Héphaïstos

Groupe : G.O.

Rapport de soutenance : 8 Mars 2023

Lenny Chiadmi-Delage, Ryan Dahout, Clément Chang, Yvan Fedaouche

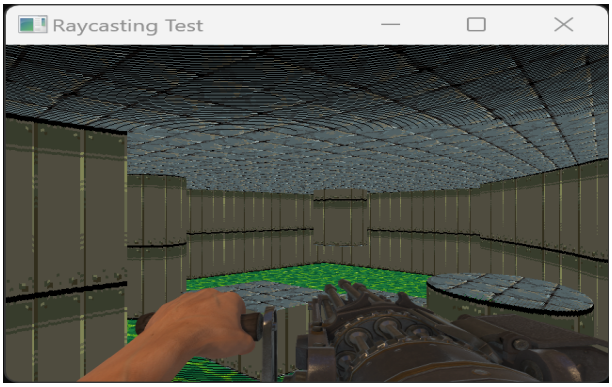


Table des matières

1 Introduction

1.1 Présentation du projet

Pour rappel le but de ce projet est simple, créer un moteur de jeu fonctionnel. Celui-ci sera constitué de 2 grandes parties, la partie logiciel et bibliothèque. Tandis que la partie logiciel commencera à être implémentée et fonctionnelle à partir de la prochaine soutenance, la partie bibliothèque est déjà partiellement implémentée et fonctionnelle.

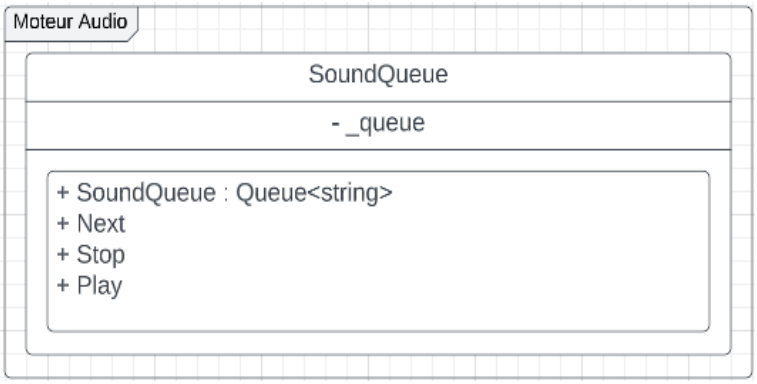
1.2 Rappel des différentes parties et Planning.

Tâches/Soutenances	**1er**	**2éme**	**3éme**
Moteur de rendu			
Algorithme de Raycasting	25%	50%	100%
Moteur Physique			
Moteur Physique	20%	60%	100%
Moteur Audio			
Moteur Audio	50%	75%	100%
IA			
IA des NPC	25%	50%	100%
Autre			
Système de cartes / niveaux	50%	100%	100%
Interfaces	25%	50%	100%
Logiciel d'édition de cartes	0%	50%	100%

Pour rappel les parties principales composants le projet à avancer pour la 1er soutenance étaient le moteur de rendu, le moteur physique, le moteur audio, l'IA, les cartes et interfaces.

2 Moteur Audio et Interfaces

2.1 Moteur Audio



2.2 Descriptif du travail réalisé

Pour la partie moteur audio, le groupe se devait, pour la première soutenance, de fournir 50% du travail. Malgré quelques impasses auxquelles nous avons dû faire face, l'équipe du secteur audio a mené à bien ce début de projet. De même pour la partie interface qui, pour cette soutenance, sera en capacité de vous monter 25% de l'interface finale.

2.2.1 Récapitulatif de l'avancement

Pour cette présentation, le moteur audio contient une liste de bande son qui représente la file de lecture et les sons en attente d'être joués ! Ainsi, pour pouvoir jouer ses meilleurs sons, l'implémentation d'un lecteur de piste était impérative, la méthode "Play" permettra donc à l'utilisateur de lancer le premier son qu'il a sélectionné et mis dans la file. Pour les autres pistes, elles attendront bien sagement leurs tours ! Si le morceau actuel lui rappelle des souvenirs à oublier (comme des exs), la méthode "Next" lui permettra de passer à la piste suivante (ouf ! Bon débarras). Attention tout de même à ce que la file ne soit pas vide, ce serait dommage de ne rien lire... De plus, si l'utilisateur doit arrêter la musique pour cause de tapage nocturne, il lui sera aisément possible d'arrêter le son grâce à la méthode "Stop". Pour finir, l'équipe offre la possibilité à l'utilisateur de convertir ses fichiers MP3 en fichier WAV.

2.2.2 Difficultés surmontées

Le moteur audio a donné lieu à un long débat quant à son implémentation. Une partie du groupe pensait qu'il fallait recréer complètement le moteur audio et ainsi implémenter soi-même les différentes méthodes sans rien utiliser, rendant la tâche bien trop compliquée pour de nouveaux programmeurs. Une fois ce différend éclairci, le travail fourni par l'équipe audio est à la hauteur de l'attente de l'ensemble du groupe.

