

## Cub3D

Mon premier RayCaster avec la minilibX

*Résumé: Ce projet est inspiré du jeu éponyme mondialement connu, considéré comme le premier FPS jamais développé. Il vous permettra d'explorer la technique du ray-casting. Votre objectif est de faire une vue dynamique au sein d'un labyrinthe, dans lequel vous devrez trouver votre chemin.*

# Table des matières

<b>I</b>	<b>Préambule</b>	<b>2</b>
<b>II</b>	<b>Objectifs</b>	<b>3</b>
<b>III</b>	<b>Règles communes</b>	<b>4</b>
<b>IV</b>	<b>Partie Obligatoire - Cub3D</b>	<b>5</b>
<b>V</b>	<b>Bonus part</b>	<b>10</b>
<b>VI</b>	<b>Exemples</b>	<b>11</b>



# Chapitre I

## Préambule

Développé par ID Software et les ultra-célèbres John Carmack et John Romero, publiée en 1992 par Apogee Software, Wolfenstein 3D est le premier "First Person Shooter" dans l'histoire du jeu-vidéo.



FIGURE I.1 – John Romero (gauche) et John Carmack (droite) posant pour la postérité

Wolfenstein 3D est l'ancêtre des jeux tels que Doom (Id Software, 1993), Doom II (Id Software, 1994), Duke Nukem 3D (3D Realm, 1996) et Quake (Id Software, 1996), qui sont des pierres angulaires additionnelles dans le monde du jeu vidéo.

Et maintenant, c'est votre tour de revivre l'Histoire...

# Chapitre II

## Objectifs

Les objectifs de ce projet sont similaires à tous les autres projets et leurs objectifs : Rigueur, utilisation du C, utilisation d'algorithmes basiques, recherche d'informations, etc.

Comme c'est un projet de design graphique, Cub3D vous permettra de travailler vos talents de designer : fenêtres, couleurs, évènements, formes, etc.

En conclusion, Cub3D est une aire de jeu remarquable pour explorer les applications pratique des mathématiques sans avoir à en comprendre les spécificités.

Avec l'aide des nombreux documents disponibles sur internet, vous utiliserez les mathématiques en temps qu'outil de création d'algorithmes élégants et efficaces.



Nous vous recommandons de tester le jeu original avant de commencer :

<http://users.atw.hu/wolf3d/>

# Chapitre III

## Règles communes

- Votre projet doit être codé à la Norme. Si vous avez des fichiers ou fonctions bonus, celles-ci seront incluses dans la vérification de la norme et vous aurez 0 au projet en cas de faute de norme.
- Vos fonctions ne doivent pas s'arrêter de manière inattendue (segmentation fault, bus error, double free, etc) mis à part dans le cas d'un comportement indéfini. Si cela arrive, votre projet sera considéré non fonctionnel et vous aurez 0 au projet.
- Toute mémoire allouée sur la heap doit être libérée lorsque c'est nécessaire. Aucun leak ne sera toléré.
- Si le projet le demande, vous devez rendre un Makefile qui compilera vos sources pour créer la sortie demandée, en utilisant les flags `-Wall`, `-Wextra` et `-Werror`. Votre Makefile ne doit pas relink.
- Si le projet demande un Makefile, votre Makefile doit au minimum contenir les règles `$(NAME)`, `all`, `clean`, `fclean` et `re`.
- Pour rendre des bonus, vous devez inclure une règle `bonus` à votre Makefile qui ajoutera les divers headers, librairies ou fonctions qui ne sont pas autorisées dans la partie principale du projet. Les bonus doivent être dans un fichier `_bonus.{c/h}`. L'évaluation de la partie obligatoire et de la partie bonus sont faites séparément.
- Si le projet autorise votre `libft`, vous devez copier ses sources et son Makefile associé dans un dossier `libft` contenu à la racine. Le Makefile de votre projet doit compiler la librairie à l'aide de son Makefile, puis compiler le projet.
- Nous vous recommandons de créer des programmes de test pour votre projet, bien que ce travail **ne sera pas rendu ni noté**. Cela vous donnera une chance de tester facilement votre travail ainsi que celui de vos pairs.
- Vous devez rendre votre travail sur le git qui vous est assigné. Seul le travail déposé sur git sera évalué. Si Deepthought doit corriger votre travail, cela sera fait à la fin des peer-evaluations. Si une erreur se produit pendant l'évaluation Deepthought, celle-ci s'arrête.

# Chapitre IV

## Partie Obligatoire - Cub3D

Nom du programme	Cub3D
Fichiers de rendu	Tous les fichiers
Makefile	all, clean, fclean, re, bonus
Arguments	Une map dans le format *.cub
Fonctions externes autorisées	<ul style="list-style-type: none"><li>• open, close, read, write, malloc, free, perror, strerror, exit</li><li>• Toutes les fonctions de la lib math (-lm man man 3 math)</li><li>• Toutes les fonctions de la MinilibX</li></ul>
Libft autorisée	Yes
Description	Vous devez créer une représentation graphique 3D "réaliste" que nous pourrions avoir au sein d'un labyrinthe en utilisant une vue subjective. Vous devez créer cette représentation en utilisant les principes du ray-casting mentionnés plus tôt.

