**Rapport PRE-TPI**

**Rapport réalisé par : Sacha Leone**

Classe : 3IND-3TPMa

Commencé le : 17.02.2025

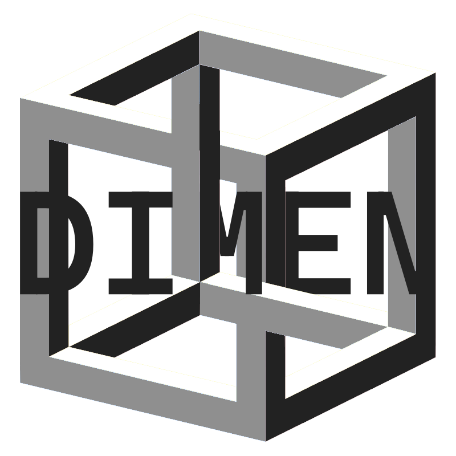
Projet : DIMEN

Table des matières

[1. Introduction 1](#_Toc193788233)

[Description du projet 1](#_Toc193788234)

[Hypothèses 1](#_Toc193788235)

[2. Matériel et méthodes 1](#_Toc193788236)

[Matériel/logiciels utilisé 1](#_Toc193788237)

[Sources utiles 1](#_Toc193788238)

[Méthodes 1](#_Toc193788239)

[3. Réalisation et observations 2](#_Toc193788240)

[4. Résultats 2](#_Toc193788241)

[5. Conclusion 2](#_Toc193788242)

[6. Annexes 3](#_Toc193788243)

# Introduction

Dans le cadre du PRE-TPI, il a fallu réaliser un projet informatique sous 80 périodes. Ce projet était supervisé par M. Schenk qui, lorsque cela s'est avéré nécessaire, a été consulté pour recevoir diverses indications. Le but du PRE-TPI est de préparer l’élève pour le TPI final. Le PRE-TPI a les même objectifs qu’un vrai TPI, il faut donc rendre le rapport, le cahier des charges et le travail réalisé. Une fois tout ces éléments rendus il faut, normalement, présenter le projet devant deux experts et le superviseur. Dans notre cas le seul qui posera des questions et examinera le projet sera M. Schenk.

## Description du projet

Le but est de développer un jeu, avec la librairie de Raylib, qui permet à un joueur de résoudre des puzzles dans un environnement tridimensionnel et bidimensionnel. Pour passer d’une dimension à une autre et résoudre les puzzles avec des éléments cachés, on change le point de vue de la caméra. Les éléments présents dans l’environnement pour que le joueur résout le puzzle sont par exemple :  
des plaques de pressions, des cubes déplaçables, des clés, des boutons, etc…

## Hypothèses

Dans le cahier des charges il est mentionné que, si l’avancement du projet était optimal il aurait été possible de pousser le projet plus loin en ajoutant des fonctionnalités, textures, niveaux, etc...

L’idée de départ est de réaliser un **MVP** (Produit minimum viable) **ou prototype** afin de présenter ce à quoi pourrait ressembler lejeu.

# Matériel et méthodes

## Matériel/logiciels utilisé

* Ordinateur sous Windows 10 (Clavier, souris, écran, etc…)
* [Visual Studio 2022](https://visualstudio.microsoft.com/fr/vs/), avec la librairie [Raylib](https://www.raylib.com) (Écriture du code)
* [GitHub](https://github.com/) (Déposer le code et nommer les étapes avec les commit)
* [Krita](https://krita.org/fr/) (Logiciel de dessin, montage, retouche)

## Sources utiles

* [Freesound](https://freesound.org/) (Effets sonores et musiques gratuits)
* [Sketchfab](https://sketchfab.com/feed) (Modèles 3D gratuits)
* [Raylib (cheatsheet)](https://www.raylib.com/cheatsheet/cheatsheet.html) (Utilisation de la librairie)
* [ChatGPT](https://chatgpt.com/) (Debug et corrections)

## Méthodes

Premièrement, la rédaction du cahier des charges est la priorité avant d’entamer le projet.

