grande différence tient dans le fait que dans une machine à états on code les transitions (donc les conditions d'arrêts d'un état pour aller vers un autre état) pour aller vers un autre état, alors que ici on ne code que la condition d'arrêt mais pas vers quel prochain état aller. Ce qui fait que le nombre de conditions est constant plutôt que d'être proportionnel aux nombres d'états du BehaviorTree ! Il s'agit donc d'une propriété très intéressante de scalabilité. Nous avons appelé ce système BehaviorMachine pour nous amuser :)

Nous avons 5 Maneuvres d'implémentées, chacune représentant une stratégie. Ces stratégies sont choisis par Davros le Seigneur du Chaos selon ses envies, à l'exception de la dernière qui est prise automatiquement si jamais la Squad venait à perdre le Point de Contrôle qui lui est associé.
Voici ces stratégies/Maneuver :

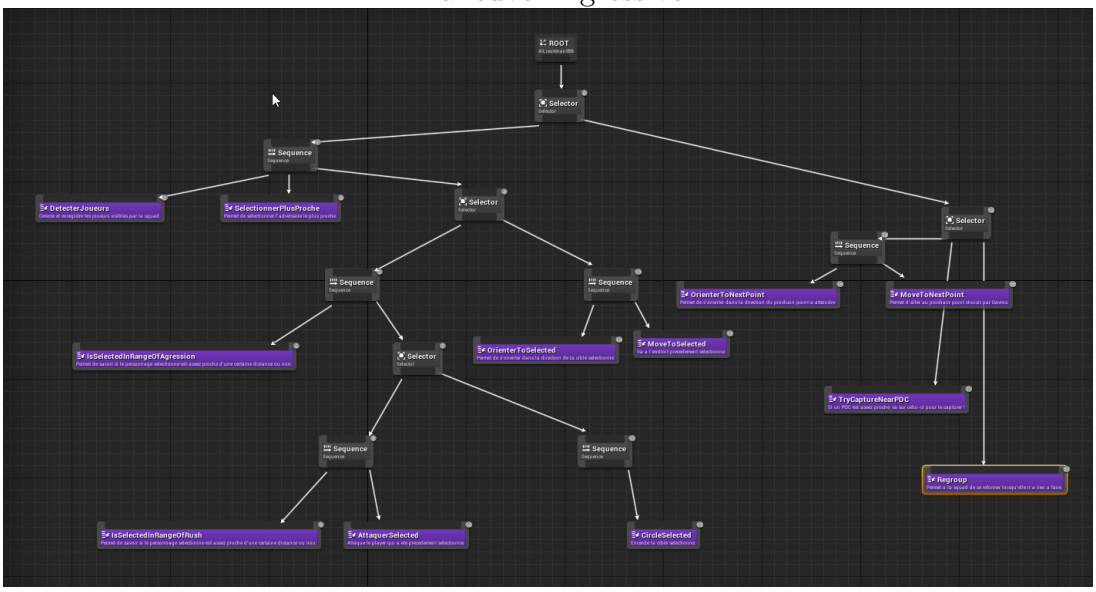
— Agressive : la stratégie par défaut, cherchant à agresser les adversaire en gardant un semblant de coordination militaire. Cette stratégie utilise notamment l'encerclement.
— Assaut : cette stratégie vise à agresser sans retenu tous les adversaires qui seront en vue.
— Défensive : cette stratégie cherchera à défendre l'endroit où elle se trouve ou l'endroit où elle doit aller. Elle n'agressera un adversaire que si celui-ci se trouve trop proche.
— Survie : cette stratégie a pour seul but de préserver son Lieutenant qui, on le rappelle, est le coeur de la Squad et si celui-ci meurt, alors la Squad est dissoute. Ainsi elle fuira les adversaires, mais si ceux-ci s'approchent vraiment trop dangereusement alors elle contre-attaquera.
— Recall : cette stratégie a pour but d'aller reconquérir le Point de Contrôle associé à la Squad si jamais celui-ci s'est fait prendre.

