

# Compte rendu : The IMAC Light Corridor



Projet de fin de deuxième semestre IMAC (Synthèse d'image)

Conception : Baptiste JOUIN (*ex-binôme Léa TOUCHARD*)

## Sommaire

---

[Sommaire](#)

[Introduction](#)

[Installation](#)

[CMakeLists.txt](#)

[Le jeu](#)

[Commandes utilisateurs](#)

[Les résultats obtenus](#)

[Le jeu en image](#)

[Fonctionnalité en bref](#)

[Architecture du projet](#)

[main.cpp](#)

[game.cpp](#)

[Méthode de travail](#)

[Utilisation de Git avec GitHub](#)

[Utilisation de Notion](#)

[Détails techniques](#)

[DrawSquare](#)

[Texture](#)

[Difficultés rencontrées](#)

[Amélioration possible](#)

## Introduction

Etant sans réponse de mon durant le projet, celui-ci à été réalisé seul. Malgré quelques difficultés je suis plutôt satisfait du résultat visuel et techniquement (organisation du code..), à noter que j'ai découvert le C++17 et OpenGL avec la formation.

The IMAC Light Corridor est développé en C++, le repository public est disponible sur GitHub à cette adresse : <https://github.com/baptistejoux/the-imac-light-corridor>

En cas de problèmes de compilation, une vidéo démo est disponible à cette adresse : <https://youtu.be/cS5yhfG6DRs>

## Installation

---

### CMakeLists.txt

J'ai conçu ce projet principalement sous MacOS, un dérivé de Linux. J'ai donc moi-même conçu le fichier CMakeLists.txt pour pouvoir compiler le projet. Il m'a tout d'abord fallu comprendre la syntaxe de CMake et l'appliquer au projet.

Il est maintenant assez simple d'installer et de lancer le code source via `cmake .`

J'ai également implémenté une commande pour compiler puis exécuter le programme via un `make run`.

Il est nécessaire d'avoir certaines bibliothèques pour ce projet :

- GLFW3
- OpenGL

## Le jeu

---

### Commandes utilisateurs

**Clique DROIT** permet d'avancer.

**Clique GAUCHE** permet de lancer la balle si elle est sticky.

**La touche K** permet de lancer la pause lorsque l'on est en jeu.

**La touche ESCAPE** permet de fermer le jeu.

**La touche Q** permet de fermer le jeu dans les menus.

**La touche R** permet de relancer le jeu dans le menu de GameOver.

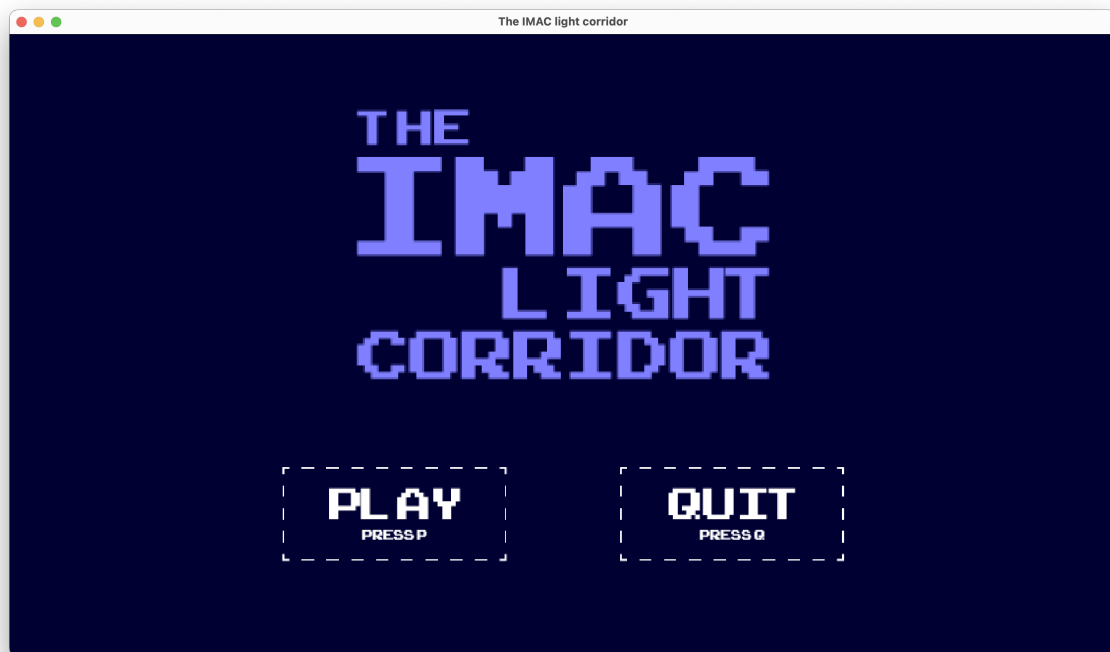
**La touche P** permet de lancer le jeu dans le menu.

