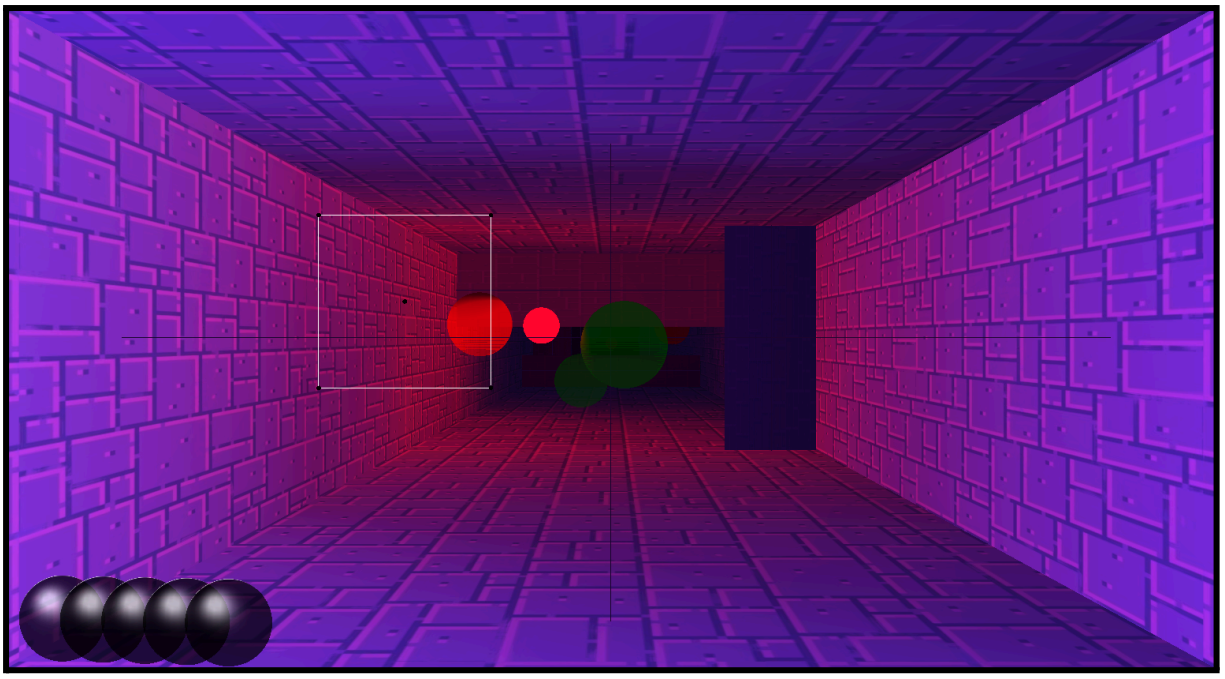
*Hakim AOUDIA*
*Sylvain TRAN*

*Semestre 2*
*Master 1 informatique*

*Université Gustave Eiffel*

# Projet de Synthèse d'images : The Light Corridor



https://gitlab.com/hak2023/the-light-corridor-aoudia-tran

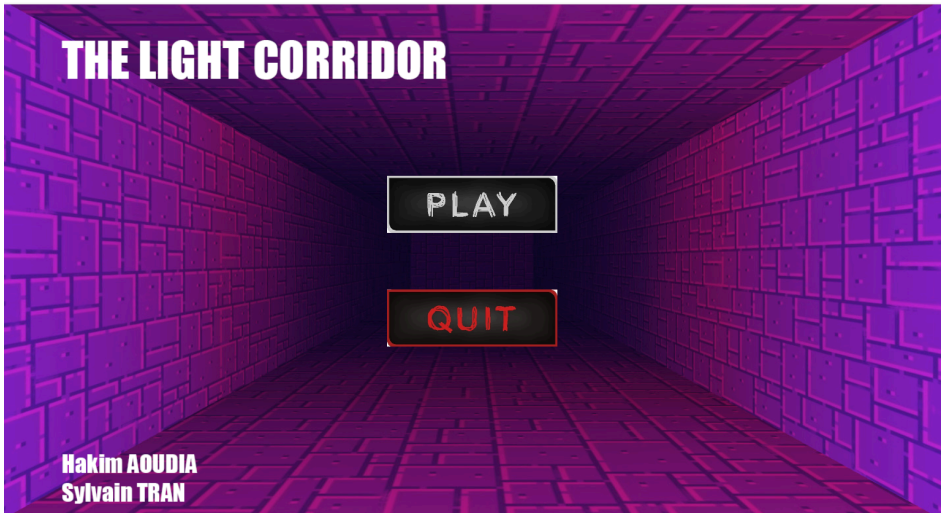
# Introduction

Lors de ce semestre nous devions programmer un jeu en C++ en utilisant OpenGL. En TP nous avons vu les bases de la synthèse d'images notamment les Translation, Scale, Rotation, Animation, Lumières etc. Donc ce projet est la combinaison de tout ce qu'on a appris durant ce semestre.

Le projet comporte de nombreux modules à implémenter nous avons d'abord lu le sujet et tenté de diviser ce le projet en plusieurs sous parties pour faciliter la programmation. Durant toute la durée de développement du projet nous avons modifié, supprimé et/ou ajouté des sous parties. Le projet s'est complexifié de plus en plus.

# Guide utilisateur

### 1 - Menu




Le joueur arrive sur cette interface. Il peut choisir entre Jouer et Quitter en utilisant le **clique gauche de la souris**.

Il peut quitter le jeu à tout moment en **appuyant sur Q.**

### 2 - Jeu

Le joueur (qui est associé à la raquette) peut faire avancer la raquette en appuyant sur le **clique gauche de la souris**.
Il peut aussi faire bouger la raquette sur l'écran en **bougeant la souris**. La position de la raquette est limitée par les murs bordant le corridor.

Au début de jeu la balle est collé à la raquette. Le joueur peut choisir ou propulser la balle en appuyant sur clique de la souris. La balle part alors tout droit et continue jusqu'au touché un obstacle.

Le joueur en bougeant la souris doit tout faire pour que la balle ne passe pas derrière la raquette. Lorsque la balle touche la raquette alors il y a un rebond qui est calculé en fonction de la position de l'impacte sur la raquette par rapport au milieu de la raquette.