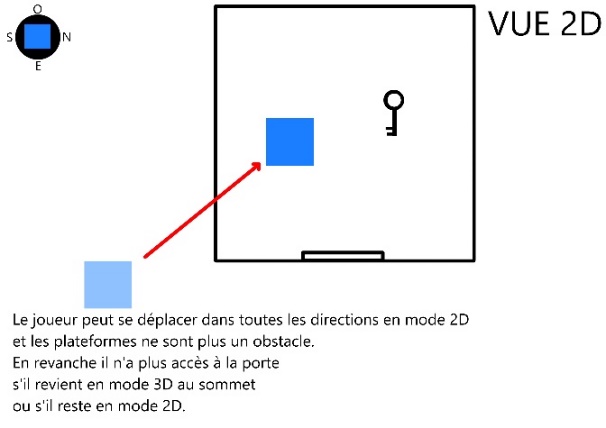
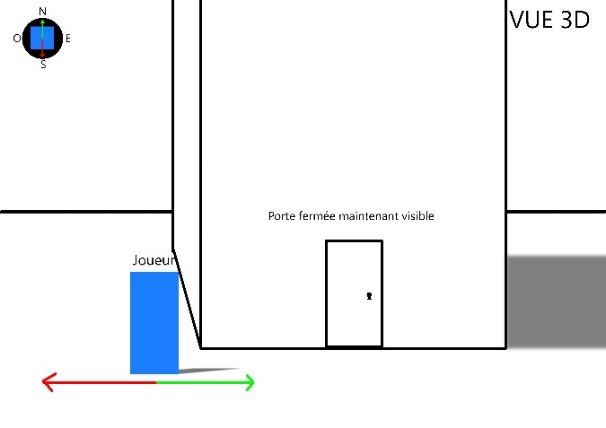
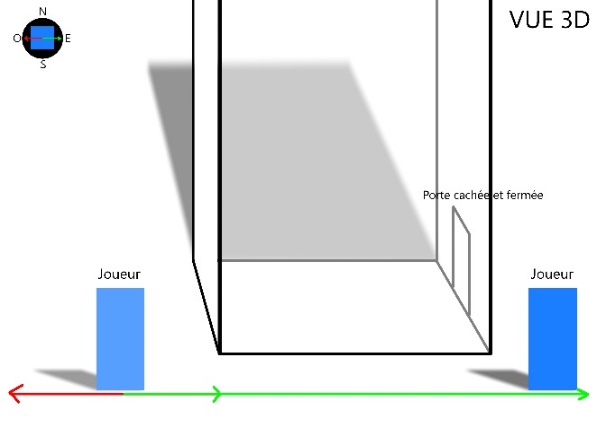
Chaque jour de travail il est demandé de remplir un journal (JT) afin d’avoir un suivi sur l’avancement du projet et sur les problèmes rencontrés.  
*(La rédaction du JT, est soutenue par le code présent dans les commits sur GitHub qui, ont été préalablement nommés et listés chronologiquement)*

Un planning est quotidiennement mis à jour.

Dans les dernières semaines du PRE-TPI, la rédaction du rapport et d’une présentation est engagée pour finaliser le projet.

Afin d’avoir une idée claire du fonctionnement primaire de l’application, un schéma explicatif a été réalisé à l’aide de Krita. Le schéma qui va suivre, apparaissait également dans le cahier des charges :

Figure 1 : Croquis sur le fonctionnement de DIMEN



# Contexte

## Dimensions

Raylib propose la création d’environnements 3D et 2D. Après réflexion, la difficulté de passer de la 2D à la 3D se serait avérée plus accrue.

Par exemple, si l’on voulait passer de la 2D à la 3D il aurait fallu enregistrer toutes les informations d’un objet 2D (coordonnées, état, orientation, échelle, etc…). Puis, retranscrire ses données dans des objets 3D et les recréer. Il aurait donc fallu faire vivre « deux parties » simultanément. Il a été préférable d’éviter cette difficulté.

