Moteur Doom-like basé sur le raycasting

Présentation de l'analyse du projet

Projet tutoré - Soutenance
CHAILAN CYPRIAN - GROS GEOFFREY - LOUNICI ILYES - RAY MARCELIN
Novembre 2025

Glossaire

Arbre
Structure hiérarchique composée de nœuds reliés sans boucle, partant d'une racine et se divisant en sous-nœuds.

Algorithmes de pathfinding
Famille d'algorithmes qui cherchent un chemin entre un point de départ et un point d'arrivée en évitant les obstacles (dans notre cas : trouver un chemin dans le labyrinthe pour les ennemis)

Espace convexe
Espace où, pour n'importe quels deux points, le segment qui les relie reste entièrement à l'intérieur de cet espace.

Optimalité asymptotique (RRT*)
Propriété d'un algorithme qui, quand on lui laisse de plus en plus de temps (plus d'itérations), **se rapproche d'un chemin optimal**. Dans RRT*, plus on ajoute de points, plus le chemin devient court et "propre"

Résumé exécutif

Moteur 3D

Développer deux moteurs pseudo-3D : raycasting et BSP pour rendre un labyrinthe immersif

IA & Ennemis

Concevoir des comportements variés (patrouille, poursuite, fuite) et des algorithmes de pathfinding

Multijoueur P2P

Synchroniser les actions des joueurs sans serveur central en utilisant des sockets Java

PNJ & Quêtes

Ajouter des dialogues et quêtes via une API d'IA pour enrichir le récit

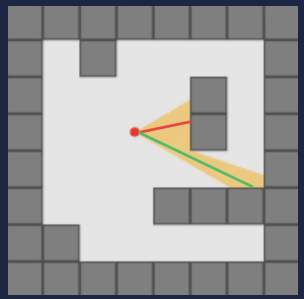
Problématique et objectifs

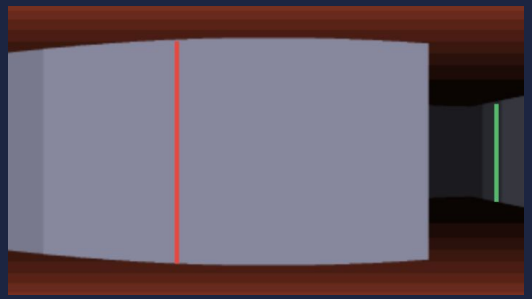
- Concevoir deux moteurs 3D maison (raycasting et BSP)
- Gérer un labyrinthe complexe : génération, collisions précises et mise à jour des graphismes en vue subjective
- Mettre en place une IA crédible pour les ennemis (behavior tree, pathfinding A* / RRT*)
- Assurer la cohérence du monde en multijoueur P2P malgré la latence, les déconnexions et l'absence de serveur central
- Intégrer des PNJ pilotés par une API d'IA sans perdre le contrôle sur le scénario du jeu

Moteurs graphiques : Raycasting et BSP

Raycasting

• Projeter un monde 2D sur un écran 3D en lançant des rayons autour du joueur.
• Chaque rayon mesure la distance jusqu'au mur le plus proche ; la hauteur du mur dépend de cette distance.
• Utilisation de l'algorithme DDA pour traverser la grille sans rater de murs.





BSP

• Découper l'espace en régions convexes en séparant les murs « devant » et « derrière ».
• Organiser ces régions dans un arbre binaire (front/back).
• Pour le rendu, on parcourt l'arbre.

Pourquoi utiliser l'Algorithme DDA ? (Digital Differential Analysis)

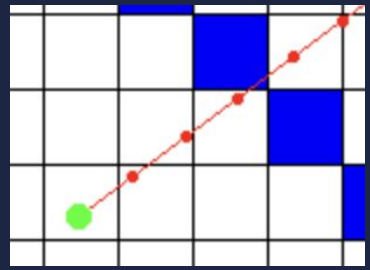
Approche naïve

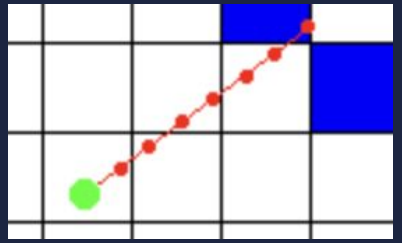
On avance le rayon par petits pas réguliers dans sa direction.