Il peut ramasser des bonus positionnés dans le corridor en les touchant avec la raquette. Pour voir les types de bonus voir le chapitre suivant **'Bonus'**.

Le joueur gagne lorsque la raquette atteint la fin du corridor.

### 3 - Menu Victoire / Défaite

Lorsque le joueur va jusqu'au bout du corridor alors un menu de victoire apparaît avec le score affiché en bas à droite de l'écran.


Le score augmente en fonction des bonus ramassés (+50 points) et par la longueur du corridor.

Si il décide de continuer de jouer alors un **nouveau niveau** est généré avec de nouveaux bonus et obstacles.


Le menu défaite apparaît lorsque le joueur perd toutes ses vies qui étaient de 5 initialement.

Le joueur perd une vie lorsque la balle passe derrière la raquette.

Si il décide de cliquer sur New Game alors le même niveau est rejoué, c'est-à- dire que les obstacles ne changent pas de position de position et les bonus aussi.

# Le corridor

### 1 - Génération des obstacles

Pendant toute la phase de développement on pensait qu'on allait générer le corridor en utilisant des objets préfabriqués, à l'image des Prefabs dans Unity en lisant un fichier. Donc on avait créé un murs en Haut, Bas, Droit, Gauche qui avait comme code respectivement 1, 2, 3, 4. Donc lorsqu' on lisait le fichier 1 correspondait à un mur Haut de taille WALL_HEIGHT / 2.

Cela fonctionnait bien sauf qu'à la fin lorsqu'on a voulu ajouter d'autres types de mur on s'est rendus compte que notre code n'était pas du tout flexible et qu'il fallait changer tous les switch case de notre projet pour l'adapter. Donc pour corriger ce problème on a préféré créer une struct Obstacle qui possède des propriétés d'un mur comme sa taille, sa position et son type.

Et au lieu de lire un fichier, on génère aléatoirement les obstacles. Cela nous permet d'avoir une diversité d'obstacle plus facilement.