J’ai choisi d’aborder une approche en 3D qui devient de la 2D, plutôt que l’inverse.

Lorsque le joueur clique sur la touche « Q », les obstacles sont « aplatis » et le joueur peut les franchir comme s’ils n’existaient pas. Lorsqu’il est posé sur un obstacle en 2D et qu’il repasse en mode 3D, après que le temps d’utilisation se termine ou qu’il reclique « Q », le joueur se trouve au sommet de l’obstacle.

## Déplacements

Le joueur est dans un **environnement tridimensionnel** et il se déplace sur les **axes X, Z,** positivement ou négativement. Il se déplace à l’aide des touches **W, A, S et D** Le joueur est **incapable de sauter**, d’où l’utilité de changer de perspective.

## Caméra dimensionnelle

**Définition « orthographique » :**

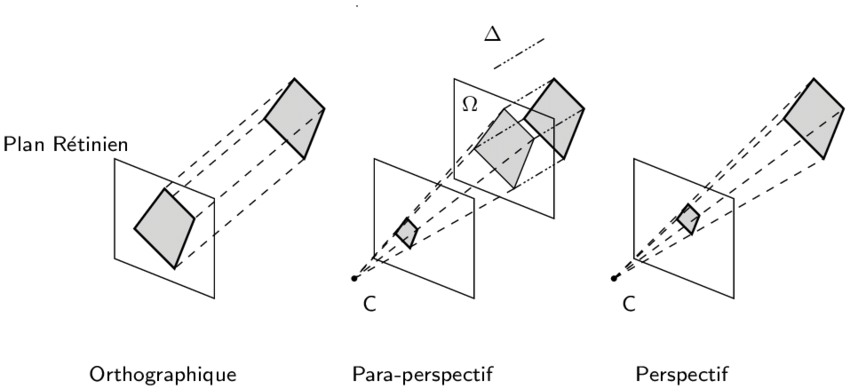
*En analyse 3D, la vue* ***orthographique*** *est la perspective qui permet d'afficher des données dans une scène en tant que plan bidimensionnel vu du dessus. (*[*ESRI : Définition de Vue Orthographique | Dictionnaire SIG*](https://support.esri.com/fr-fr/gis-dictionary/orthographic-view#:~:text=%5BAnalyse%203D%5D%20En%20analyse%203D,la%20totalit%C3%A9%20de%20l'affichage.)*)*

Figure 2 : [ResearchGate](https://www.researchgate.net/figure/Trois-modeles-de-cameras-orthographique-para-perspectif-et-perspectif-Le-meme-objet_fig2_30514862), Trois modèles de caméra.

Lorsque le joueur clique sur la touche « **E** », le point de vue de la caméra, toujours centrée sur le joueur, **se déplace de 90 degrés** dans un cercle. Voici deux schémas qui illustrent les déplacements de la caméra en mode 3D :

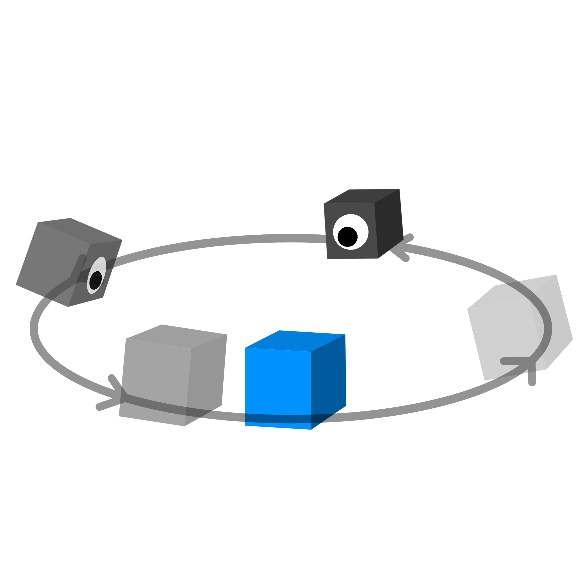
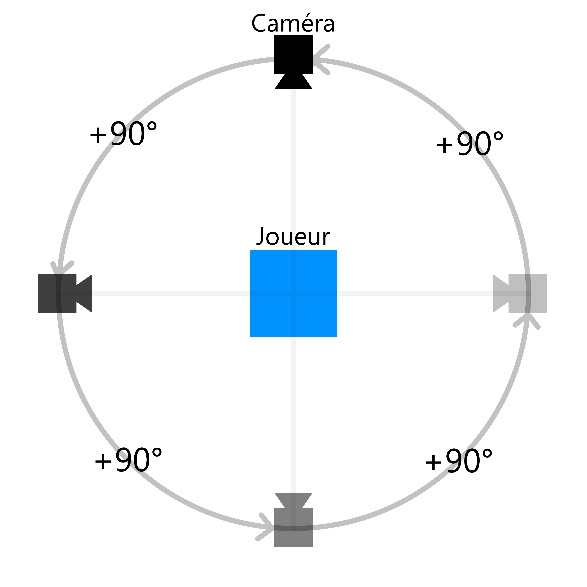