2.3 Interfaces

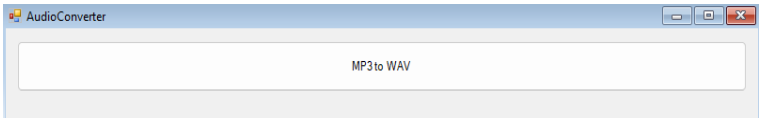
Pour cette soutenance, le secteur interface sera en capacité de vous monter 25% de l'interface finale. Malgré une longue période de concertation, le groupe a son chemin tout tracé.

2.3.1 Descriptif du travail réalisé

Pour cette soutenance, le secteur interface sera en capacité de vous monter 25% de l'interface finale. Malgré une longue période de concertation, le groupe a son chemin tout tracé.

2.3.2 Récapitulatif de l'avancement

Pour cette première soutenance, l'équipe qui s'attèle à l'implémentation des différentes interfaces est en capacité d'arborer 25% du travail. Il est donc possible de créer, d'ouvrir ou encore d'enregistrer son nouveau projet. Le groupe met aussi à disposition la possibilité de faire retour en arrière et d'annuler une action, de changer de police ou encore de zoomer sur la page. L'équipe interface a aussi la lourde tâche de faire le lien entre les différents secteurs du projet. On peut y voir les premiers liens qui commencent à voir le jour. L'interface destiné à l'audio possède déjà son propre bouton de conversion.



2.3.3 Difficultés surmontées

L'implémentation de la partie interface n'a pas été de tout repos. Le design bien que très basique regroupe les outils de base qu'a besoin un logiciel. Pour cela, plusieurs réunions afin de trouver la façon la plus propre et adéquate, ont dû être menées.

3 IA DES NPC

3.1 Ce qui était convenu de faire :

Concernant l'IA des NPC, nous devions en effectuer 25% pour la première soutenance. Ce qui a été respecté par ce dernier. Jusqu'ici, l'objectif était de créer la structure d'un code cohérent et d'implémenter quelques types simples de NPC (par exemple : au comportement passif, actif, agressif). L'objectif était aussi de réaliser des choses un peu plus complexes telles que des NPC utilisant des algorithmes.

3.2 Ce qui a été fait :

Nous avons commencé par implémenter des NPC patrouilleurs au comportement passif :

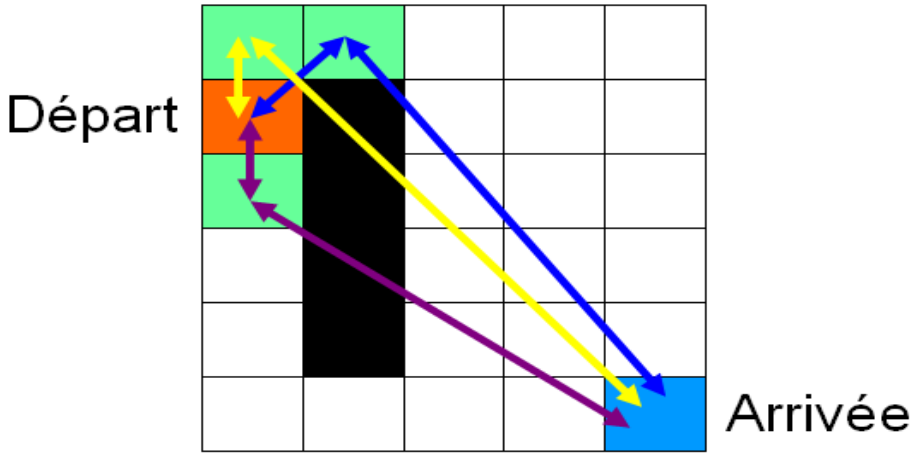
- NPC patrouillant sur l'axe X en fonction de sa position de départ
- NPC patrouillant sur l'axe Y en fonction de sa position de départ
- NPC patrouillant sur une diagonale en fonction de sa position de départ

Ces trois types de NPC vérifient que la position où il doit se rendre est accessible et sinon il retourne à la position précédente et ainsi de suite.

Nous avons réussi à implémenter un type de NPC agressif suivant l'algorithme A* afin de trouver le chemin optimal pour arriver au joueur.

Pour rappel, l'algorithme A* est un algorithme heuristique qui permet de trouver très rapidement un plus court chemin entre deux points avec d'éventuels obstacles. Outre sa vitesse, cet algorithme est réputé pour garantir une solution en sortie.

A* commence à un nœud choisi. Il applique à ce nœud un " coût " (habituellement zéro pour le nœud initial). A* estime ensuite la distance qui sépare ce nœud du but à atteindre. La somme du coût et de l'évaluation représente l'heuristique assignée au chemin menant à ce nœud.



Dans notre cas, il est très utile pour trouver rapidement et efficacement le chemin optimal pour aller à la rencontre du joueur en prenant en compte les obstacles comme des murs par exemple. Ayant terminé le travail requis pour la première soutenance, nous avons commencé à implémenter un type de NPC fuyant le joueur en s'inspirant et revisitant l'algorithme A* pour la situation demandée. Nous avons même réussi à le rendre opérationnel pour la première soutenance.