Comme il est difficle de bien expliquer le comportement des Maneuvres par des mots,

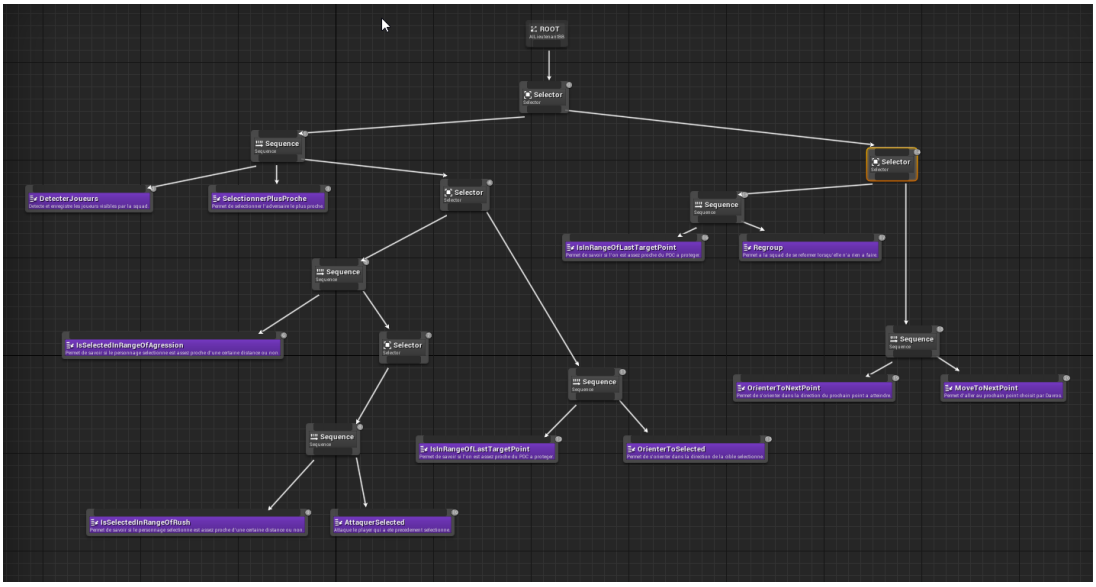
voici les BehaviorTrees associés à chacune d'entre elles.
On notera les grandes similarités entre ceux-ci. C'est un effet voulu car cela nous a permis de factoriser notre code et de le réutiliser en ajustant les paramètres pour obtenir des effets différents.