À chaque pas, on teste : *« suis-je dans une case mur ? »*

Si le pas est trop grand → le rayon peut sauter par-dessus un mur, surtout au niveau des coins.

Si on réduit le pas pour éviter ça, le nombre de pas explose → performances catastrophiques.





Pourquoi utiliser l’Algorithme DDA ? (Digital Differential Analysis)

Solution DDA

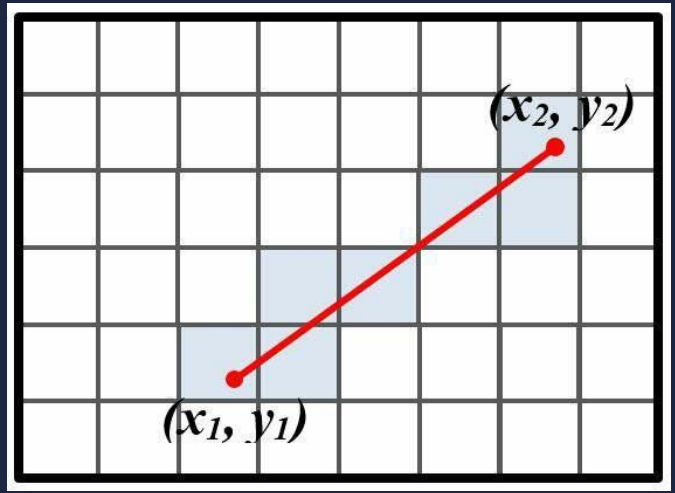
On profite du fait que le monde est une grille de blocs.

Au lieu d’avancer d’une distance fixe, on saute de frontière de case en frontière de case.

À chaque itération, on calcule si le rayon touche d’abord une ligne verticale ou horizontale de la grille.

On passe à la case voisine correspondante et on teste si c’est un mur.

On s’arrête dès qu’on entre dans une case mur : aucun mur n’est sauté.



BSP (Binary space partitioning)

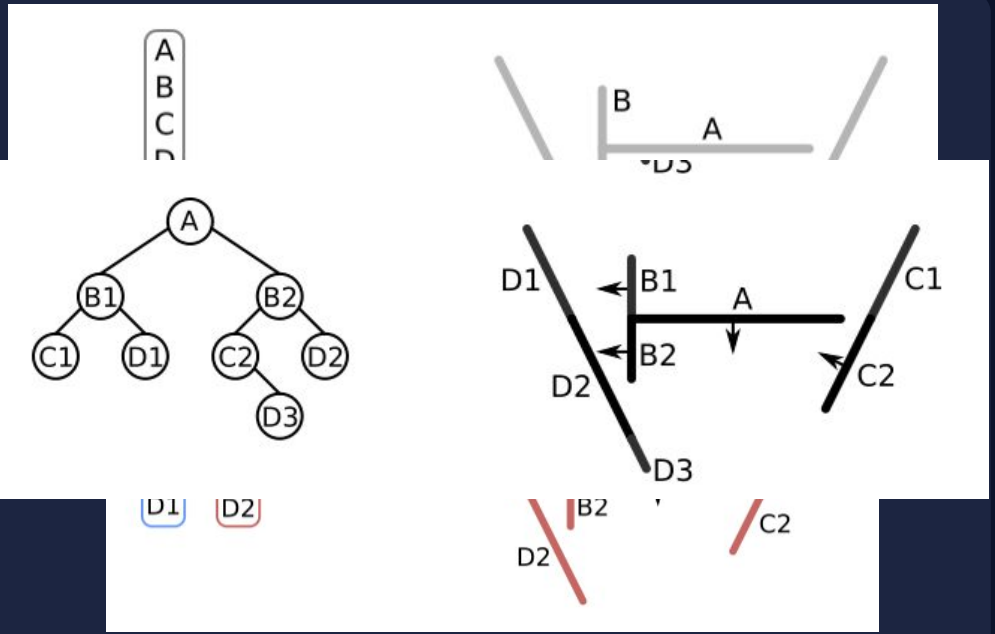
Découpage de la carte

On va découper la carte afin d'obtenir uniquement des espaces convexes.