3.3 Déroulement du travail jusqu'à la première soutenance :

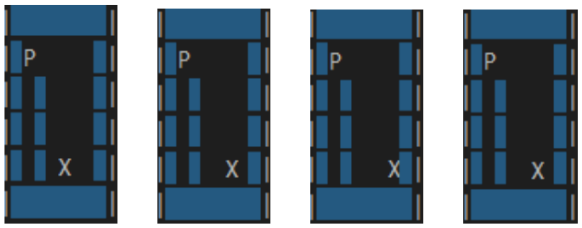
Nous avons commencé par une démarche de documentation afin de comprendre les enjeux du sujet et pouvoir commencer efficacement.
La documentation reposait principalement sur :

- L'algorithme A*
- La création de NPC
- Le Path Finding

Après cela, nous avons rencontré des difficultés lors de la création d'une structure cohérente car c'était la première fois que nous codions une solution seulement à partir de notre imagination.

Nous avons commencé par implémenter des comportements passifs pour les NPC. Le premier avancera sur un axe X selon sa position de départ tant que cela est possible, c'est-à-dire que tant que la prochaine position se trouve dans la map il avancera. Lorsque cela n'est plus possible, le NPC retournera sur sa position précédente et alternera comme cela ainsi de suite.

Voici un exemple :

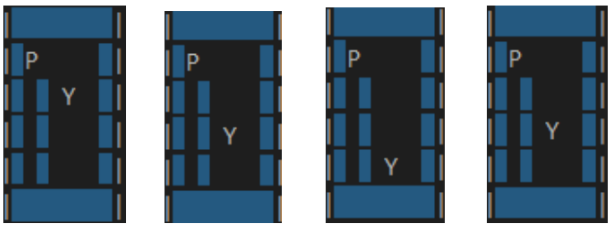


Le X représente le NPC patrouillant sur un axe X et le P le joueur.

Le NPC avance sur l'axe X jusqu'à arriver devant une case inaccessible (représenté en bleu).. Il revient donc à sa position précédente.

Le deuxième avancera sur un axe Y selon sa position de départ tant que cela est possible, c'est-à-dire que tant que la prochaine position se trouve dans la map il avancera. Lorsque cela n'est plus possible, le NPC retournera sur sa position précédente et alternera comme cela ainsi de suite.

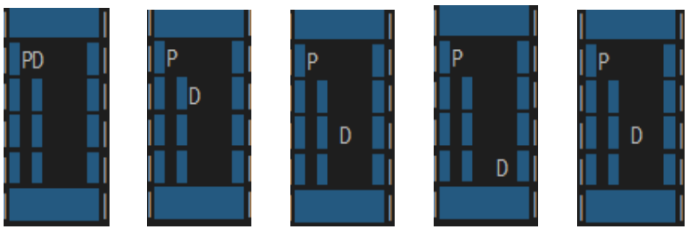
Voici un exemple :



Le Y représente le NPC patrouillant sur un axe Y et le P le joueur. Le NPC avance sur l'axe Y jusqu'à arriver devant une case inaccessible (représenté en bleu). Il revient donc à sa position précédente.

Le dernier avancera sur un axe diagonal selon sa position de départ tant que cela est possible, c'est-à-dire que tant que la prochaine position se trouve dans la map il avancera. Lorsque cela n'est plus possible, le NPC retournera sur sa position précédente et alternera comme cela ainsi de suite.

Voici un exemple :

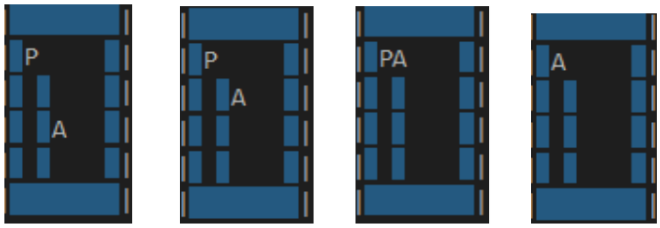


Le D représente le NPC patrouillant sur un axe D et le P le joueur. Le NPC avance sur une diagonale jusqu'à arriver devant une case inaccessible (représenté en bleu). Il revient donc à sa position précédente.

Nous avons, ensuite, commencé à coder son premier type de NPC : un monstre agressif pourchassant le joueur en suivant l'algorithme A* évoqué précédemment. Pour cela, nous avons pensé à plusieurs fonctions comme :

- Une fonction calculant la distance entre un point A et un point B en prenant leurs coordonnées (double) en paramètres
- Une fonction testant si des coordonnées entrées en paramètres sont accessibles
- Une fonction auxiliaire renvoyant les coordonnées optimales où aller pour se rapprocher du joueur
- Une fonction principale prenant en paramètre les coordonnées du joueur et renvoyant la liste des coordonnées que le NPC doit suivre afin d'aller à la rencontre du joueur

Voici un exemple :

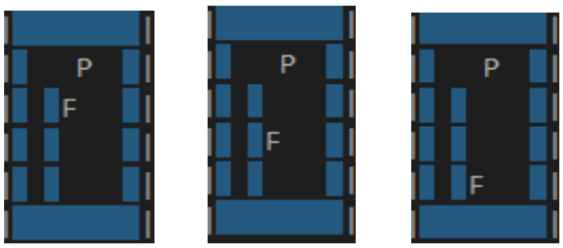


Le A représente le NPC agressif et le P le joueur. Le NPC a emprunté le chemin le plus court le menant au joueur en évitant les positions inaccessibles (représentées en bleu).