Les contraintes sont les suivantes :

- Vous devez utiliser la minilibX. Soit dans la version disponible sur votre OS, ou depuis ses sources. Si vous décidez de travailler avec les sources, les mêmes règles que la libft s'appliquent.
- La gestion des fenêtres doit être parfaite : gestion de la minimalisation, du passage d'une autre fenetre, etc
- Vous devez afficher des textures différentes (vous avez le choix) selon si les murs sont face nord, sud, est, ouest.



- Votre programme doit être capable d'afficher un objet (sprite) au lieu d'un mur.
- Votre programme doit être capable d'avoir des couleurs différentes pour le sol et le plafond
- En prévision du jour où `deepthought` aura des yeux pour évaluer votre projet, votre programme doit pouvoir sauver la première image rendue au format `bmp` lorsque le second argument est `--save`.
- Si il n'y a pas de second argument, le programme affiche l'image dans une fenêtre, tout en respectant les règles suivantes :
  - Les touches flèches du gauche et droite du clavier doivent permettre de faire une rotation de la caméra (regarder a gauche et a droite)
  - Les touches W, A, S et D doivent permettre de déplacer la caméra (déplacement du personnage)
  - Appuyer sur la touche `ESC` doit fermer la fenêtre et quitter le programme proprement
  - Cliquer sur la croix rouge de la fenêtre doit fermer la fenêtre et quitter le programme proprement
  - Si la taille de fenêtre demandée dans la map est plus grande que celle de l'écran, la taille de fenêtre doit être celle de l'écran.
  - L'utilisation d'images de la `minilibX` est fortement recommandée.
- Votre programme doit prendre en premier argument un fichier de description de scène avec pour extension `.cub`
  - La map doit être composée d'uniquement ces 4 caractères : `0` pour les espaces vides, `1` pour les murs, `2` pour un objet, `N,S,E` ou `W` qui représentent la position de départ du joueur et son orientation.  
Cette simple map doit être valide :

```
111111
100101
102001
1100N1
111111
```

- La map doit être fermée/entourée de murs, sinon le programme doit renvoyer une erreur.
- Mis à part la description de la map, chaque type d'élément peut être séparée par une ou plusieurs lignes vides.
- La description de la carte sera toujours en dernier dans le fichier, le reste des éléments peut être dans n'importe quel ordre.
- Les informations de chaque élément peuvent être séparées par un ou plusieurs espace(s).

- Pour chaque élément, le premier caractère est l'identifiant (un ou deux caractères), suivi de toutes les informations spécifiques à l'élément dans un ordre strict tel que :

- Resolution :

```
R 1920 1080
```

- identifiant : **R**
- taille de rendu axe x
- taille de rendu axe y

- texture nord :

```
NO ./path_to_the_north_texture
```

- identifiant : **NO**
- chemin vers la texture nord

- South texture :

```
SO ./path_to_the_south_texture
```

- identifiant : **SO**
- chemin vers la texture sud

- West texture :

```
WE ./path_to_the_west_texture
```

- identifiant : **WE**
- chemin vers la texture ouest

- East texture :

```
EA ./path_to_the_east_texture
```

- identifiant : **EA**
- chemin vers la texture est

- Sprite texture :

```
S ./path_to_the_sprite_texture
```

- identifiant : **S**
- chemin vers la texture sprite

- Couleur du sol :

```
F 220,100,0
```

- identifiant : **F**
- couleurs R,G,B range [0,255] : **0, 255, 255**



# Chapitre V

## Bonus part



Les bonus ne seront évalués que si votre partie obligatoire est parfaite. Par parfaite, nous voulons naturellement dire que tout est complet, que ça n'échoue dans aucun cas de figure, même en cas d'erreur terrible comme un mauvais usage ou autre. Si vous n'avez pas tous les points de la partie obligatoire, votre partie bonus sera totalement ignorée.

Liste des bonus :

- Collision contre les murs
- Une skybox.
- Texture sur le sol ou le plafond.
- Un HUD.
- Possibilité de regarder en haut et en bas.
- Sauter ou s'accroupir.
- Un effet d'ombre basé sur la distance.
- Une barre de vie.
- Plus d'objets dans le labyrinthe.
- Collision avec les objets.
- Gain ou perte de points/de vie en prenant des objets/pièges.
- Porte qui peuvent être ouvertes/fermées.
- Portes secrètes.
- Animations (tir d'arme ou des sprites animés).
- Plusieurs niveaux.
- Son et musique.
- Des armes et des méchants à combattre!



Dans le cadre des bonus, vous êtes autorisés à utiliser d'autres fonction tant que leur utilisation est justifiée pendant l'évaluation, et vous pouvez modifier le fichier scène pour qu'il réponde a vos besoins. Soyez malins !