On part d'un mur, ici A, et on découpe la carte en deux espaces.

Tout ce qui est derrière le mur A est placé dans le nœud gauche, tout ce qui est devant dans le nœud droit. Si un mur est coupé par A, on le divise en deux pour pouvoir les classer.

On continue récursivement jusqu'à avoir parcouru tous les murs.



BSP - Rendre l’image

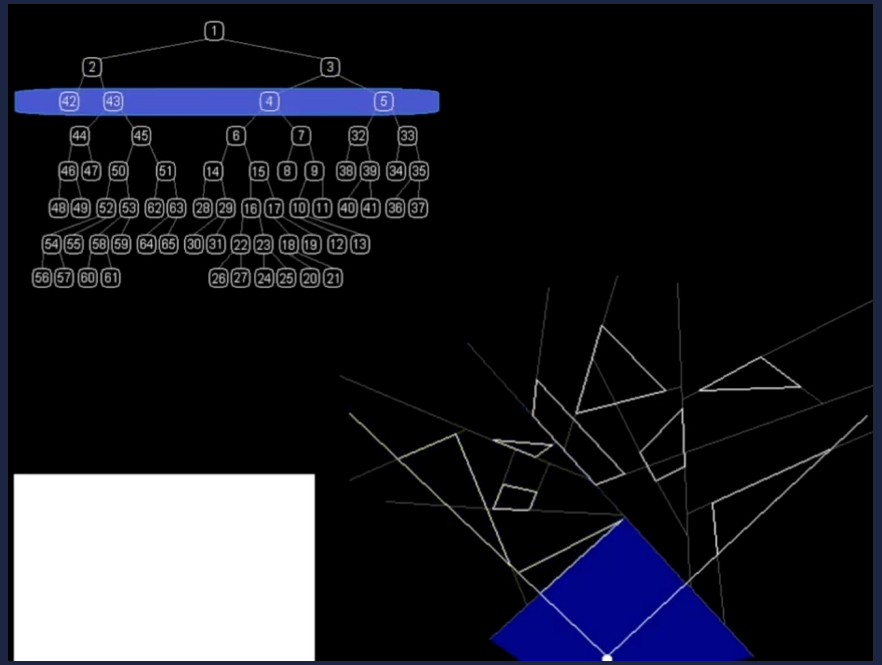
Trouver où est le joueur

On n’a plus besoin de la carte, on utilise seulement l’arbre.

Dans un premier temps, on cherche où se trouve le joueur :
On part de la racine :

- si le joueur est devant la racine, on va dans le sous-arbre droit ;
- sinon, dans le sous-arbre gauche.

On continue récursivement jusqu’à atteindre le nœud feuille où se trouve le joueur (c’est-à-dire l’espace convexe).
Ici, le joueur est dans le nœud 42.