Ayant achevé le principal pour la première soutenance, nous en avons profité pour prendre de l'avance en implémentant un NPC fugitif. Le but de ce dernier est de calculer la position optimale afin de fuir le joueur, c'est-à-dire choisir la position qui est la plus éloignée de la position actuelle du joueur en prenant toujours en compte si les positions sont valides. La structure de cette classe se rapproche du NPC à caractère agressif avec des fonctions telles que :

- Une fonction calculant la distance entre un point A et un point B en prenant leurs coordonnées (double) en paramètres
- Une fonction testant si des coordonnées entrées en paramètres sont accessibles
- Une fonction renvoyant les coordonnées optimales où aller pour se rapprocher du joueur

Voici un exemple :

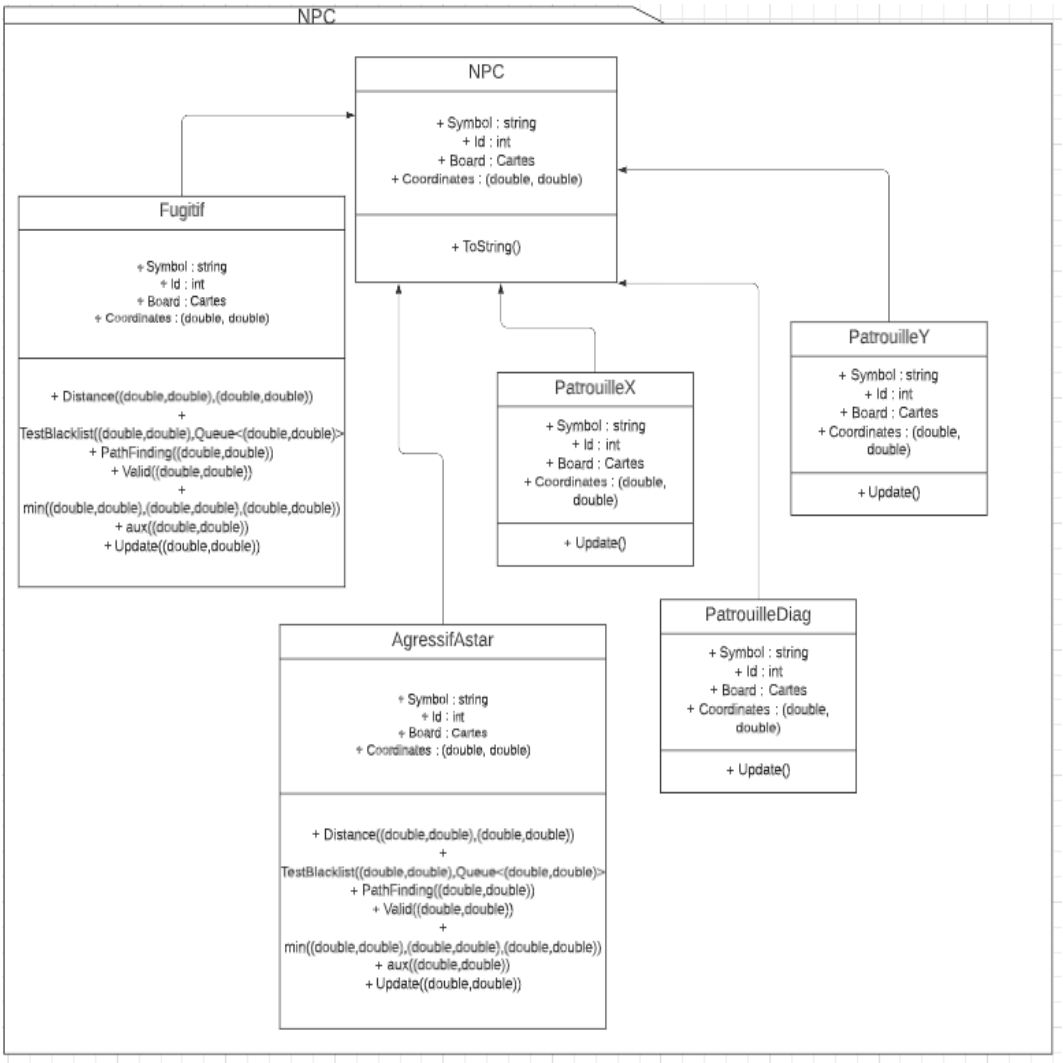


Le F représente le NPC fugitif et le P le joueur. Le NPC se rend à la position la plus éloignée du joueur sans prendre en compte les positions inaccessibles (représentées en bleu).

3.4 Ce qui est envisagé d'être fait pour la soutenance intermédiaire :

Etant donné qu'à présent la base de la solution est bien mise en place et que les comportements principaux sont implémenter, nous allons approfondir notre travail pour la soutenance intermédiaire. Ce travail reposera sur l'ajout de nombreuses actions des NPC comme des attaques avec (ou non) des armes et même des interactions avec le joueur. Nous ajouterons aussi peut-être de nouveaux comportements de NPC. Notre travail sur l'IA des NPC gagnera en qualité, en richesse et en variété.