Figure 3 : Angles de caméra à plat.

Figure 4 : Angles de caméra

Lorsque le joueur veut passer en **vue orthographique**, il clique sur « **Q** » et la caméra se place **au-dessus du joueur** et aplati les éléments visibles. Si on considère toutes les positions possibles de la caméra, elles s’apparentent donc à une demi-sphère. Voici un schéma qui illustre toutes les positions de caméra possibles :

Bien que Raylib fournisse déjà une classe « **Camera** », une nouvelle classe « **DimensionCamera** » a été nécessaire. Une analyse plus complète de cette classe suivra dans la suite du document.

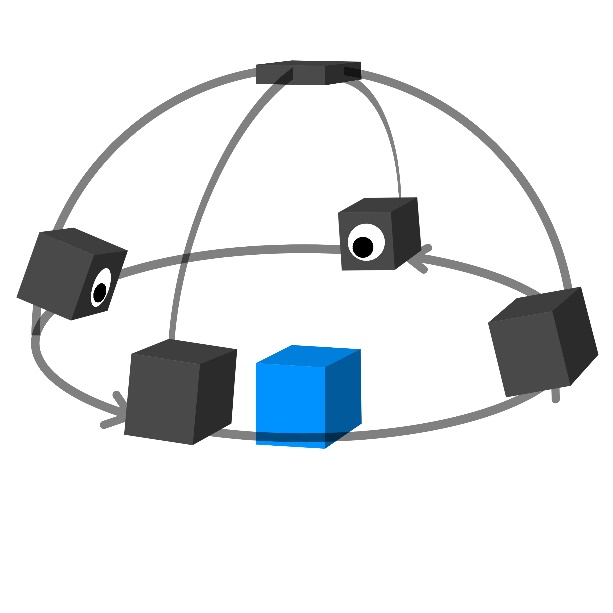


Figure 3 : Angles de la caméra et vue du dessus, en 3D

## Joueur

Le joueur est représenté par un cube, qui a la possibilité de se déplacer dans un niveau. Il subit les règles physiques basiques comme : la gravité, la friction, la vitesse, l’accélération, etc…

Ce joueur a pour but de résoudre un puzzle imposé. Il peut changer de dimensions, surmonter les obstacles trop grands, voir le niveau sous d’autres angles ou encore atteindre des objets difficiles d’accès. Tout ceci dans l’intérêt de finir le niveau.

## Classes

Pour faciliter l’écriture et améliorer la lisibilité du code, plusieurs classes, incluant les propriétés de Raylib, ont été préparées. Voici la liste de toutes les classes créées :

* Player
* Cooldown
* Obstacle
* DimensionCamera
* MovableCube
* PressurePlate

Nous allons parcourir chacune de ces classes afin de comprendre comment elles interagissent entre elles et pourquoi elles sont pertinentes.