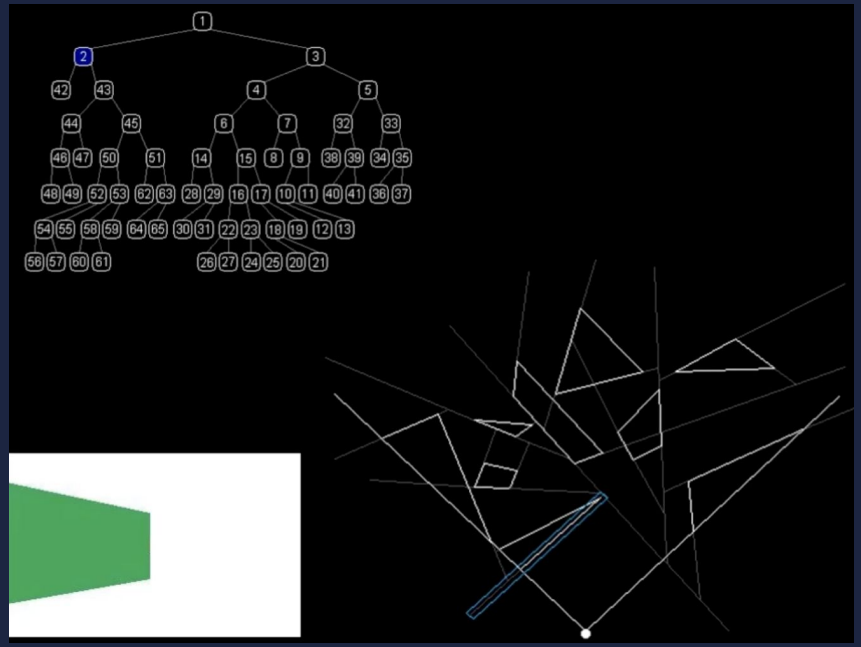
BSP - Rendre l’image

Rendre les murs un par un.

Maintenant, on parcourt les nœuds récursivement en partant du nœud où se trouve le joueur.

L’algorithme déterminant l’ordre de parcours des nœuds reste encore un peu flou pour nous pour l’instant, mais l’important est de parcourir l’arbre et d’afficher les murs que le joueur voit.

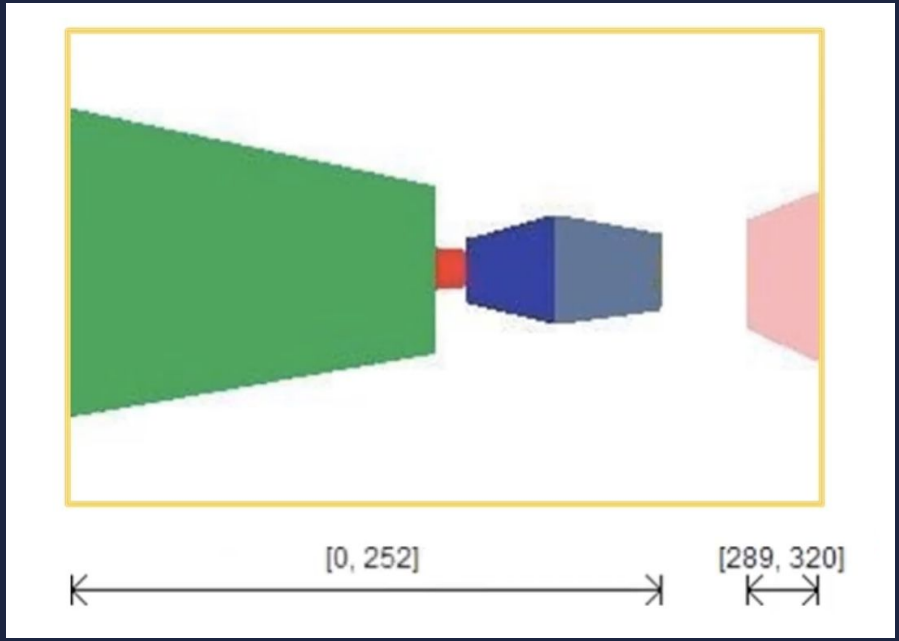
Ici, le premier nœud est le 2, situé au-dessus du nœud du joueur, et il représente le mur vert.



BSP - Rendre l’image

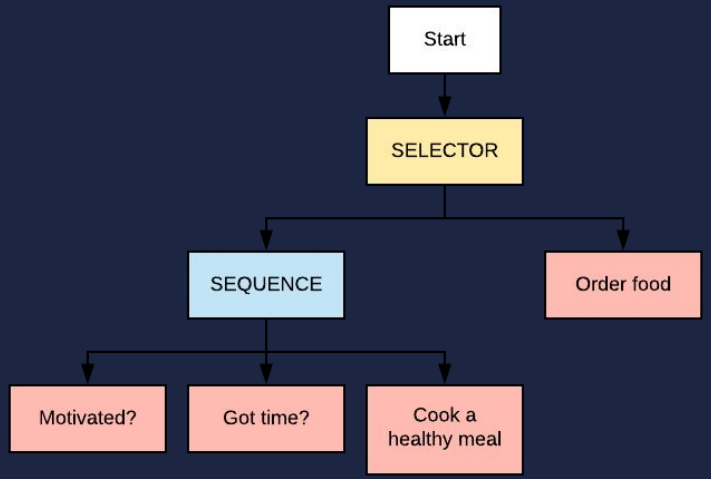
Quand s’arreter ?