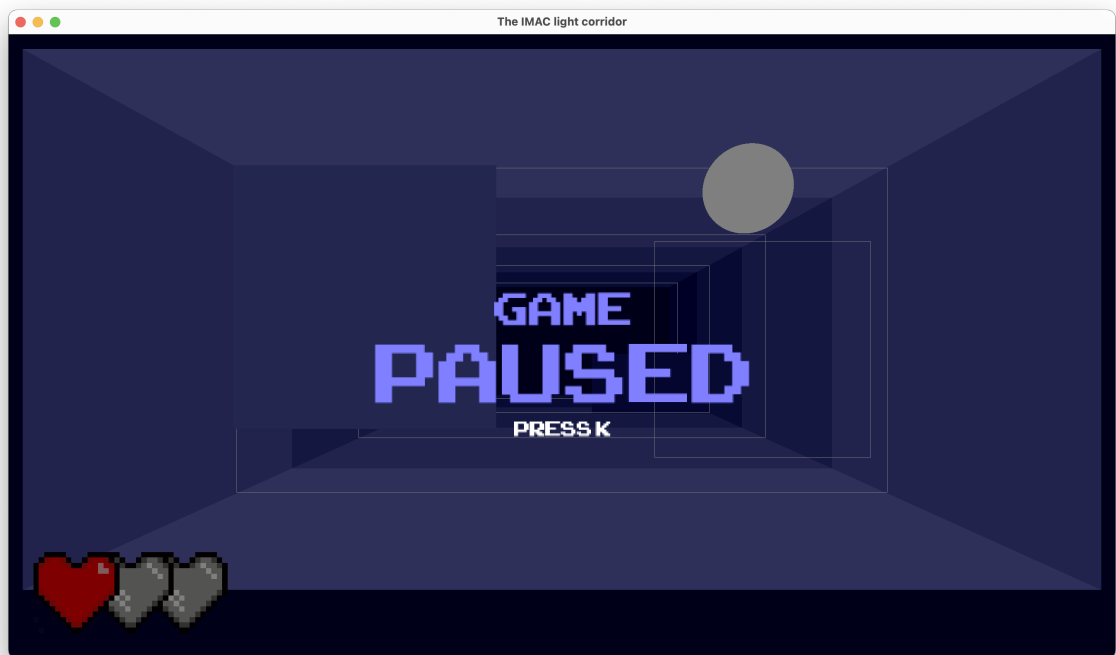
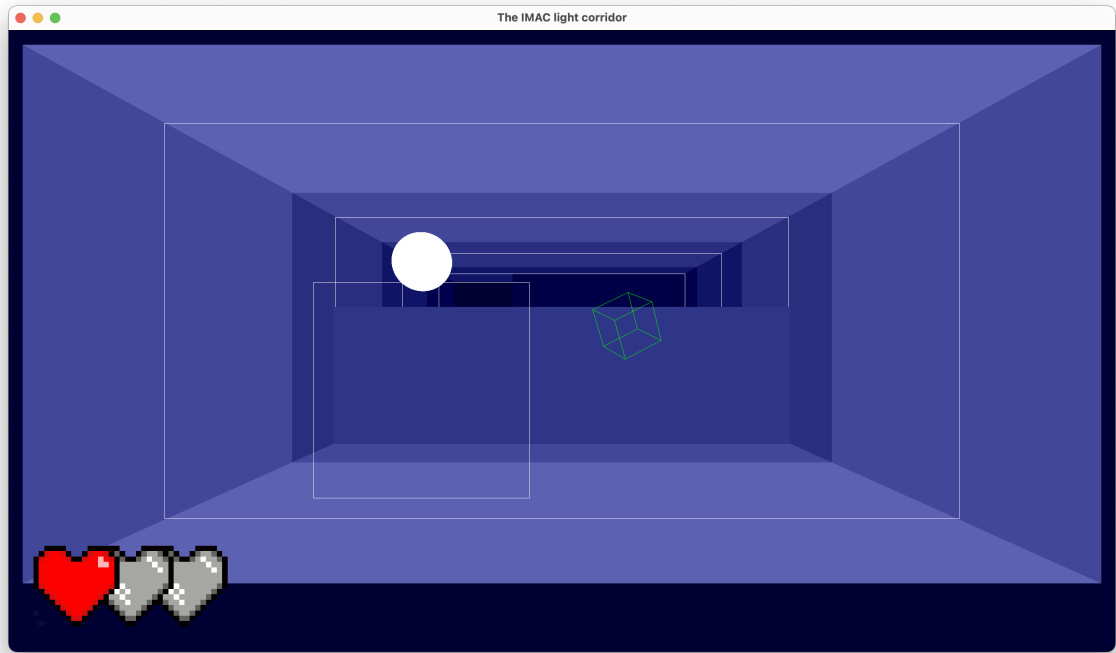
**La touche L**, quand elle est maintenue, permet de voir les arrêtes.

