LOG8235 – Agents intelligents pour jeux vidéo

Hiver 2024

TP No. 3

Numéro d'équipe : 04

1951042 - Abderrahim Laribi

2057242 - Ayoub Chihab

2077446 - Félix Lamarche

2307835 - Léo Valette

Soumis à :

Daryl Barampaze

14 Avril 2024

Ce qui suit est la liste des fichiers et des fonctions modifiées pour section du travail à accomplir :

1. Utilisation d'un behavior tree

Nous prenons les décisions des agents et les faisons bouger dans un behavior tree. Un service (**BTService_UpdatePlayerLoS**) gère l'état du joueur dans le blackboard. La fonction **UpdateLoSOnPlayer()** du fichier **SDTAIController.cpp** roule un algorithme plus lourd pour établir l'état du joueur (cette fonction est distribuée sur le temps dans le budget fixe attribué aux agents).

Il y a plusieurs **Nodes** de tasks créer:

- **BTTask_IsInPursuitGroup** pour savoir si l'agent est dans le groupe de poursuite.
- **BTTask_IsPlayerInLoS** pour savoir si le joueur est visible par l'agent.
- **BTTask_IsPlayerPoweredUp** pour savoir si le joueur a son pouvoir d'activé.
- **BTTask_SetCollectibleAsTarget** pour trouver le meilleur collectible à chasser en tant que cible.
- **BTTask_SetFleeLocAsTarget** pour poser le meilleur emplacement de fuite comme cible à atteindre.

Le fichier **STDAIController.cpp** est responsable de beaucoup des comportements des agents avec plusieurs fonctions:

- **UpdateLoSOnPlayer():** Décide si le joueur est visible par l'agent, et met à jour la dernière position du joueur au groupe de poursuite. Décide également de la position de poursuite du joueur.
- **SetClosestCollectibleAsTarget():** Trouve le collectible le plus près de l'agent et pose sa cible à atteindre à la position de ce collectible.
- **SetBestFleeLocationAsTarget():** Trouve la meilleure destination de fuite pour l'agent et pose sa destination à cette position.

Nous avons fait beaucoup de changements au code originel pour pouvoir mieux utiliser les behavior trees en séparent plus les différentes fonctionnalités pour pouvoir plus simplement les combiner pour créer les comportements de l'agent.

2. Création de groupe de poursuite

Ajout d'une classe **AIAgentGroupManager**, qui nous permet d'ajouter des agents IA au groupe ou les enlever du groupe en utilisant les méthodes **RegisterAIAgent** et **UnregisterAIAgent**. La classe fournit une méthode **Disband** pour enlever tous les agents IA du groupe si jamais ils perdent la trace du joueur. Elle fournit aussi une méthode **DrawDebugGroup** qui dessine des sphères au-dessus de la tête des agents IA pour représenter leurs états.

3. Comportement de poursuite en groupe

La même classe **AAIAgentGroupManager** nous permet de stocker le LKP du joueur en utilisant **UpdatePlayerLKP** et aussi de vérifier si un agent IA se trouve proche du LKP pour

investiguer avec la méthode **AgentAtLKP**. La méthode **DrawDebugGroup** permet aussi de dessiner une sphère au niveau du LKP du joueur.

- **void RegisterAIAgent(ASDTAIController *aiAgent):** Ajoute un agent au groupe de de poursuite
- **void UnregisterAIAgent(ASDTAIController *aiAgent):** Enlève un agent au groupe de de poursuite
- **void CheckIfDisband():** Vérifie que plus personne n'a de ligne de vue sur le joueur avant de Disband.
- **bool HasGroupLoSOnPlayer():** Retourne si il reste un agent avec une ligne de vue sur le joueur.
- **void UpdatePlayerLKP(FVector lkp):** Change le LKP.
- **bool AgentAtLKP():** Vérifie si le joueur a atteint la LKP.
- **FVector GetPlayerLKP():** Un getter.
- **void Disband():** Enlève tous les agents du groupe de poursuite.
- **FVector CalculateAgentPosition(int i, float radius, float angleBetweenAgents, float agentRadius):** Calcul la position en cercle des agents.
- **FVector CalculateOffsetVector(FVector agentPosition, float agentRadius):** Calcul la translation de la position de la boule en cas d'achalandage ou de mur.
- **FVector HandleLineTrace(FVector agentPosition, float agentRadius):**
- **FVector HandleAgentProximity(FVector agentPosition, float agentRadius):** Calcul la position en cercle des agents sans collision entre eux.
- **FVector HandleWallProximity(FVector agentPosition, float agentRadius):** Vérifie si la position est dans le mur.
- **FVector ProjectToNavigation(FVector agentPosition, float agentRadius)**: Vérifie la position dans l'espace navigable.

4. Attribution d'un budget fixe pour la mise à jour des agents

Création d'un "LoadBalancerManager" qui va permettre de lancer l'update des agents dans un temps fixe. Les fichiers sont LoadBalancerManager.cpp et LoadBalancerManager.h.

- **GetInstance()** : Permet aux agents de récupérer l'instance du manager.
- **Initialize()** : Initialisation du Manager.
- **Destroy()** : Delete de l'instance du Manager.
- **RegisterNPC(AActor* npcCharacter)** : Enregistre un agent dans le manager.
- **UnregisterNPC(AActor* npcCharacter)** : Retire un agent du manager.
- **RegisterPlayer(AActor* player)** & **UnregisterPlayer(AActor* player)** : Ajoute ou retire un player au manager.
- **GetPlayer()** : Récupère le joueur lié au manager.
- **TickWorld(UWorld* World, ELevelTick TickType, float DeltaSeconds)** : Fonction appelé à chaque frame qui permet de lancer la fonction UpdatePlayerInteraction(DeltaSeconds) des agents tout en comptant le temps total écoulé afin de limiter le temps par frame et de décaler l'update des agents restant aux frame suivantes. Affiche aussi les infos du groupe de poursuite.

Modification des fichiers SDTAIController.cpp, SDTAIController.h, SoftDesignTrainingMainCharacter.cpp et SoftDesignTrainingMainCharacter.h pour ajouter les fonctions **BeginPlay()** & **EndPlay(const EEndPlayReason::Type EndPlayReason)** qui permettent de créer et de détruire l'instance du manager et d'enregistrer les agents et le joueur.

Pour la mise à jour dynamique du tick rate en fonction de la visibilité des acteurs à la caméra, nous avons modifié les fichiers suivants : SDTAIController.cpp, SDTAIController.h.

- **UpdateIsActorOnCamera()** : Modifie la variable IsActorOnCamera qui permet de savoir si l'agent est visible à la caméra.
- **UpdateTickRateMovementComponent()** : Modifie le tick rate du *UCharacterMovementComponent* de l'agent en fonction de sa visibilité à la caméra et de sa présence dans le groupe de poursuite. Ce component est responsable d'une partie importante du temps de calcul lié aux agents.
- **UpdateTickRateSKinMeshComponent()** : Même chose pour le *SkeletalMeshComponent*.
- **Tick(float deltaTime)** : On déplace cette fonction du fichier SDTBaseAIController.cpp dans la classe fils pour pouvoir utiliser les nouvelles fonctions. Dans le tick, on met à jour la visibilité de l'agent à la caméra et on modifie en fonction le tick rate des différents components. On n'affiche le Navigation Path que si l'agent est dans la caméra, ça permet de gagner beaucoup de performance, car la fonction d'affichage du chemin est très lourde en temps d'exécution (sur ma machine).