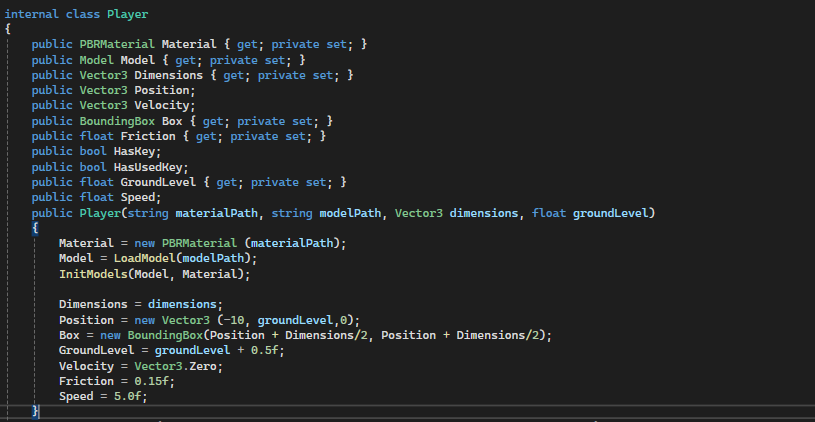
# Réalisation

## Création des classes principales

### Player

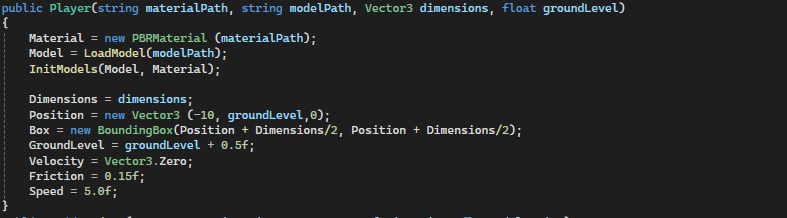
Le premier élément central a été la classe Player. Elle permet à l’utilisateur de voir l’environnement qui l’entoure, d’interagir avec et de l’explorer.

#### Propriétés



Cette classe permet au joueur de se déplacer avec les touches directionnelles et d’interagir avec son environnement. J’ai aussi ajouté une logique simple de chute avec gestion de la gravité.

#### Constructeur



#### Méthodes

##### 1. Update

Cette méthode met à jour les déplacements et la position du joueur dans l'environnement, tout en appliquant les effets de la gravité.

* Gravité : Si la position verticale du joueur (Position.Y) est au-dessus du niveau du sol (GroundLevel), la méthode applique une force de gravité qui modifie la composante Y de la vélocité du joueur.
* Déplacement : En fonction des touches enfoncées (W, S, A, D), la méthode calcule le mouvement du joueur dans les directions correspondantes (moveDirection et strafeDirection). Le mouvement est normalisé pour éviter des vitesses disproportionnées.
* Freinage progressif : Si le joueur ne se déplace pas, la méthode appelle la fonction Brake() pour appliquer une décélération progressive à la vitesse du joueur.
* Mise à jour de la position : Enfin, la position du joueur est mise à jour en fonction de sa vélocité et du deltaTime.
* Mise à jour de la hitbox : La boîte de collision (hitbox) du joueur est mise à jour en fonction de sa position actuelle.

##### 2. Brake

Cette méthode applique un freinage progressif à la vitesse du joueur sur les axes X et Z.

* Décélération : La vélocité du joueur est réduite par un facteur de décélération (0.9f). Cela permet d'implémenter un freinage progressif sur les mouvements horizontaux du joueur.
* Évitement de valeurs proches de zéro : Si la vitesse est trop faible (proche de zéro), elle est directement réglée à zéro pour éviter des valeurs indésirables qui pourraient faire osciller la position du joueur.

##### 3. HandleCollision

Cette méthode gère les collisions du joueur avec les obstacles dans l'environnement, en ajustant la position du joueur pour résoudre les chevauchements.

