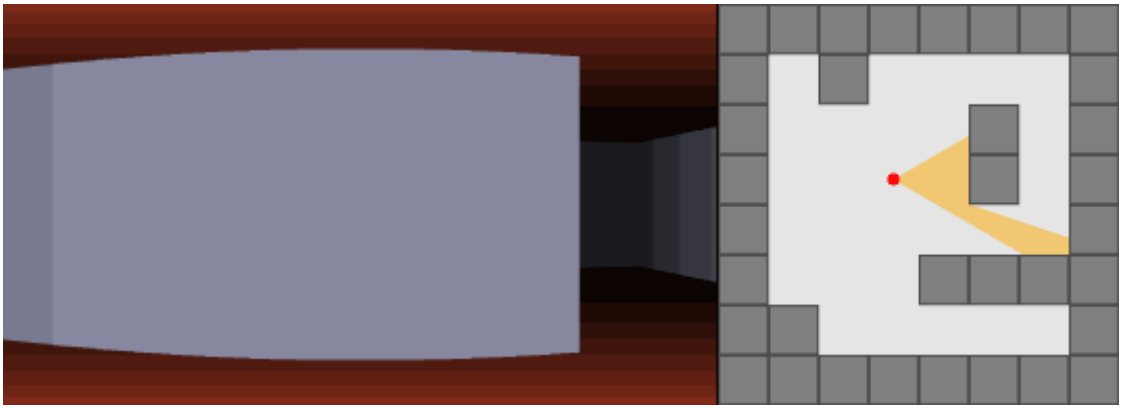
Document descriptif du projet tutoré
Sujet : Raycasting et Doom-like



Sommaire

Marcelin Ray - Ilyès Lounici - Cyprian Chailan - Geoffrey Gros
Vincent Thomas
BUT 3 | RA-IL 1

1. Contexte

À la suite des discussions avec notre tuteur, nous avons choisi d'orienter notre projet vers le développement d'un moteur graphique simplifié inspiré, sans chercher à produire un jeu complet.
L'objectif est avant tout d'explorer des aspects techniques concrets, notamment la gestion graphique, la simulation d'ennemis et de PNJ et la communication réseau entre plusieurs joueurs.

Cette approche permet d'approfondir la compréhension de plusieurs notions fondamentales:

- moteur graphique et rendu 3D ;
- intelligence artificielle de déplacement et de comportement ;
- synchronisation d'état entre clients dans un environnement réseau ;
- la création de conversations dans le cadre d'une mission.

2. Objectifs et périmètre

Objectifs principales:

Nous allons réaliser un prototype afin d'explorer et résoudre les différents défis dans la réalisation d'un jeu. Nous couvrirons tous les aspects de la réalisation du moteur graphique au PNJ intelligent afin de les rendre plus difficiles à prédire ou bien réaliser des conversations avec des png.

Les principaux points que nous souhaitons réaliser sont :

- un moteur graphique utilisant raycasting ;
- le complexifier avec BSP ;
- génération automatique de labyrinthe ;
- génération d'ennemies plus au moins intelligent ;
- création d'un système multijoueur.

3. Étude de l'existant

Avant de définir nos choix techniques, nous avons mené une étude de l'existant afin d'identifier les technologies et les méthodes déjà utilisées dans des projets similaires.

3.1. Références historiques

Les premiers moteurs de rendu 3D, tels que ceux utilisés dans *Wolfenstein 3D* (1992) et *Doom* (1993), ont profondément marqué l'histoire du jeu vidéo.

- *Wolfenstein 3D* reposait entièrement sur le **raycasting**, une technique permettant de simuler la 3D à partir d'un plan 2D ;
- *Doom*, quant à lui, a introduit la gestion de structures plus complexes et l'optimisation du rendu via la **partition binaire de l'espace (BSP)**.

Ces approches simples mais ingénieuses ont ouvert la voie aux moteurs modernes.

3.2. Projets open-source et ressources actuelles

Plusieurs projets open-source disponibles sur GitHub reprennent ces principes, notamment :

- https://github.com/AXDOOMER/mochadoom : dans ce dépôt, le propriétaire a recréé le jeu Doom en Java.