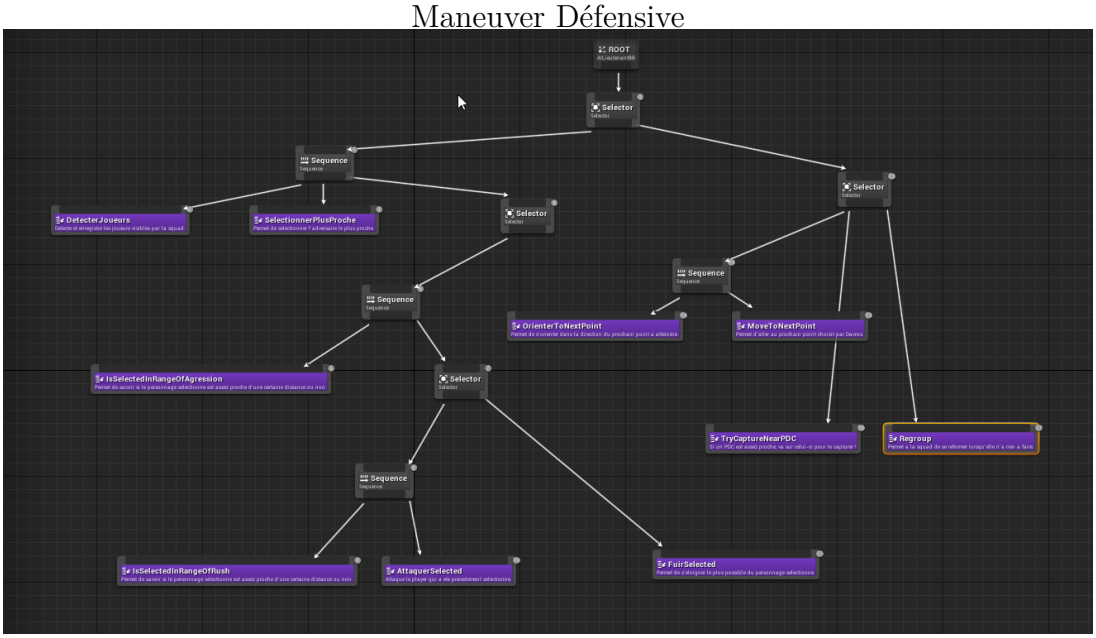
Maneuver Agressive



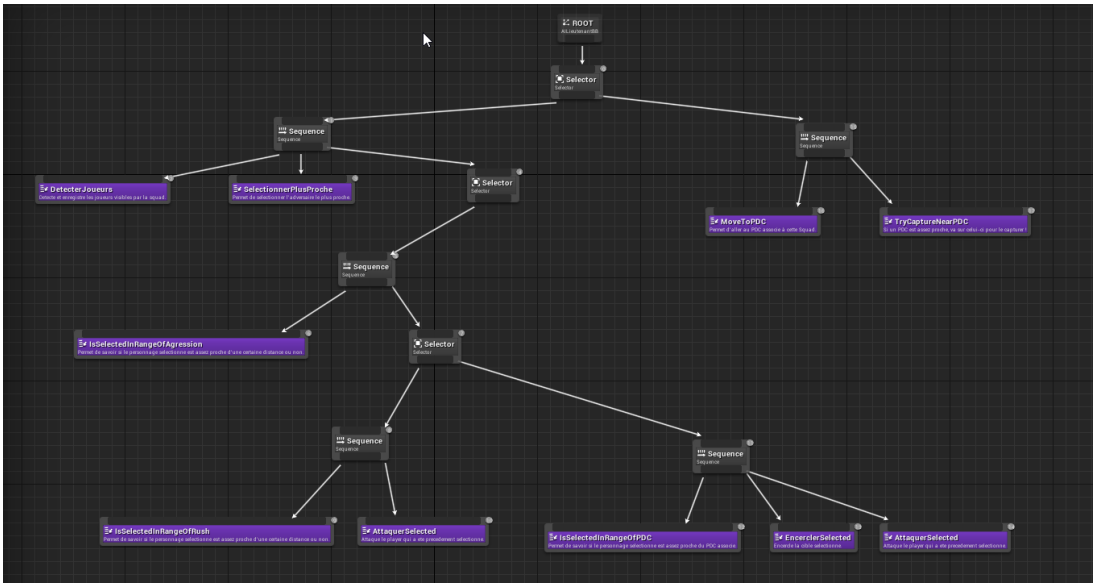
Maneuver Assault



Maneuver Défensive



Maneuver Survie



Maneuver Recall

1.3 Messages

Les Messages sont les éléments de base permettant de transmettre de l'information à notre Squad. Tous les éléments du jeu ont la possibilité de leur en envoyer !

Voici la liste de tous les Messages que nous avons utilisé :
— COMPLETED // La tache d'un membre a été complété.
— FAILED // La tache d'un membre ne peut pas être effectuée.
— DIED // Le membre en train d'effectuer la tâche est mort.
— JOIN // Un sbire vient de rejoindre la Squad.
— KILLED // La cible de l'attaque a été tuée.
— IN_VISION_RANGE // Il y a un ennemi en vue.
— IN_AGRESSION_RANGE // Il y a un ennemi a porté d'agression.
— IN_RUSH_RANGE // Il y a un ennemi a portée de rush.
— OUT_OF_VISION_RANGE // Il n'y a pas d'ennemis en vue.
— OUT_OF_AGRESSION_RANGE // Il n'y a pas d'ennemi à portée d'agression.
— OUT_OF_RUSH_RANGE // Il n'y a pas d'ennemi à portée de rush.
— RECALL // Pour que la Squad revienne défendre son PDC !!! :)
— STOP_RECALL // Pour que la Squad arrête de défendre son PDC :)
— STOP // Détruit le parcours courant de la Squad
— ADD_TARGET_POINT // Permet de notifier la Squad que l'on vient de lui rajouter un targetPoint !

La plupart des Messages ne sont envoyés qu'une seule fois, lorsque c'est nécessaire. Mais les Messages du système de vision tel que IN_VISION_RANGE, IN_AGRESSION_RANGE, OUT_OF_VISION_RANGE ... sont malheureusement envoyés à toutes les frames. Cela ne coûte pas très cher car il est rapide de déterminer si il y a un adversaire à portée, mais le système aurait pû être plus performant de ce côté là et n'envoyer un message que lorsqu'il y a un changement d'état, plutôt que à toutes les frames.