* Collision classique : Si le joueur entre en collision avec un obstacle (détectée par CheckCollisionBoxes), la méthode calcule l'overlap (chevauchement) entre les boîtes de collision du joueur et de l'obstacle pour déterminer dans quelle direction ajuster la position du joueur.
* Axes prioritaires : En fonction de la direction du chevauchement, la méthode ajuste la position du joueur sur les axes X, Z ou Y en déplaçant le joueur juste hors de l'obstacle.
* Vue de dessus : Si la vue est de type "top view", le joueur se positionne à la hauteur de l'obstacle touché.

##### 4. HandleHallway

Cette méthode gère les collisions du joueur avec des obstacles dans un couloir ou un espace restreint.

* Collision avec des obstacles : La méthode détecte les collisions avec d'autres obstacles dans la scène. Si une collision est détectée, elle ajuste la position du joueur en fonction de la direction du chevauchement, de manière similaire à HandleCollision, mais avec un traitement spécifique pour les espaces restreints (comme les couloirs).

##### 5. HandlePlate

Cette méthode gère les interactions du joueur avec des plaques de pression.

* Compression de la plaque : Lorsque le joueur entre en collision avec une plaque de pression, la méthode ajuste la position de la plaque en fonction de l'état de la plaque (pressée ou non) et applique une compression sur l'axe Y.
* Réajustement de la position du joueur : Le joueur est déplacé à la hauteur correcte par rapport à la plaque de pression, en fonction de l'état de la plaque (pressée ou non), et la vélocité verticale (Velocity.Y) est réinitialisée.

##### 6. InitModels

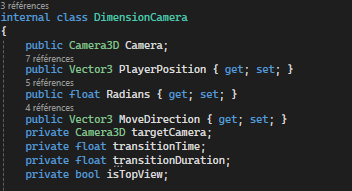
Cette méthode initialise les modèles du joueur en générant des tangentes et en appliquant le matériau approprié à chaque maillage du modèle.

* Génération des tangentes : Pour chaque maillage du modèle, les tangentes sont générées pour garantir que les effets de lumière, comme les ombres, soient correctement calculés.
* Application du matériau : Le matériau fourni est appliqué au modèle du joueur.

### DimensionCamera

Ensuite, j’ai créé DimensionCamera. Cette classe permet de passer d’une vue 3D classique à une vue orthographique en 2D.

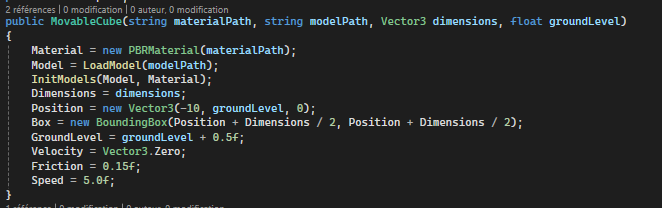
#### Propriétés



J’ai intégré une touche pour activer le changement de perspective, en conservant la position du joueur entre les deux vues. Cela permet d’avoir deux manières de percevoir le même environnement, ce qui est la mécanique principale du jeu.

En plus de la caméra 2D/3D, j’ai intégré un système de caméra orbitale. Grâce à une touche dédiée, la caméra peut pivoter autour du joueur à différents angles. Cela permet de mieux observer l’environnement en 3D et de visualiser les solutions du puzzle sous un autre angle. Le système repose sur un tableau d’angles prédéfinis et une logique d’incrémentation.

#### Constructeur



#### Méthodes

##### 1. DimensionCamera (Initialisation)

Le constructeur initialise les variables associées à la caméra, y compris la position du joueur, l'angle (en radians), les directions de mouvement, ainsi que la configuration de la caméra cible et de la transition entre les vues.

* Camera : Initialise la caméra avec la méthode DefaultPosition().
* targetCamera : Représente la caméra cible vers laquelle la caméra actuelle va se déplacer lors de la transition.
* transitionTime : Variable de suivi de la progression de la transition entre les vues.
* transitionDuration : Durée totale de la transition entre les vues, définie ici comme 0.5f (0.5 seconde).
* isTopView : Indicateur de la vue de dessus. Il est initialisé à false (vue par défaut).

##### 2. DefaultPosition

Cette méthode initialise la caméra avec une vue en perspective standard.