On stocke la taille de l’écran que l’on dessine au fur et à mesure, afin que les murs situés derrière puissent ignorer les parties déjà rendues.



IA des ennemis et pathfinding

Arbre de comportement (behaviour tree)



Pathfinding

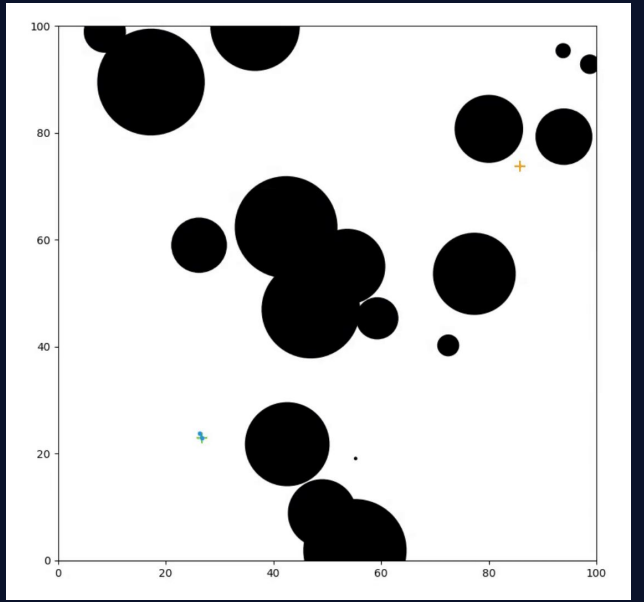
Algorithme	Utilisation
A*	Recherche du chemin le plus court avec heuristique
RRT*	Planification exploratoire optimisée pour espaces continus

Algorithme RRT*

Principe

• But : trouver un chemin « pas trop long » dans un espace continu en évitant les obstacles (utile pour des déplacements plus complexes que sur une simple grille).

• RRT* construit progressivement un arbre de positions à partir du point de départ.



Personnages non-joueurs (PNJ)

Rôle narratif
Certaines rencontres livrent des indices sur l'histoire et les secrets du labyrinthe...

Quêtes et récompenses
Des missions secondaires variées pourrait rendre l'expérience plus vivante en offrant par exemple des objets rares

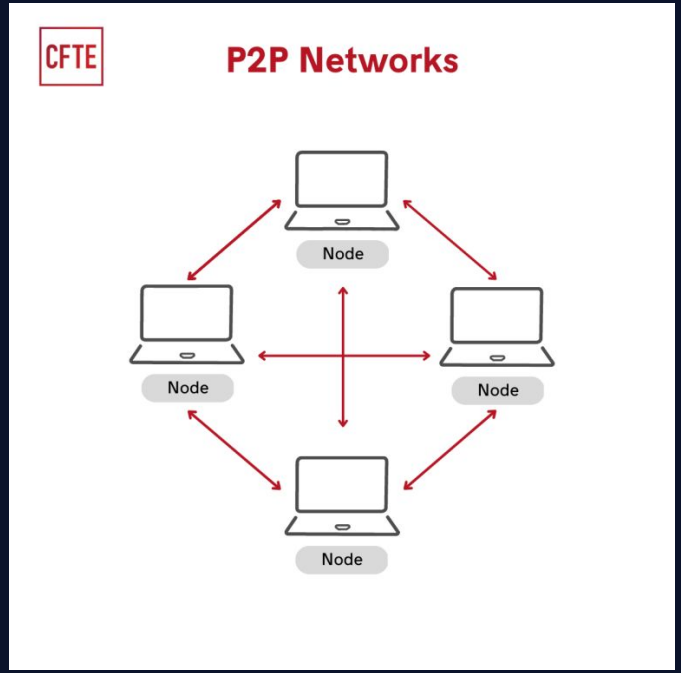
Dialogues dynamiques
Utilisation d'une API IA pour générer des conversations cohérentes et adaptatives

Multijoueur pair-à-pair (P2P)