Voici un exemple d'un noeud d'un Behavior Tree, voici le noeud d'attaque de l'adversaire sélectionné :

```
1  /* ExecuteTask est une fonction du moteur que nous reutilisons pour
       lui faire faire des initialisations communes a toutes les taches.
       Puis nous lui faisons appeller ExecuteTaskImpl, une fonction
       virtuelle que nous overridons ici pour avoir le code le plus
       simple et lisible possible. */
2  EBTNodeResult::Type UAttaquerSelected::ExecuteTaskImpl(
       UBehaviorTreeComponent& OwnerComp, uint8* NodeMemory, AManeuver*
       maneuvre) {
3      // Ajout d'un logger pour pouvoir facilement traquer le
           comportement de nos Squads
4      MY_LOG_IA(TEXT("%s MODE = AttaquerSelected"), *maneuvre->
           GetDescriptor()->GetName());
5
6      // Envoie de toutes les commandes d'attaques
7      maneuvre->Attaquer(maneuvre->GetJoueurSelected());
8
9      return EBTNodeResult::InProgress;
10 }
11
12 /* On receptionne les messages */
13 void UAttaquerSelected::OnMessage(UBehaviorTreeComponent & OwnerComp,
       uint8 * NodeMemory, FName Message, int32 RequestID, bool bSuccess
       ) {
14     // OnMessage est aussi override et gere la plupart des gestions
           de Message qui sont communes a toutes les taches, on ne s'
           occupe ici que des Messages particuliers a la tache d'attaque
15     Super::OnMessage(OwnerComp, NodeMemory, Message, RequestID,
           bSuccess);
16
17     // On distingue les messages
18     AManeuver* maneuvre = Cast<AManeuver>(OwnerComp.GetOwner());
19     ASquadMemberController * sbire = maneuvre->GetLastMessage()->
           GetSquadMemberController();
20     ABFPCharacterPlayable * joueur = maneuvre->GetJoueurSelected();
21     switch (maneuvre->GetLastMessage()->GetType()) {
22     case MessageType::COMPLETED:
23         // Si un sibre a finit sa courbe de menace alors on lui
               trouve une autre menace !
24         maneuvre->FaireMenacerMember(sbire, joueur);
25         break;
26     case MessageType::DIED:
27         // Si un sbire qui etait attaquant meurt, il faut le
               remplacer par un autre !
28         maneuvre->Attaquer(joueur); // On recalule toutes les
```

```
                commandes de toute la Squad. Peut etre ameliore.
29              break;
30          case MessageType::KILLED:
31              FinishLatentTask(OwnerComp, EBTNodeResult::Succeeded);
32              break;
33          case MessageType::OUT_OF_VISION_RANGE:
34          case MessageType::OUT_OF_AGRESSION_RANGE:
35              // Si le joueur est trop loin, alors on sort !
36              FinishLatentTask(OwnerComp, EBTNodeResult::Failed);
37              break;
38          case MessageType::OUT_OF_RUSH_RANGE:
39              // Si le joueur est trop loin, alors on sort !
40              if(shouldStopAttaqueOutOfRushRange)
41                  FinishLatentTask(OwnerComp, EBTNodeResult::Failed);
42              break;
43          default:
44              break;
45          }
46      }
```

1.4 Les SquadMemberControllers

Il y en a de 2 types, les Lieutenants et les Sbires. Ces derniers peuvent être aussi nombreux qu'ils veulent ce qui est aussi le cas des Lieutenants sauf que nous n'avons pas implémentés cette fonctionnalité (la fusion de Squads) par manque de temps.
Nous avons donc assumé que les Lieutenants ne sont que 1 dans chaque Squad, ce qui nous a simplifié la vie.
De plus si jamais un lieutenant meurt, la Squad est dissoute d'après les règles du jeu.

Ainsi, comme il y a toujours un et un seul Lieutenant par Squad, nous nous en sommes servis pour le définir comme étant le "centre" de celle-ci, c'est-à-dire que toutes les requêtes de vision et tous les déplacements sont fait par rapport à la position de celui-ci.

1.5 Les Commandes

Les Commandes sont les ordres que nous envoyons à des membres de notre Squad. Chacun d'entre eux ne peux suivre qu'une seule commande à la fois, mais tant qu'il en possède une, il l'exécutera pour toutes les frames et cela jusqu'à la fin du jeu.

Nous avons 5 commandes en tout dans notre jeu :

— la MoveCommand qui permet à un membre de se rendre à un point du NavMesh. Nous avons un certain nombre de paramètres sur celle-ci comme quel NavMesh utiliser (les Lieutenants n'utilisent pas les même NavMesh que les sbires pour ne pas faire longer les murs à notre Squad), et le taux de rafraîchissement par défaut à $2.0 \pm 10\%$ pour éviter que tous les membres de la Squad rafraîchissent leur pathfinding en même temps.