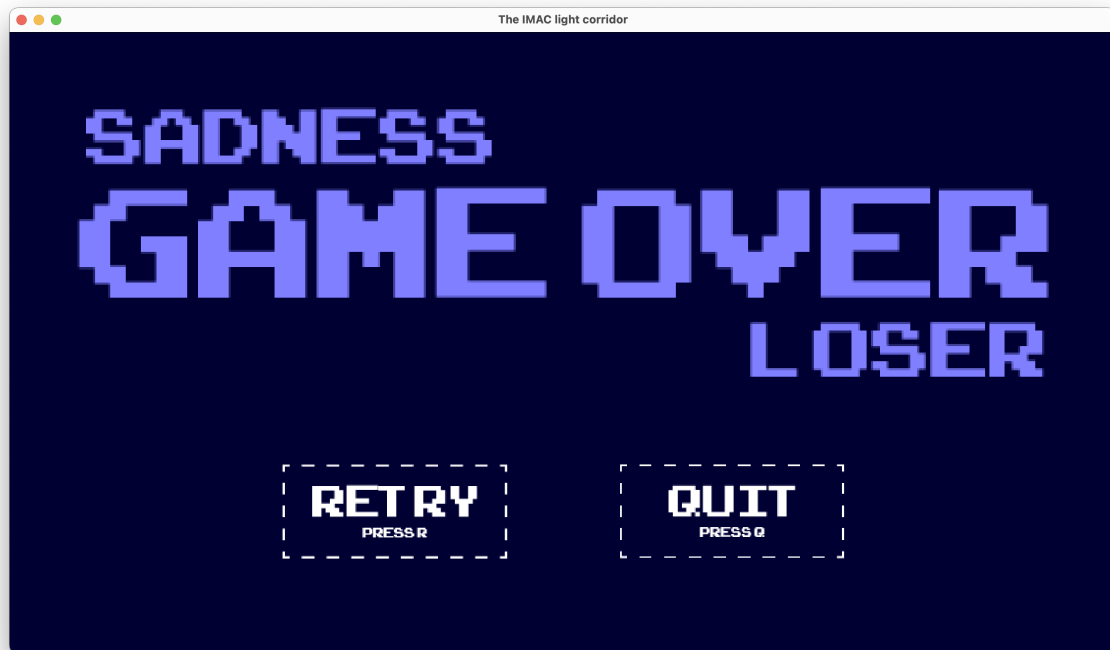
Enfin, la touche `ARROW_UP`, `ARROW_DOWN`, `ARROW_LEFT`, `ARROW_RIGHT`, `SHIFT+ARROW_LEFT`, `SHIFT+ARROW_RIGHT` permet de déboguer le jeu en se déplaçant.