### 2 - Génération des bonus

Pour les bonus c'est similaire à la génération d'obstacle car on crée un nombre aléatoire de bonus (qui dépend de la taille du corridor) auquel on associe une position aléatoire dans le corridor.

# Raquette / Joueur

### 1 - Déplacement de la raquette

La raquette possède 2 types de mouvement. Un mouvement en avant et un mouvement sur l'écran (horizontal et vertical).

Le déplacement en avant est assez simple, il suffit d'incrémenter la position de la raquette sur l'axe des Y et de vérifier si la raquette touche un obstacle (partie suivante).

Pour les mouvements horizontaux et verticaux, il faut gérer les cas où la raquette dépasse des bords du corridor.
Dans notre cas on a décidé de créer la raquette en utilisant 5 points (Milieu, Haut Droite, Haut Gauche, Bas Droite, Bas Gauche). Cela nous a permis de tester facilement un point sors des bords du corridor et donc de clamper la position.

### 2 - Gestion collisions avec obstacles

Pour tester si la raquette entre en collision avec la raquette on a d'abord récupéré la liste des obstacles qui composent le niveau.
Ensuite on regarde dans quel section se trouver la raquette et avec la liste on peut savoir si il y a un obstacle dans cette section.

On connaît alors la position, taille et le type de l'obstacle dans la section ou se trouve la raquette. Il est alors simple de faire une condition qui vérifie si un des points de la raquette (selon le type d'obstacle) entre en collision ou non avec l'obstacle.

Dans le cas où la raquette entre en collision avec un obstacle alors on autorise plus le déplacement du joueur avec le clique droit tant que la raquette n'a pas changé de position.
Et on applique un léger recul de la raquette pour créer un **effet visuel**.

C'est ce principe qu'on va utiliser pour tester les collisions entre une balle et un obstacle qu'on verra dans le chapitre suivant.

## Balle

### 1 - Rebond sur les murs du corridor

Dans le sujet du projet il est écrit : "*obstacles symétriquement par rapport à la normales du mur impactée".*
En faisant nos recherches on a vu que cette "formule" s'appelle en anglais "**ray reflection**" (ou reflected ray). On a commencé par implémenter une formule qui fonctionnait relativement bien mais qui causait des rebonds étranges. On s'était trompé car on avait fait une addition au lieu d'une soustraction dans la formule.

Le site qui nous a aidé pour trouver le problème de notre formule. Il faut renvoyer la bonne normale selon quel est le mur qu'on a touché. Pour cela il faut aussi s'assurer de la bonne rotation du mur. Au début on avait des murs qui étaient à l'envers, cela ne se voit pas visuellement mais pour une lumière et une collision cela provoque des problèmes.

On a aussi oublié au début de normaliser la direction ce qui provoquait une accélération de la balle.

A part ces petites erreurs globalement le rebond de la balle sur un mur reste assez simple à implémenter.

### 2 - Rebond sur la raquette

Dans le sujet il était dit qu'il fallait créer un rebond en fonction du point d'impact de la balle sur la raquette par rapport au centre de la raquette.
Donc pour cela on va appliquer un rebond / une direction plus forte lorsque la balle a une distance plus grande par rapport au centre

On calcule donc une distance verticalement et horizontalement. Le calcule se fait en faisant le ratio entre la distance du point d'impacte et le centre de la raquette par le taille de la raquette.

### 3 - Rebond sur les obstacles

Les rebonds sur les obstacles est le même que le rebond sur les murs bordant le corridor. La différence est qu'il faut détecter les obstacles. donc pour cela on va faire comme pour la raquette. C'est-à- dire qu'on possède la liste des obstacles dans le corridor. Donc leurs positions, leurs tailles et leurs types. Et on regarde si la balle se trouve dans une section qui possède un obstacle.

Si la balle se trouve dans une section avec un obstacle on fait les calculs nous permettant de vérifier si oui il y a une collision. Selon si la collision est contre un mur de face ou arrière la normal renvoi ne sera pas la même.