— la MoveToActorCommand qui permet globalement la même chose que la MoveCommand mais qui suit un acteur à la place avec un offset. Si l'offset est non-nul, cet acteur peut être suivis de différentes manières, avec un offset ABSOLU, RELATIF ou RELATIF_TO_POINT pour obtenir différent comportements. Cette fois-ci le taux de rafraîchissement par défaut est à $0.5 \pm 10\%$.
— la SimpleAttackCommand qui va chercher à aller vers un acteur, et si nous sommes suffisamment proche de celui-ci déclencher une attaque en arrêtant de bouger.
— la NoCommand permettant d'arrêter ce que faisait un membre de la Squad précédemment.
— la EndCommand permettant de signaler au sbire qu'il ne recevra plus de commandes de sa Squad et qu'il doit donc agir de son propre chef.

Lors des requêtes de pathfinding, comme celle-ci sont amené à souvent échouer du fait que les sbires se déplacent en Formation autour du Lieutenant (on y vient), nous avons implémentés un système de spirale permettant que si le sbire n'arrive pas à atteindre un point de la carte, il essayera alors un autre point proche de celui-ci jusqu'à ce qu'il en trouve un où se rendre. Ces points étant générés en spirale autour du point original.

1.6 Les Formations

Les Formations sont des positions relatives à la position du Lieutenant que les Sbires d'une Squad peuvent occuper pour donner l'impression que les membres de la Squad se déplacent de manière coordonnée.
Nous avons 2 Formations implémentées dans notre code, la formation en rectangle et la formation en cercle.
Celles-ci prennent en compte des paramètres tels que le nombre de sbires, le nombre de lieutenant (1 !), l'espacement entre les sbires, le nombre de sbires en première ligne, l'angle d'ouverture etc ... pour s'adapter à toutes les situations.
Vous avez pu voir sur notre présentation que nous étions grâce à cela capable de générer de nombreuses Formations différentes à partir de seulement 2 implémentations différentes.

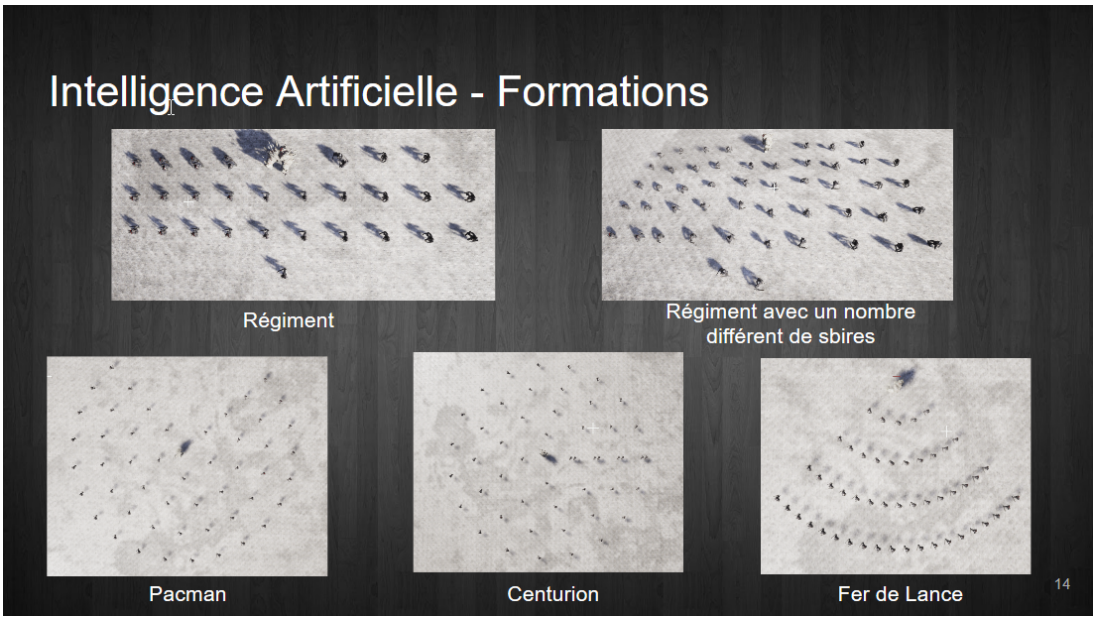


Figure 2 – Les différentes Formations

1.7 Comportements individuels

Une Squad correspond à un ensemble de sbires et un lieutenant représentant le cœur de celle-ci. Ainsi, tant que le lieutenant est vivant, toutes ces unités ne font qu'exécuter les commandes envoyé par la Maneuver (le cerveau tactique de la Squad) et lui envoyer des Messages sur leurs états respectifs et les résultats d'exécution de ces commandes.
Mais si le lieutenant vient à mourir, la Squad est alors détruite et les sbires la formant deviennent livrés à eux-même.

