Documentation du Projet IA

du 14/02/2025 au 21/02/2025

1. Introduction

Présentation du projet

Ce projet vise à améliorer le comportement des ennemis dans un jeu vidéo en intégrant diverses techniques d'intelligence artificielle (IA). L'objectif est d'optimiser leurs réactions et leurs déplacements en utilisant des algorithmes avancés. Nous avons implémenté **FSM (Finite State Machine)**, **Pathfinding A***, **Behavior Trees** et **GOAP (Goal-Oriented Action Planning)**.

Objectifs

- Améliorer le réalisme du comportement des ennemis.
- Implémenter un système de déplacement intelligent évitant les obstacles.
- Optimiser les performances du jeu pour garantir une exécution fluide.

---

2. Technologies et outils utilisés

Technologies

- **Langage** : C++(17)
- **Bibliothèque graphique** : SFML
- **Outils complémentaires** : GitHub

Structure du code

- Séparation des différentes parties du code à l'aide de fichiers d'en-tête (`.hpp`).
- Utilisation de principes de **programmation orientée objet** tels que l'héritage et le polymorphisme pour garantir un code modulaire et extensible.
- Organisation du code avec des **namespaces** pour éviter les conflits.

---

3. Fonctionnement de l'IA

3.1 Finite State Machine (FSM) : Gestion des états

La FSM permet aux ennemis d'adopter différents comportements en fonction des situations rencontrées. Un ennemi peut être dans l'un des états suivants :

- **Patrouille** : Déplacement sur un trajet défini.
- **Chasse** : Détection et chasse du joueur.
- **Attaque** : Combat à courte portée.
- **Fuite** : Retraite lorsque l'ennemi n'a plus de vie.

L'ennemi change d'état en fonction des conditions établies (distance avec le joueur, points de vie).

3.2 Pathfinding A* : Calcul du chemin optimal

L'algorithme A* est utilisé pour déterminer le chemin le plus efficace vers une destination en prenant en compte les obstacles du terrain. Grâce à cette approche, les ennemis peuvent naviguer dans l'environnement de manière fluide et réaliste sans rester bloqués contre des obstacles.

3.3 Behavior Trees : Prise de décision avancée

Les arbres de comportement permettent une gestion hiérarchique des actions des ennemis. Chaque décision repose sur une série de tests conditionnels structurant leur comportement.

Exemple : Un ennemi blessé pourra choisir entre continuer le combat ou chercher un abri pour se soigner, en fonction de sa situation actuelle.

3.4 GOAP : Planification des actions

Contrairement aux comportements rigides des FSM ou des Behavior Trees, GOAP permet aux ennemis d'adopter des stratégies plus dynamiques. Ils établissent un plan en fonction des ressources disponibles et des objectifs à atteindre.

Exemple : Un ennemi cherchant à attaquer le joueur peut d'abord récupérer une arme avant d'engager le combat.

---

4. Fonctionnalités avancées

Gestion des erreurs

L’intégration de **try-catch** permet d’éviter les crashs en gérant les exceptions de manière sécurisée. (try-catch dans GOAP)

Utilisation des pointeurs intelligents

L’utilisation de **std::unique_ptr** et **std::shared_ptr** assure une gestion efficace de la mémoire.

Utilisation du MultiThreading

L’utilisation du multiThreading permet une gestion efficace des ennemis.

---

5. Résultats et tests

- **Amélioration de la réactivité des ennemis** : Comportements plus réalistes et fluides.
- **Gestion des erreurs**

---

6. Conclusion

Points positifs

- Implémentation réussie des techniques d’IA.
- Optimisation des performances grâce à la gestion efficace de la mémoire.

Annexe

- **Dépôt GitHub** : https://github.com/dark-dylan-93220/ProjetIA
- **Outils utilisés** : SFML, Git, Visual Studio, notre cerveau

Dylan Hollemaert & Lucas Marcucci