* Calcul de l'offset : L'offset de la caméra est calculé en fonction de l'angle (Radians), de manière à positionner la caméra à une distance de 10 unités derrière le joueur, avec une élévation de 1 unité en Y.
* Initialisation de la caméra : La caméra est configurée avec une perspective (CameraProjection.Perspective), une position calculée à partir de l'offset, et une cible qui est la position actuelle du joueur.
* La méthode retourne la caméra initialisée pour être utilisée par la classe.

##### 3. TopViewPosition

Cette méthode configure la caméra pour qu'elle adopte une vue de dessus (vue orthographique).

* Calcul de la direction de mouvement : Le vecteur de direction du mouvement est calculé en fonction de l'angle (Radians), mais dans ce cas, il est utilisé pour orienter la cible de la caméra par rapport au joueur.
* Initialisation de la caméra de vue de dessus : La caméra est positionnée à une hauteur élevée (200 unités en Y) et regarde le joueur avec une vue orthographique (CameraProjection.Orthographic). La cible est ajustée pour être au niveau du sol (Y = 0).
* Retour de la caméra cible : La caméra est prête à être utilisée pour la vue de dessus.

##### 4. ToggleCameraView

Cette méthode permet de basculer entre la vue par défaut et la vue de dessus.

* Alternance des vues : La méthode inverse la valeur de isTopView. Si la vue actuelle est la vue de dessus, elle revient à la vue par défaut, et inversement.
* Transition : Lors du basculement, la caméra cible est mise à jour pour correspondre à la vue appropriée, et la transition commence à zéro (transitionTime = 0.0f).

##### 5. UpdateCamera

Cette méthode met à jour la position de la caméra et gère la transition entre les vues.

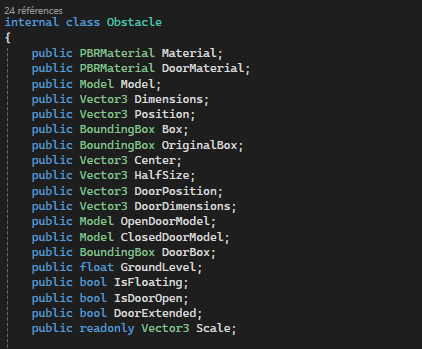
* Transition : Si le temps de transition est inférieur à la durée de la transition, la méthode met à jour la position de la caméra actuelle pour la rapprocher de la caméra cible, en interpolant entre les positions, cibles et autres paramètres de la caméra (comme Up, FovY, et Projection).
* Interpolation : Utilise la fonction Vector3.Lerp pour interpoler de manière linéaire entre les positions et cibles de la caméra, et Raymath.Lerp pour interpoler le champ de vision vertical (FovY).
* Fin de la transition : Lorsque le temps de transition atteint ou dépasse la durée définie, la caméra atteint sa position cible et la transition est terminée.

### Obstacle

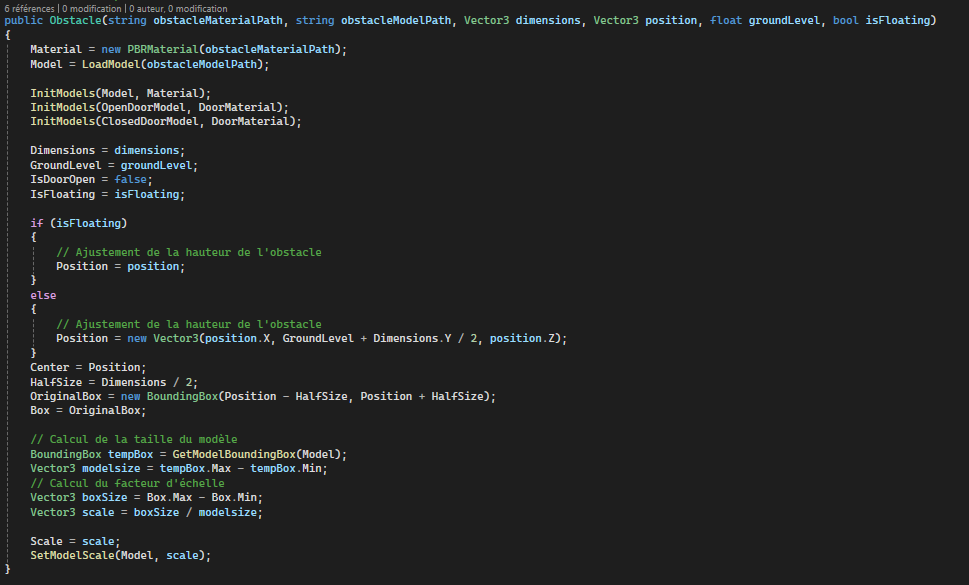
Obstacle gère les objets statiques qui bloquent le joueur et qui peuvent être surpassés en mode 2D. Il y a la possibilité de créer des obstacles de toutes les dimensions souhaitées grâce à la propriétés « Scale » qui est utiliser dans une méthode que nous verrons plus loin.