## Bonus

### 1 - Types de bonus

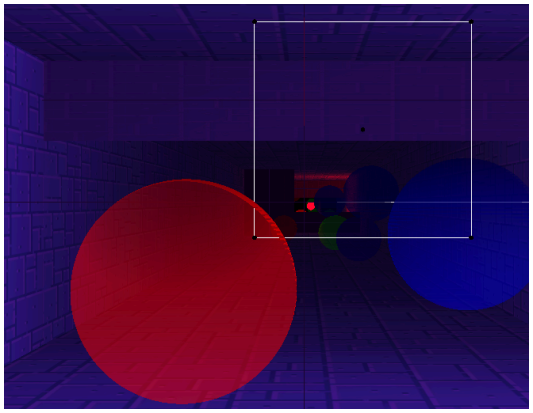
Le sujet demandait au minimum 2 bonus. on en a implémenté 3 types.

Bonus Vert : Rend la raquette collante, la prochaine fois que la balle touche la raquette, la balle reste accrochée.

Bonus Rouge : Quand la raquette touche le bonus alors le joueur gagne une vie supplémentaire.

Bonus Bleu : Augmente la taille de la raquette (taille limitée sinon le joueur serait bloqué).

### 2 - Détection de collision



Pour cette collision on aurait pu utiliser la même méthode que celle pour la collision entre la raquette et la balle, le problème est que cette méthode n'est pas très élégante et longue.

On voulait simplifier en utilisant une fonction distance. Le problème est que cette fonction n'est pas extrêmement précise pour une raison qu'on ignore. Il doit y avoir un problème dans notre condition. Le résultat est acceptable car en jeu le problème ne se voit pas. C'est pour cela qu'on a gardé la méthode des distances.

Et donc on a pas utilisé la méthode des distances pour la collision entre la raquette et la balle car pour cette collision on a besoin de précision.

# Lumière

### 1 - Lumière au niveau de la balle

Pour les lumières on a récupéré du code du TP sur les Shaders qu'on a fait en cours.
La difficulté était seulement de déplacer la lumière pour qu'elle suive la balle.

Il fallait aussi ne pas oublier de désactiver la lumière pour les éléments qui n'en ont pas besoin comme pour les vies, le score et les menus.

### 2 - Lumière au niveau de la caméra

La sujet indiqué qu'il fallait une lumière au niveau de la caméra. On a placé une lumière au niveau de la caméra derrière la raquette. Donc quand on bouge la raquette la lumière bouge aussi. On a préféré se rendre à une lumière statique.

# Transparence

### 1 - Texture 2D des vies

Dans le TP quand on a utilisé les textures on avait utilisé des couleurs RGB. Mais comme le sujet ici demande qu'il fallait un objet au moins transparent, on a voulu rendre les vies transparente. Pour cela on a donc utilisé RGBA a la place. Comme on a 4 canaux maintenant grâce à cela on peut utiliser **glColor4f** qui a comme 4ème paramètre l'opacité. Il ne faut pas oublier d'activer le "**Alpha blending**", c'est ce qui nous permet d'ajouter de la transparence a un objet.

### 2 - Objets bonus

Pareille pour les objets bonus il faut activer le **Alpha blending** avec **glEnable** et changer l'opacité de l'objet en utilisant le 4 eme parametre de couleurs.

# Difficultés

### 1 - Scores

On a cherché comment afficher du texte à l'écran en utilisant OpenGL. Et on a vu qu'OpenGL ne permettait de faire cela. Ni GLFW et GLAD. Pour afficher du texte il fallait qu'on utilise une bibliothèque externe comme GLUT ou la librairie FreeType.

Comme on ne voulait pas ajouter une librairie seulement pour le texte (en plus on ne savait pas si on avait le droit), on a décidé d'utiliser des textures pour afficher des numéros à l'écran.

On récupère un nombre entier et pour chacun de ses nombres qui le compose on affiche le bon numéro à l'écran. Une façon donc détourner pour afficher le score à l'écran.

## 2 - Collision arrière

La collision arrière est même dans le rendu final bugé. Dans certains cas la balle "tremble" ou bien entre partiellement dans un mur. On suppose que dans certains cas plusieurs de nos formules de collision s'appliquent à la balle. Mais on a pas trouver le problème exact.