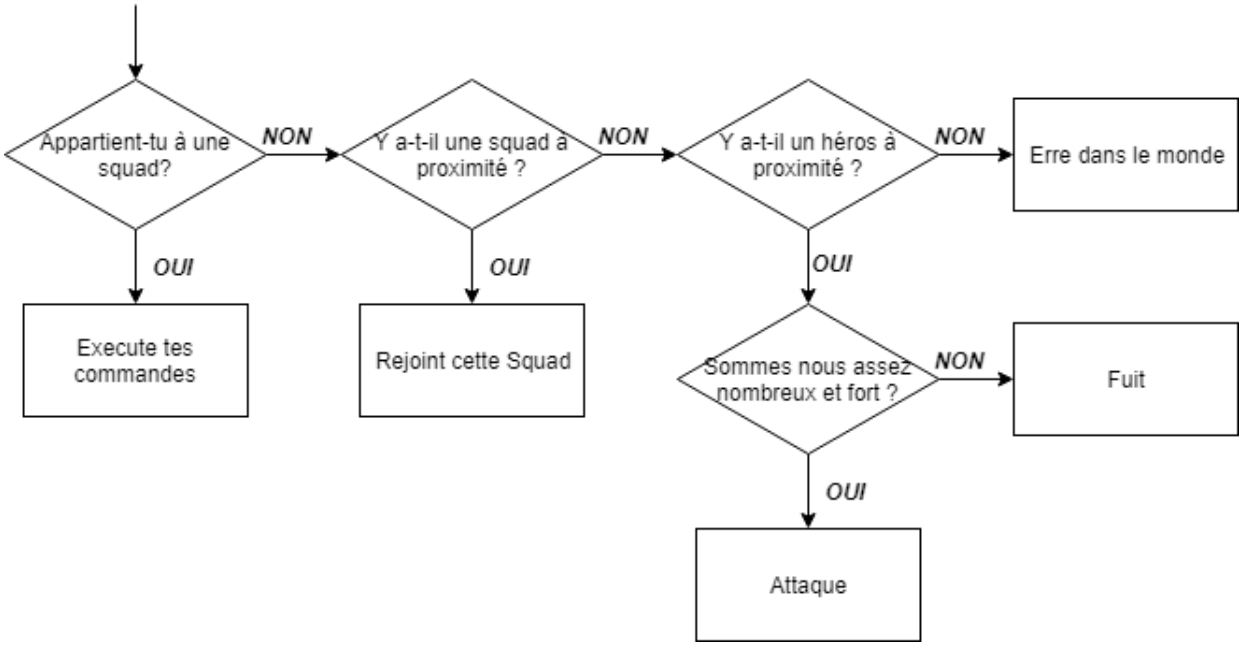


FIGURE 3 – Diagramme de comportement d'un sbire

Si un sbire n'a plus de Squad, il va en premier vérifier si il existe une autre Squad à proximité, et si c'est le cas, ira la rejoindre.
Si il n'y a pas de lieutenant à proximité, le sbire va regarder si il y a un ennemi à proximité (un héros). Si c'est le cas, il va vérifier si la somme des points de vie associée à ceux des sbires de son entourage est supérieur à celui-ci. Si c'est le cas, il décide d'attaquer le héros. Sinon, il fuit dans la direction opposé au héros jusqu'à une distance jugée appropriée et qu'il ne le voit plus dans son champ de vision.
Si ni Squad ni héros ne sont à proximités, le sbire va alors errer dans le monde, en espérant tomber sur une Squad qu'il pourra rejoindre.

