Pathfinding et FSM

par Pierre-Olivier Roy, William Leduc et Raphaël Duhamel
remis le 21 février

Changement apporté au code initial:

Tout d'abord, nous avons retiré l'enum qui contenait les différents états, considérant les enjeux de développement futur que ça peut engendrer. Nous l'avons remplacé par une liste contenant les différents états. Nous avons ensuite modifier la machine à état initial pour quelle traverse cette liste afin d'appeler la logique de l'état actuel.

Structure des états :

Les états obéissent à une interface commune. Cette interface permet de standardiser le fonctionnement de chaque état. L'interface permet également de faciliter la création de nouveaux états. Le projet est devenu bien plus simple à développer à la suite de cet ajout.

Essentiellement, chaque état possède :

- Une méthode pour vérifier si les conditions d'entrées sont valides.
- Une méthode qui s'exécute à l'entrée dans l'état.
- Une méthode qui s'exécute à chaque update si c'est l'état sélectionné.
- Une méthode qui vérifie si les conditions de sorties sont valides
- Une méthode qui s'exécute à la sortie de l'état.

Différents états:

RoamingState: Le personnage patrouille d'un point à un autre. La transition hors de cet état se produit si le joueur est détecté à proximité. Dans cet état, le personnage non-joueur traverse une liste de destinations une à une. Il se redirige vers le dernier point choisi lorsqu'il entre dans cet état.

PursuitState: Le personnage poursuit le joueur. Cet état est activé si le joueur est proche et est quitté soit si le joueur n'est plus détecté, soit après une durée de poursuite déterminée. Le personnage non-joueur ne détecte pas le joueur s'il y a un mur entre les deux.

IdleState: Le personnage est inactif. Cet état est généralement un état de repos ou d'attente et peut servir de transition entre d'autres états plus actifs. Le personnage non-joueur reste sur place et ne peut plus interagir avec le joueur tant que le délai d'attente n'est pas terminé.

Transition d'état:

De Roaming à Pursuit: Si le personnage est en train de patrouiller et détecte le joueur à proximité *(PlayerIsNear() retourne true)*, il peut passer à l'état de poursuite.

De Pursuit à Idle: Si le personnage non-joueur poursuit le joueur mais perd sa trace ou atteint la durée maximale de poursuite, il passe à l'état inactif en se mettant en mode pause *(SetIdle(true))*.

De Idle à Roaming ou Pursuit: L'état inactif peut transiter vers la patrouille ou la poursuite selon que le joueur est détecté à proximité ou non. Si le joueur n'est pas détecté et que la durée d'inactivité est écoulée, le personnage peut retourner à la patrouille.

Recherche de chemin (Pathfinding) :

Pour commencer nous avons ajouté un Navmesh Surface à notre niveau. Cette objet permet de lire les éléments du décor de la scène et de calculer les chemins possibles au travers du décor.

Nous avons ensuite ajouté un Navmesh Agent à nos personnages, ce qui leur permet d'interagir avec le Navmesh Surface. Nous avons également ajouté aux personnages des scripts qui interagissent directement avec le système Navmesh de unity, permettant au personnage joueur de se déplacer au travers du Navmesh à l'aide de clic et au personnages non-joueurs de se déplacer seul au travers du parcours.

Nous avons ajouté à certains endroits de la scène des Navmesh Link, permettant aux personnages de sauter au-dessus de certains obstacles de la scène. Avant notre refactorisation de la scène, les personnages devaient sauter à l'aide de l'entrée de touche. Suite au changement, les personnages sautent automatiquement lorsqu'ils arrivent à ses Navmesh Link.

Par défaut, Navmesh utilise le A* pour la recherche de chemin. Nous n'avons rien modifier à cet égard

Gestionnaire de version :

Nous avons utilisé Sourcetree pour avoir une interface graphique plus facile d'utilisation que Git. À l'aide de Sourcetree, nous avons commenté chaque push afin d'avoir un suivi significatif des changements et pouvoir retracer l'origine des bugs.