Pour finir voici les diagrammes UML de la partie IA :



4 Moteur Physique

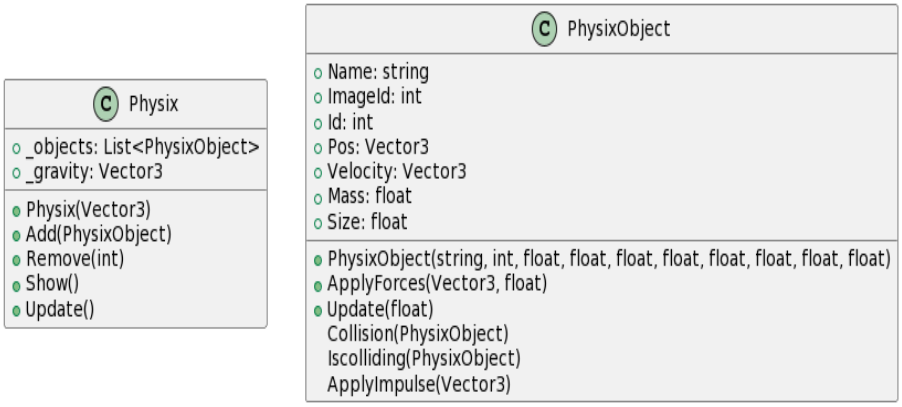
4.1 Objectif du travail :

L'objectif de ce travail est de développer un moteur physique pour simuler les interactions physiques entre les objets dans un environnement virtuel afin d'élargir le champ des possibles. Le moteur physique doit être capable de gérer la dynamique des objets en prenant en compte les forces qui s'exercent sur eux.

4.2 Bilan de la recherche :

Aujourd'hui plusieurs moteurs physiques sont disponibles sur le marché, tels que Bullet, PhysX ou même Box2D. Ces moteurs physiques sont utilisés pour simuler des jeux vidéos, des applications de réalité virtuelle, des simulations en tout genre et autres domaines nécessitant un déplacement de corps dans un monde virtuel. Chacun de ces moteurs a pour but d'être le plus fidèle à la réalité afin de proposer une meilleure immersion dans la simulation créée.

Le diagramme actuel du moteur Physique est :



4.3 Physix :

4.3.1 Introduction :

Physix est le nom de notre moteur physique et tient son nom de celui de Nvidia (PhysX).
Notre moteur physique suit la méthode d'Euler pour la simulation de la dynamique des objets.

4.3.2 La méthode D'Euler :

La méthode d'Euler est une méthode numérique pour la simulation de la dynamique des objets. Nous avons décidé d'implémenter cette méthode plutôt que celle de Verlet qui est plus précise car elle est beaucoup moins coûteuse en ressources que celle de Verlet qui prend en compte la position actuelle et précédente pour calculer sa prochaine position. De plus, grâce à la méthode d'Euler nous sommes libres d'implémenter d'autres méthodes pour améliorer la précision et la stabilité de la simulation pour atteindre une précision de calcul pour ressembler à celle de Verlet. Même si la méthode de Verlet est plus stable, plus précise, symétrique dans le temps et permet aussi de conserver l'énergie des objets, Physix est un moteur physique simplifié et c'est pourquoi nous avons opté pour la méthode d'Euler. La méthode d'Euler calcule la nouvelle position de chaque objet à partir de sa position actuelle, son accélération et sa vitesse à chaque étape.

4.3.3 Avancée du développement :

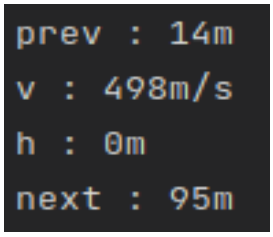
Pour l'instant, le moteur de jeu est constitué de deux classes : Physix et PhysixObjects.

- Physix est la classe qui gère tous les objets contenus sur la carte. Elle possède une méthode Update qui permet de mettre à jour tous les objets en fonction de leur vitesse, de leur position actuelle et de s' ils rentrent en collision avec d'autres objets.
- PhysixObjects est la classe qui définit un objet de la simulation. Elle possède plusieurs méthodes qui permettent de rendre plus compréhensible la méthode Update de la classe Physix.

Nous avons rencontré des problèmes lors du calcul de la prochaine position. En effet, il suffit d'inverser une multiplication et une division pour que tous les calculs soient faux. C'est pourquoi nous avons dû nous assurer que ces erreurs ne soient pas présentes dans le code actuel. Lors de nos tests, une balle gagnait en vitesse a chaque rebond.

Un test lambda lorsque l'on applique mal la méthode d'Euler :

- prev représente la hauteur précédente de la balle.
- v la vitesse actuelle de la balle.
- h la hauteur actuelle.
- next la hauteur de la balle après l'application des forces sur la balle.



4.4 Conclusion et objectifs pour la prochaine soutenance :

Nous pouvons affirmer que nos objectifs pour cette première soutenance ont été atteints malgré les quelques difficultés qui ont été surmontées. Notre moteur Physix est capable de gérer les objets présents dans une simulation grâce à la formule d'Euler, en les faisant interagir entre eux ou en leur appliquant une certaine gravité.

Pour la prochaine soutenance nous avons pour objectif de faire en sorte que le joueur soit un objet physique, les éléments de la carte aussi afin d'améliorer la précision des calculs. Toutes ces nouvelles classes héritent directement de la classe PhysixObject et permettront d'appliquer les bonnes méthodes aux bons objets physiques.