Pourquoi P2P ?
Nos ressources limitées excluent un serveur dédié. La topologie pair-à-pair permet de synchroniser les joueurs directement

Fonctionnement
Chaque client joue aussi le rôle de serveur : il diffuse ses actions et reçoit celles des autres, mettant à jour l'état local en temps réel

Technologies
Utilisation des sockets Java et d'une API d'annuaire pour stocker les IP/ports des hôtes



Synchronisation directe entre pairs

Planification des 6 itérations

1 Prototypes

Impératif :
Création d'une base commune
Protype :
P2P
Raycasting

Optionnel :
Assemblage des prototypes
A*

2 Intégration & début algo

Impératif :
Assemblage des prototypes
A*
début BSP

Optionnel :
Action du joueur

3 Joueur & BSP

Impératif :
Action du joueur
fin BSP

Optionnel :
RTT*
Behavior tree

4 Arbre comportemental & Algo Pathfinding

Impératif :
Behavior tree
RTT*

Optionnel :
Ajout des montres

5 Bestiaire & prompt IA

Impératif :
Ajout des montres
Création du prompt

Optionnel :
ajout du pnj au jeu

6 PNJ

Impératif :
ajout du pnj au jeu

Optionnel :
Ajout de la génération procédurale

Détail de la première itération

Objectifs de l'itération 1 (30 h)

Travail impératif

• **Création d'une base commune** : branches de travail, squelette Java (packages, boucle de jeu minimale)...

• **Prototype raycasting** : affichage d'un petit labyrinthe en pseudo-3D, colonnes de murs sans textures, premiers tests

• **Prototype P2P minimal** : connexion de 2 clients, envoi/réception de messages simples (positions des joueurs) via sockets Java.

Objectifs optionnels :

• **Assemblage des prototypes** : utiliser le raycasting avec le joueur contrôlé en local, positions déjà échangées par le P2P.

• **Premiers tests d'A*** : implémentation d'un A* sur une grille simple pour préparer le pathfinding des ennemis.

Evaluation des risques

Risques

Selon nous, les tâches qui vont nous demander le plus d'effort à mettre en place sont :

BSP
Parmis BSP, deux tâches essentielles semblent ardu : Le fait d'afficher un mur (avec la gestion du clipping, etc…) et le fait de découper l'espace grâce à BSP

PNJ
La mise en place d'un prompt solide évitant les hallucinations des IA sera capital

Multijoueur
La synchronisation des différents clients à l'aide du P2P sera important

Behaviours Tree
La mise en place de l'arbre de comportement pour les actions des ennemies et les algorithmes de déplacement (A* et RRT*)

Conclusion

- Cette analyse nous a permis de définir une architecture claire autour de trois axes : moteur graphique, IA / gameplay et multijoueur P2P.
- Les principaux défis identifiés portent sur la performance (raycasting / BSP), la complexité de l'IA (behavior tree, pathfinding) et la synchronisation réseau sans serveur central.
- La planification en 6 itérations de 30 h nous donne une feuille de route réaliste pour aller du prototype aux fonctionnalités avancées (PNJ intelligents, quêtes, multijoueur).
- L'objectif est maintenant de transformer cette analyse en un prototype jouable démontrant un moteur Doom-like basé sur le raycasting et le P2P.

Sources des images d’illustrations

- https://en.wikipedia.org/wiki/Ray_casting
- https://lodev.org/cgtutor/raycasting.html
- https://www.youtube.com/watch?v=yTRzfKh4Tg0
- https://chronicles.glitch.bg/index.php/2020/11/06/binary-space-partitioning-fps-with-cad-level-editor-from-scratch/
- https://www.google.com/imgres?q=p2p%20image&imgurl=https%3A%2F%2Fblog.cfte.education%2Fwp-content%2Fuploads%2F2023%2F03%2FDiagram-of-P2P-Network-1024x1024.png&imgrefurl=
- https://www.google.com/imgres?q=behavior%20tree&imgurl=https%3A%2F%2Fblog.zhaytam.com%2Fimg%2FBehaviorTreeExample.png
-

Questions ?

Merci pour votre attention