# Chapitre VI

## Exemples



FIGURE VI.1 – Wolfenstein3D Jeu original utilisant le ray-casting.

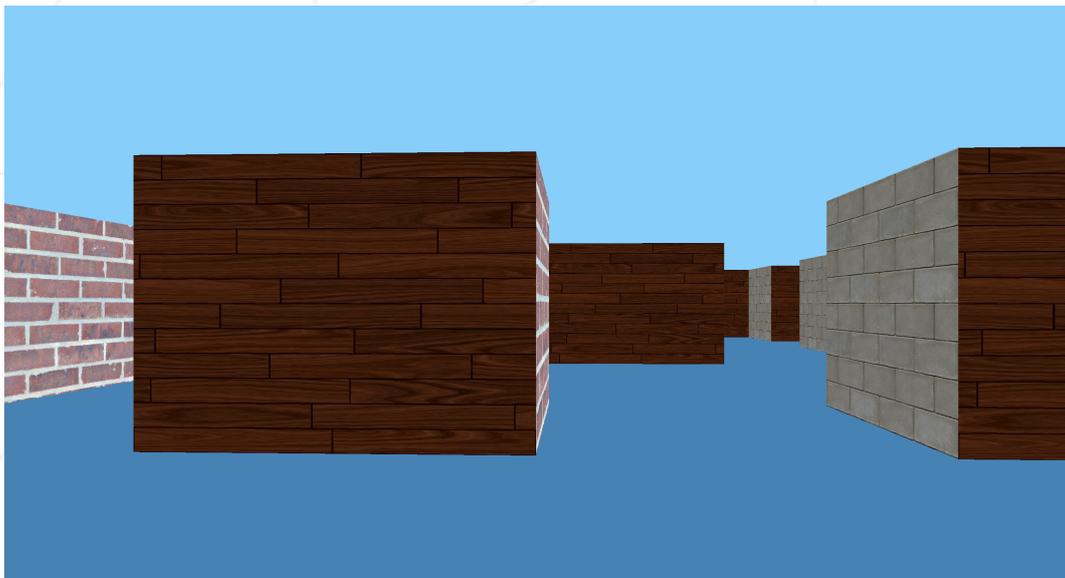


FIGURE VI.2 – Exemple de ce que la partie obligatoire attend de vous.



FIGURE VI.3 – Exemple de bonus avec une minimap, des textures sol et plafond, et un célèbre hérisson animé.

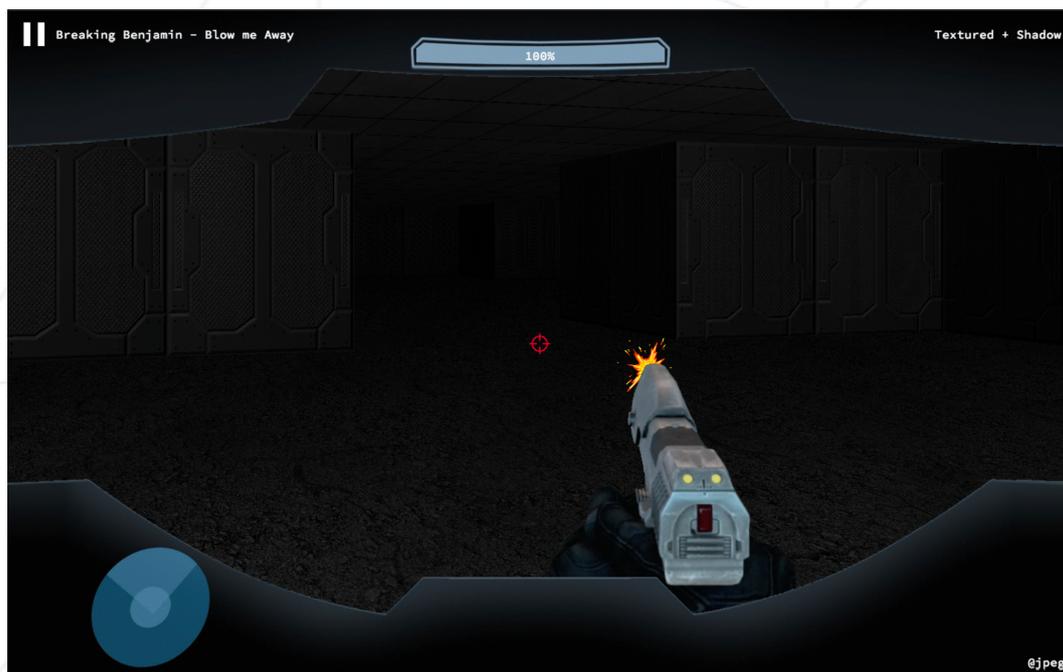


FIGURE VI.4 – Un autre exemple de bonus avec un HUD, une barre de vie, un effet d'ombre et une arme qui peut tirer



FIGURE VI.5 – Autre exemple de partie bonus avec une arme de votre choix et le joueur qui regarde le plafond