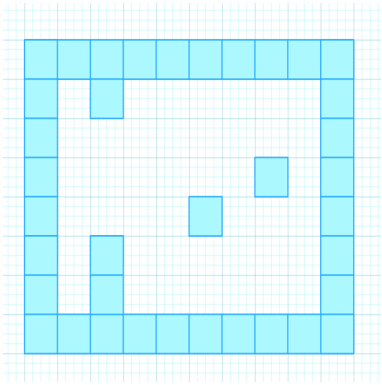
5 Moteur de rendu et Cartes.

5.1 Description du travail réalisé.

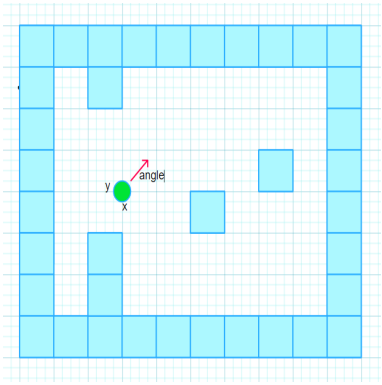
Le raycasting est une technique de rendu 3D principalement utilisée dans les jeux vidéo des années 90, qui à partir d'une simple matrice en 2D permettait de réaliser en temps réel un environnement qui a l'apparence d'une 3D rudimentaire à l'image de Wolfenstein (1992) et Doom 64 (1997).

5.1.1 La base.

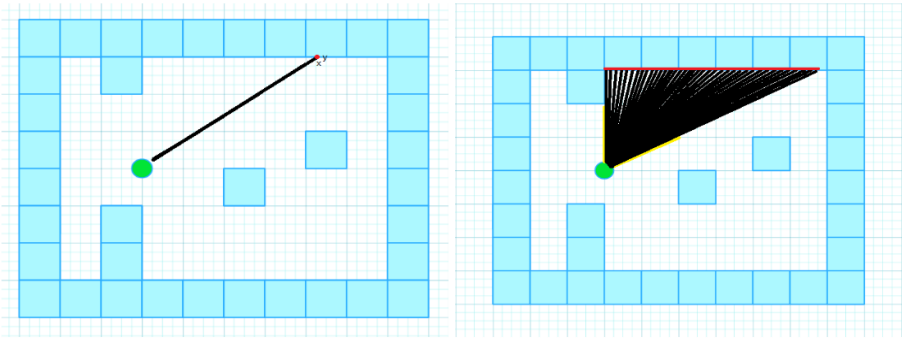
Comme précisé précédemment le monde peut être représenté à l'aide d'une matrice à deux dimensions constituée cases vides et murs. Exemple :



Quant au joueur, celui-ci est représenté par des coordonnées représentant sa position et un angle représentant la direction de son regard.



À l'aide de cet angle l'algorithme projette des rayons par colonne de pixels de l'écran permettant de calculer les coordonnées du mur à afficher pour chacune d'entre elles.



Enfin, les coordonnées du mur permettent de calculer la distance du joueur par rapport à celui-ci et en conséquence la taille du mur à dessiner sur l'écran.

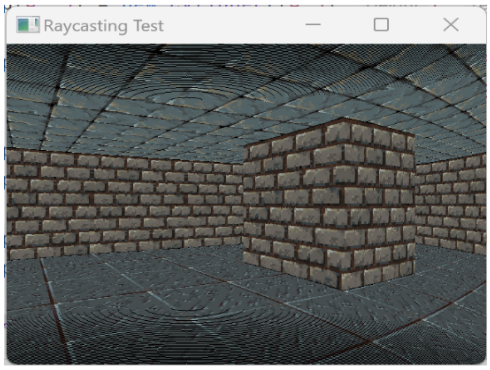


Image 1 - Ray Casting Basique.

Avec ce simple algorithme nous pouvons obtenir un premier rendu 3D, cependant celui-ci possède de nombreuses limitations et peut être grandement amélioré à l'aide de petits ajouts.

5.1.2 Les différents types de murs.

La matrice représentant la carte est constituée de vides et de murs, à l'aide de nouvelles classes nous pouvons y ajouter de nombreuses variantes dont deux principales :

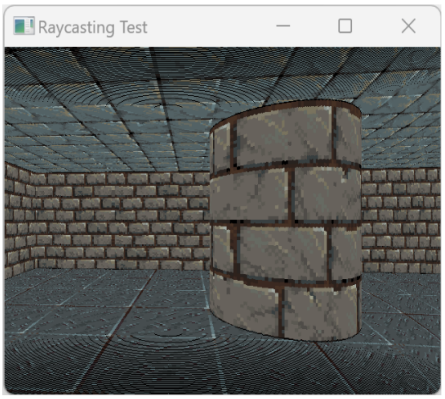


Image 2 - Le cylindre.

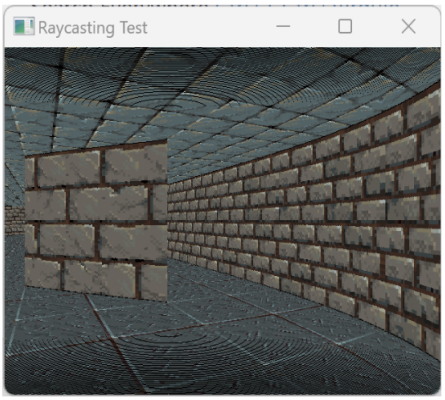


Image 3 - Le triangle.