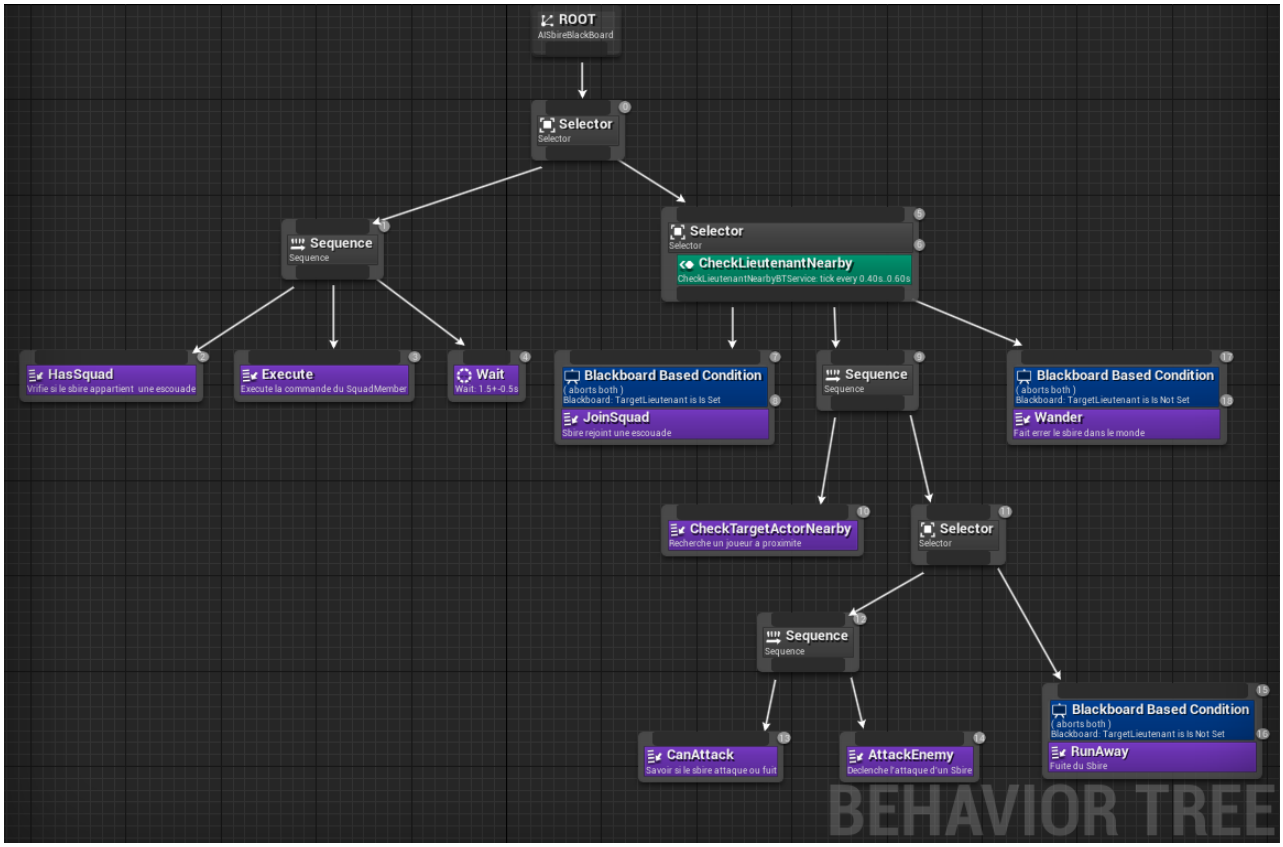


Figure 4 – Arbre de Comportement d'un Sbire

Si notre sbire possède une Squad, il sera dans le nœud Execute et y restera (retourne un état *InProgress*). Dans ce nœud, il exécute les commandes envoyés par le squadManager de façon automatique, son calcul de comportement est alors efficace est entièrement contrôlé par la Squad. Mais si le lieutenant de sa Squad vient à mourir, et donc la Squad avec, au moment de sa destruction, une commande EndCommand est envoyé à tous ses membres, étant la seule commande échouant un exécute (retourne un état *Failed*). Ainsi, un sbire se dirigera dans la seconde partie de son BehaviorTree, ou il réfléchira par lui-même.
Nous avons conservé un comportement très simpliste mais intéressant pour permettre à un maximum de sbires de pouvoir interagir indépendamment d'une Squad dans notre monde.

Quand une Squad vient à mourir, nos sbires attendent un court moment (nœud Wait) avant de prendre une décision, pour simuler une désorientation suite à la perte de leur lieutenant. Ce temps n'est pas le même pour chaque sbire afin que le joueur observe bien que chaque sbire agit maintenant par leur propre volonté, et donc plus au même moment.

Un service analysant si une Squad est présente à proximité du sbire est appelé régulièrement, car rejoindre une nouvelle Squad est la priorité de notre sbire, et si il en existe effectivement une dans son champ de vision, il arrêtera d'errer ou de fuir pour aller la rejoindre.