3.3. Analyse comparative

Critère	Wolfenstein 3D	Doom	Notre projet
Type de rendu	Raycasting	Raycasting + BSP	Raycasting + BSP
Génération procédurale	Non	Non	Oui (labyrinthe aléatoire)
Multijoueur	Non	Partiel (LAN)	Oui (client-serveur & P2P)
IA ennemis	Très simple	Moyenne	Évolutive (IA basique)
Objectif	Jeu complet	Jeu complet	Prototype technique

Cette analyse montre que notre projet s'inspire des bases historiques tout en y intégrant des éléments modernes tels que la génération procédurale et le multijoueur.

4. Étude technique

4.1 Algorithme de rendu graphique

Nous utiliserons le raycasting pour la génération graphique 3D optimisée avec BSP. Cela nous permettra d'optimiser les ressources prises par notre jeu. Nous y intégrerons aussi l'ajout de texture afin de rendre le jeu plus beau.

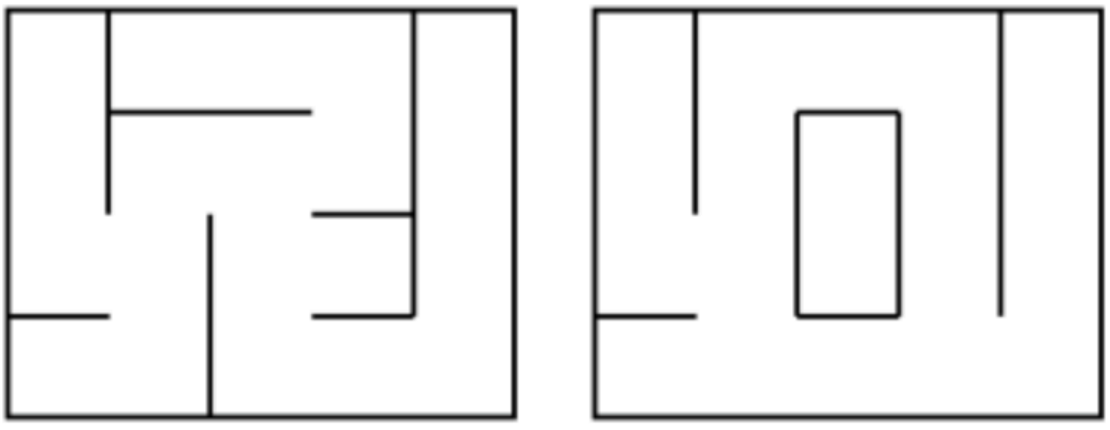
Le raycasting est une technique de rendu utilisée dans les jeux vidéo pour simuler la 3D à partir d'un environnement 2D. Elle repose sur le principe d'envoyer des rayons depuis le point de vue du joueur pour déterminer ce qu'il peut voir à l'écran.

Le BSP, ou Binary Space Partitioning, est une technique informatique utilisée principalement dans les jeux vidéo 3D et les moteurs de rendu pour organiser et gérer efficacement l'affichage des objets dans un espace tridimensionnel. Elle permet de déterminer rapidement quels éléments d'une scène sont visibles ou non par le joueur, afin d'optimiser les calculs graphiques. Dans notre cas, il servira à savoir si on doit charger un tronçon ou non.

4.2 Algorithme de génération procédurale

Pour la génération de labyrinthe, on partira sur deux algorithmes Prim et Kruskal à implémenter afin d'explorer deux façons de faire des labyrinthes dit parfait
Un labyrinthe c'est un ensemble de cellules qui est relier les unes aux autres, si ces liens sont unique alors la c'est un labyrinthe parfait.

Exemple :



Labyrinthe « parfait »

Labyrinthe « imparfait »

4.3. Intelligence artificielle

Pour l'ia des ennemis, nous réaliserons des **algorithmes de parcours de graphe** (BFS, Dijkstra), combinés à des comportements conditionnels simples (patrouille, poursuite, fuite).
La mise en place d'ennemis complètement autonomes c'est-à-dire en fonction des actions jouées, elle décidera toute seule si c'est le meilleur moment pour attaquer le joueur ou bien, il faut se repositionner afin d'attaquer le joueur sous un autre angle. Cela pourrait se faire soit avec un modèle entraîné par nos soins ou bien en utilisant un modèle déjà entraîné que nous devrions sélectionner.