De nombreuses autres formes peuvent être obtenues en mélangeant les 3 formes primitives que sont le rectangle, cylindre et triangle tel qu'un losange.

5.1.3 Variations de hauteurs et sur l'axe Z.

Les murs possèdent donc désormais des formes variables, mais le moteur serait beaucoup plus modulable si leur taille et hauteur l'étaient aussi. C'est pourquoi cette fonctionnalité a été implémentée afin de pouvoir réaliser des cartes beaucoup plus diverses, variées et complexes.

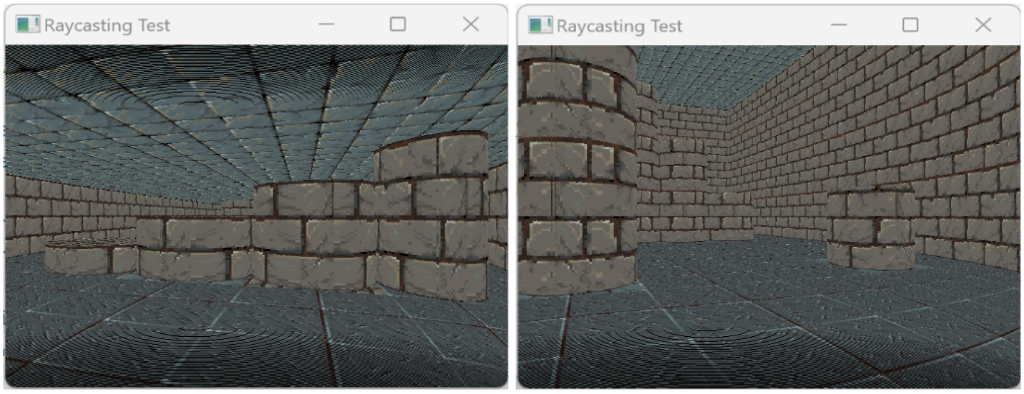


Image 4 et 5 - Variations de taille et modification de la hauteur du plafond de la carte.

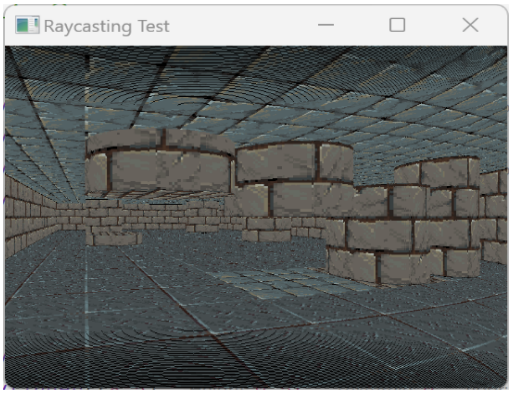


Image 6 - Variation de la taille des murs.

Il est également possible pour l'utilisateur de se mouvoir sur l'axe Z librement ce qui lui permettra par exemple d'implémenter un système de saut, vol ou autres.

5.1.4 Sol, plafond et toit.

Comme vous avez pu le constater sur les images précédentes, les cartes possèdent un sol et un plafond tandis que les murs possèdent également un toit. La texture de ceux-ci peut être changée entre chaque case permettant de créer des cartes beaucoup plus complexes en plus d'apporter des possibilités esthétiques pour celles-ci.

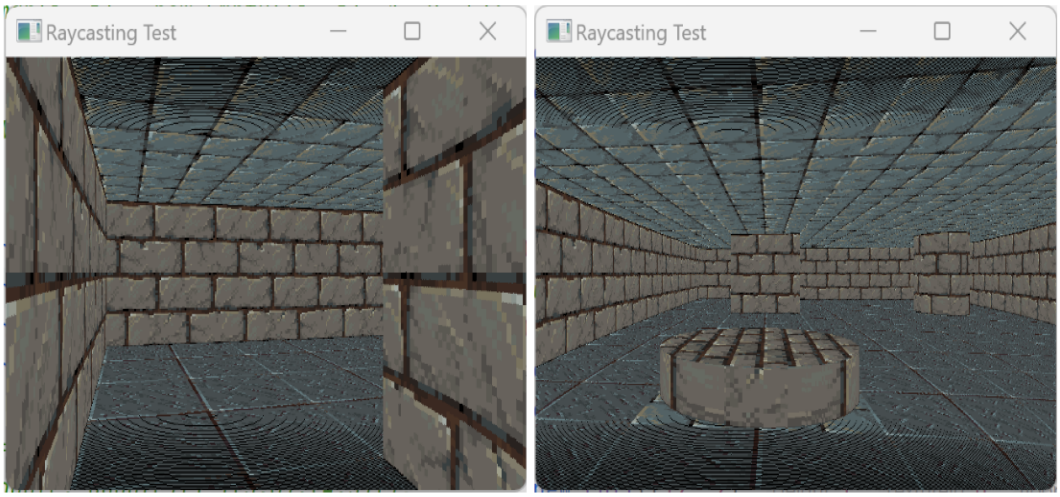


Image 7 et 8 - Sol, plafond et toit.

5.1.5 Limitations actuelles.

Le moteur de rendu possède plusieurs limitations définitives ou résolues d'içi aux prochaines soutenances. Voici une liste exhaustive de celles-ci : Impossibilité de placer plusieurs murs au sein d'une case, d'orienter la caméra verticalement et les sols, plafonds et toits deviennent transparents lorsque la caméra est trop proche de ceux-ci.