## Les résultats obtenus

### Le jeu en image







## Fonctionnalité en bref

- Un menu d'entrée avec plusieurs options de jeu et un tutoriel.
- Un menu de pause avec des options pour reprendre, recommencer ou quitter le jeu.
- Un écran de fin de partie avec un résumé du score et des options pour rejouer ou quitter.
- Le jeu comprend un système de vie qui affiche la vie du joueur en jeu. Le joueur commence avec 3 vies et perd une vie chaque fois que la balle dépasse la raquette.
- En plus des vies, le jeu propose deux types de bonus. Le premier bonus, appelé "sticky", colle la balle à la raquette pour faciliter les tirs. Le deuxième bonus, appelé "vie", ajoute une vie supplémentaire au joueur (maximum 3 vies). Ces bonus apparaissent sous forme de formes 3D flottantes que le joueur doit attraper avec la raquette. Les bonus "vie" apparaissent l'axe avec un position aléatoire tout les 2000 points, et les bonus sticky tous les 5000 points. (Score disponible dans le terminal)
- J'ai également ajouté des obstacles mobiles, de taille aléatoire, la raquette ne peut pas passé au travers, il disparaissent en fade-out.

## Architecture du projet

J'ai développé le jeu pour qu'il soit facilement et à tout moment évolutif. Cela signifie que l'ajout de fonctionnalités est normalement plus facile (comme l'ajout de bonus, par exemple). Cela a été réalisé grâce au code splitting, à l'architecture des fichiers, mais aussi à l'architecture globale du projet.

### main.cpp

J'ai intégré le projet et, pour éviter tout problème, j'ai essayé de garder ce fichier aussi petit que possible. La logique est immédiatement reportée sur les autres fichiers lorsque cela est possible.

### game.cpp

C'est ici le vrai "cœur" du jeu, j'y ai écrit les fonctions primaires (appelées par main) :

```
void initGame(), void gameLoop(), void closeGame(), resetGame()
```

Les rôles de chaque fonction parlent d'eux-mêmes, c'est ensuite au sein de ces fonctions que je lance mes fonction **move** et **draw**.