Pour les PNJ on pourrait appeler un modèle déjà entraîné comme Mistral. il suffirait juste de lui donner un prompt afin qu'elle réponde selon une situation précise. Le joueur pourra par exemple, lui demander son chemin ou bien réaliser une quête générer par l'ia.

4.4. Système multijoueur

Nous allons réaliser deux systèmes de multijoueurs avec le système peer to peer et le système client-serveur.

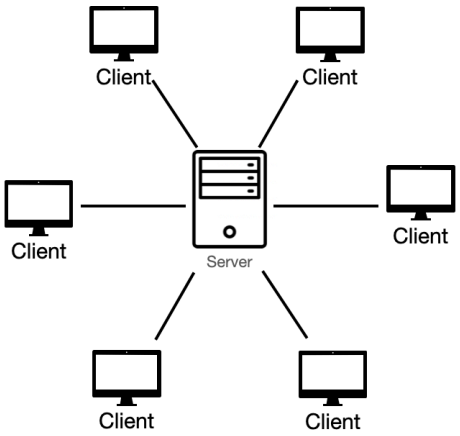
Dans le modèle client-serveur, les joueurs ne sont pas directement connectés entre eux. Chaque joueur utilise un client, c'est-à-dire le programme du jeu installé sur sa machine, tandis qu'un serveur central s'occupe de gérer l'ensemble de la partie. Le serveur est considéré comme l'autorité principale du jeu : c'est lui qui détient l'état officiel du monde virtuel et qui décide du déroulement des actions.

Lorsqu'un joueur effectue une action — par exemple avancer, sauter ou tirer —, son client envoie une requête au serveur pour l'en informer. Le serveur analyse et valide cette action en fonction des règles du jeu. Il met ensuite à jour la position du joueur, l'état de la partie, et détermine les conséquences de cette action (par exemple, si un tir touche une cible ou non).

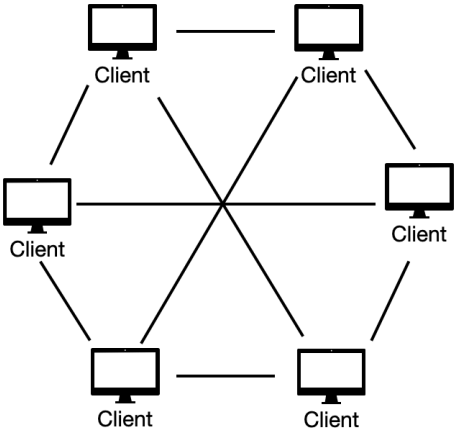
Le système pair-à-pair, aussi appelé peer-to-peer (P2P), est un mode de connexion où les joueurs communiquent directement entre eux, sans passer par un serveur central. Dans ce modèle, chaque joueur est à la fois client et serveur : il envoie et reçoit des informations aux autres participants afin de synchroniser la partie.

Lorsqu'un joueur effectue une action, comme se déplacer ou tirer, son jeu transmet directement l'information aux autres machines connectées. Chacun des ordinateurs reçoit alors ces données et met à jour sa propre version du jeu pour refléter ce qui vient de se passer. Ainsi, la cohérence du monde virtuel repose sur la bonne communication entre tous les joueurs.

Pour réaliser ces deux solutions, nous utiliserons les sockets de java afin d'assurer la transmission des données entre les clients pour P2P et les clients et le serveur.



Client Server Architecture



P2P Architecture

5. Technologies utilisées

Le projet sera développé en Java, langage que nous maîtrisons et qui offre une bonne gestion des threads et des sockets.
L'interface graphique sera construite à l'aide de la bibliothèque Swing, qui permet un contrôle précis du rendu et des événements utilisateurs.

Domaine	Technologie	Rôle
Langage	Java	Développement principal
Graphisme	Swing	Rendu 2D, 3D et gestion des entrées utilisateur
Réseau	Java Sockets	Communication client-serveur et P2P
IA	Algorithmes de graphes	Gestion du déplacement des ennemis
Génération procédurale	Prim & Kruskal	Construction automatique de niveaux
Architecture	MVC & Threads	Séparation claire des modules