5.1.6 Les cartes.

Comme annoncé dans le cahier des charges, les cartes sont découpées en plusieurs tronçons d'une taille prédéfinie et uniforme et chargées dans une matrice à 2 dimensions pour en faciliter l'accès. Les cartes sont également automatiquement entourées de murs lors de leur création pour éviter tout problème lié à l'affichage.

5.1.7 Récapitulatif de l'avancement.

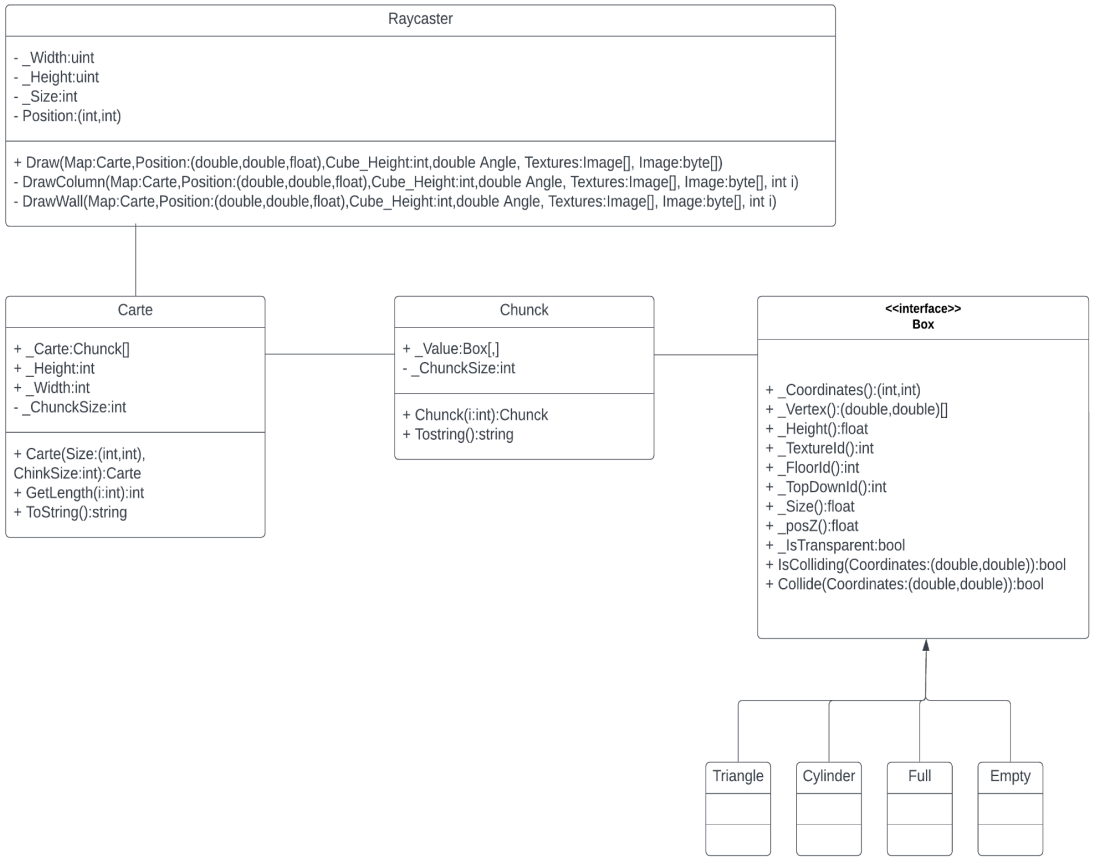
Pour récapituler l'avancement le moteur de rendu permet de retranscrire à l'écran un espace constitué de multiples formes de taille variable dans lesquelles on peut se mouvoir et orienter son regard horizontalement librement.

5.1.8 Autre.

Les deux bibliothèques utilisées par le moteur de rendu sont SFML afin de d'afficher un array de byte et détecter les entrées clavier de l'utilisateur dans la démo réalisée et TPL (Parallel Task Library) pour les solutions de multithreading que la bibliothèque propose.

SFML, www.sfml-dev.org.

Pour finir voici les diagrammes UML de la partie carte et moteur de rendu :



5.2 Conclusion

Pour conclure, le moteur de rendu est en avance sur les prévisions données dans le cahier des charges, tandis que le système de cartes s'en tire à bon compte.

5.2.1 Difficultés rencontrées.

Les principaux objectifs pour la prochaine soutenance sont l'implémentation d'un système de sauvegarde pour les cartes, d'entités pour le moteur de rendu et la possibilité de réaliser des formes plus complexes au sein de chaque case.

6 Conclusion globale.

Le groupe G.O a atteint ses objectifs. Le groupe reste malgré les quelques différents, bien soudés. La communication au sein du groupe permet à l'ensemble de l'équipe d'avancer dans la même et bonne direction. De plus, l'avance qu'ont prit certains dans leurs secteurs respectifs bonifie l'ambiance positive et la cohésion au sein du groupe de travail.

6.1 Avancement.

	Prévisions	Réalisé
Moteur de rendu	25%	50%
Moteur Physique	20%	40%
Moteur Audio	50%	50%
IA	25%	35%
Cartes	50%	50%
Interfaces	25%	25%