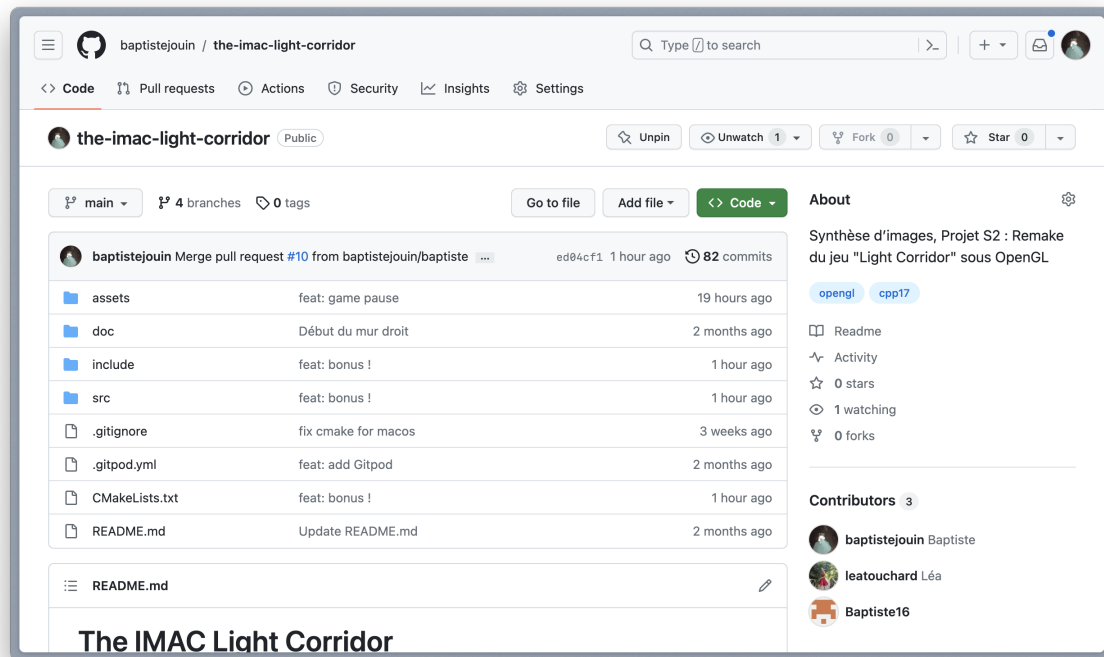
Ensuite, chaque logique a son propre fichier de code et de header (nommé de manière identique pour rester organisé). J'ai écrit mes types relatifs aux éléments dans les headers et j'ai utilisé les includes quand cela était nécessaire.

## Méthode de travail

---

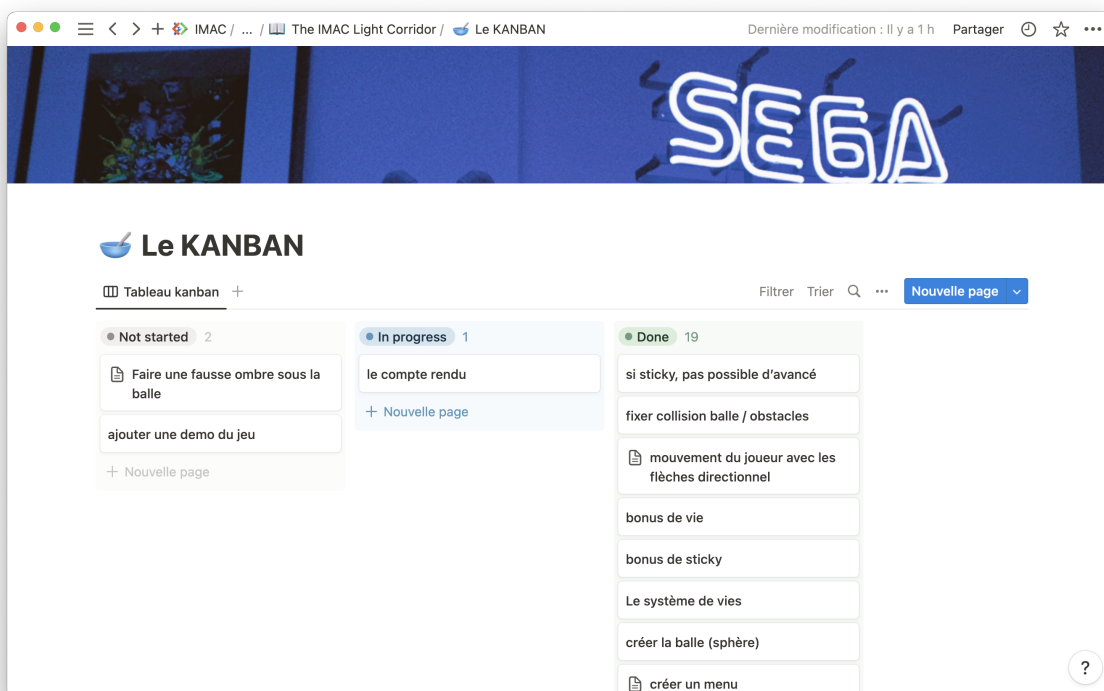
### Utilisation de Git avec GitHub

J'ai choisi de travailler avec Git/GitHub et les branches pour diviser les modifications d'un projet en plusieurs versions indépendantes. En utilisant GitHub et les branches, j'ai pu facilement gérer mon code. J'ai verrouillé la branche principale dès le début du projet, me forçant donc à utiliser des "Pull Requests" afin de fusionner (merge) mes modifications. Cela m'a permis de garder une branche principale propre et de limiter au maximum les conflits. J'ai essayé de me tenir à des conventions de nommage pour mes commits à base de "feat" pour feature et "fix" pour un bug...