2 Différences entre la version finale et la présentation intermédiaire

2.1 Système de Squad

L'architecture initiale était la suivante :

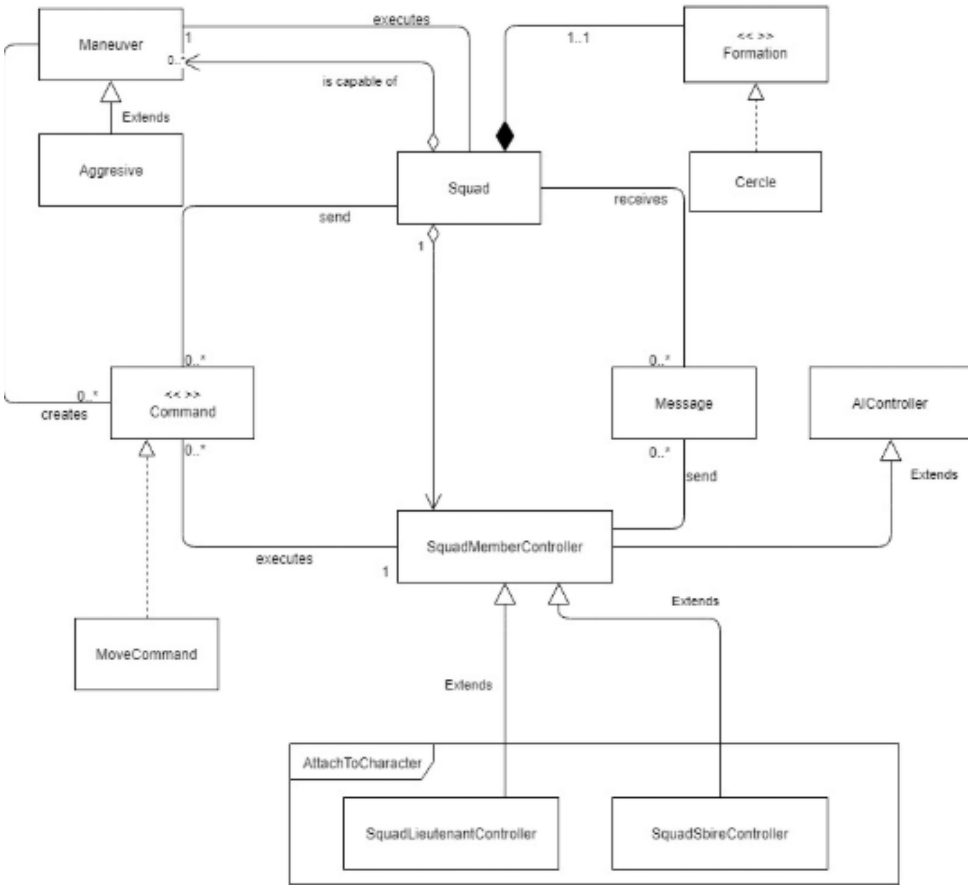


FIGURE 5 – Architecture finale de Squad

Initialement nous n'avions pas de séparation entre le cerveau et le corps de la Squad. Tout était mélangé. Suite à une discussion, nous avons séparé ces deux éléments. Nous avons également transféré la formation dans la manoeuvre pour nous permettre de spécialiser bien plus efficacement les différentes manoeuvres.

2.2 Manoeuvre des Squad

Une grosse différence qu'il y avait entre la version intermédiaire et la version finale était que les commandes des sbires étaient recalculés et ré-envoyés à toutes les frames car nous n'avions pas encore de systèmes de Message ! Ainsi les pathfindings étaient calculés à toutes les frames pour tous les personnages ce qui n'étaient pas idéal ... du tout ! A cause de cela nous ne pouvions pas avoir plus de 5 sbires en même temps en jeu ! Nous

en avons aujourd'hui plus de 180 sans que cela ne pose problème grâce à de nombreuses optimisations à la fois dans l'envoie des commandes et les commandes elle-mêmes qui ne rafraîchissent leur chemins qu'à un certain framerate distribué uniformément pour ne pas que tous les NPC rafraîchissent en même temps.

3 Conclusion

Nous avons voulu utiliser des Squads dans notre jeu, car ce système permettait de contrôler un très grand nombre d'IA, à un coût raisonnable. En effet, en regroupant des ensembles de NPC, on peut les faire agir comme un seul, et donc, utiliser un unique cerveau pour plusieurs unités.