## Utilisation de Notion

Dès le début du projet, j'ai créé une page Notion dédiée au projet. Il s'agit d'un tableau de bord comprenant la vidéo originale du jeu, le sujet en PDF et un Kanban pour visualiser les tâches restantes et ajouter des annotations si nécessaire.



# Détails techniques

---

## DrawSquare

J'ai implémenté une fonction `drawSquare` qui est utilisée dans de nombreuses autres fonctions. Elle permet notamment de construire le corridor, les obstacles, les vies et le menu (via des textures appliquées dessus), `drawCube`, et la raquette...

Pour pouvoir remplir toutes ces tâches, il a fallu rendre cette fonction intelligente grâce aux paramètres obligatoires/optionnels.

Par exemple, si cette fonction contient une texture (ce qui est optionnel), elle dessine un quadrilatère en gardant le bon ratio de la texture qui y est attachée.

## Texture

J'ai implémenté la fonction `loadTexture` dans le but de retourner un objet bien précis :

```
typedef struct TextureLoaded
{
    GLuint textureID;
    unsigned char *stbImage;
    int width, height, nbChannels;
} TextureLoaded;
```

Les textures sont ensuite sauvegardées sous forme de pointeurs dans la structure `Game` dans un conteneur `map`, tel que `std::map<const char*, TextureLoaded> *textures;`, pour relier chaque texture à une clé.

La fonction `loadTexture` supporte tous les formats de STBI (RGB pour jpg et RGBA pour png) et donc supporte la transparence.

Pour ce qui est du design, je me suis orienté vers le côté rétro, donc via des pixel art. Cependant, les textures sont automatiquement lissées, j'ai donc ajouté les lignes nécessaires pour contrer ce phénomène.

Le menu est composé de plusieurs texture (et non pas une seule pour pouvoir être changé facilement en cas de besoin).

J'ai également trouvé pertinent de faire toutes les textures de cœur dans une seule et même image (donc 4 cœurs disponibles dans 1 seule image) et ensuite de les diviser au sein du programme en 4 textures différentes. Un fichier pour toutes les dérivées d'une seule, toujours pour une meilleure organisation.



# Difficultés rencontrées

---

La principale difficulté que j'ai pu rencontrer a probablement été l'ajout des headers dans chacun des fichiers, il m'a fallu éviter les redondances pour éviter les boucles infinies et les erreurs de compilations. Le `#pragma once` ne suffisait pas toujours.

Comme dit précédemment j'ai développé le jeu seul, ce qui rendait l'avancement plus long comparé au binôme ; un des avantages a cela est que je n'ai pas eu besoin de scinder le travail ou résoudre des conflits de commit, le projet est comme je l'imaginais à l'origine.

## Amélioration possible

---

Il est possible d'ajouter des bonus. Je pense que l'ajout serait peu demandeur en temps avec la structure existante.

Pour améliorer la gestion du code, il aurait été préférable de diviser davantage le code en utilisant des dossiers et des sous-dossiers. Cependant, en raison des contraintes de temps, la structure actuelle a été maintenue jusqu'à la fin du projet.

Actuellement, la simulation de la lumière dans le jeu est semi-réaliste, semblable au jeu original selon mes observations. Cela présente l'avantage d'optimiser les performances du jeu, mais la lumière ne paraît pas réaliste.

Enfin, si j'avais disposé de plus de temps, j'aurais aimé ajouter l'affichage du score ainsi que des effets sonores pour enrichir l'expérience de jeu.