Pour information un obstacle a deux constructeurs l’un étant uniquement l’obstacle, l’autre possédant une porte sur une des faces.

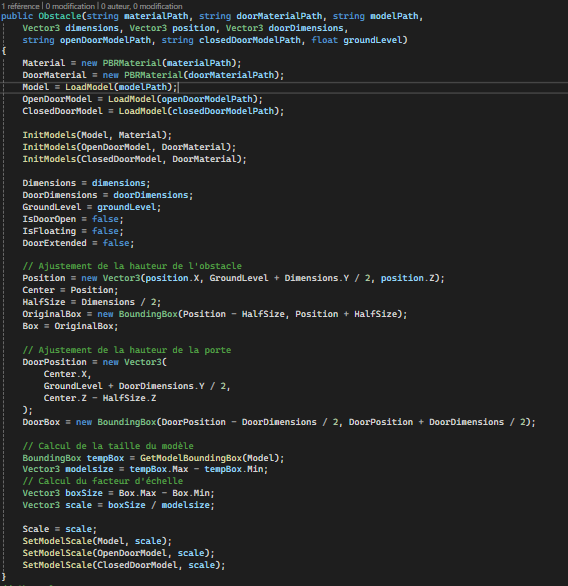
#### Propriétés



#### Constructeur (sans porte)



#### Constructeur (avec porte)



#### Méthodes

##### 1. ToggleDoor

Cette méthode permet de basculer l'état de la porte de l'obstacle, entre ouverte et fermée.

* Basculement de l'état de la porte : Si la porte est actuellement ouverte (IsDoorOpen est vrai), elle sera fermée, et vice versa. Cela se fait simplement en inversant la valeur du booléen IsDoorOpen.

##### 2. InitModels

Cette méthode initialise les modèles en appliquant des tangentes et en définissant leur matériau.

* Génération des tangentes : Pour chaque maillage du modèle, cette méthode génère les tangentes nécessaires pour un rendu correct (notamment pour les effets de lumière).
* Application du matériau : Si le modèle a des matériaux, le matériau spécifié dans l'argument de la méthode est appliqué au modèle pour déterminer son apparence visuelle (comme la texture et les propriétés physiques).

Cette méthode se répète dans d’autres classes.

##### 3. SetModelScale

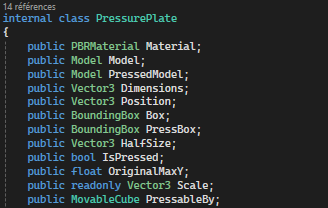
Cette méthode applique un facteur d'échelle sur chaque sommet du modèle.

* Application de l'échelle aux sommets : Pour chaque maillage du modèle, la méthode ajuste la position des sommets en multipliant leurs coordonnées par les valeurs des facteurs d'échelle (X, Y, Z), ce qui redimensionne le modèle.
* Assurance de la mise à l'échelle correcte : Cette mise à l'échelle est appliquée à chaque sommet de chaque maillage du modèle, ce qui garantit que tous les éléments du modèle sont proportionnellement redimensionnés.

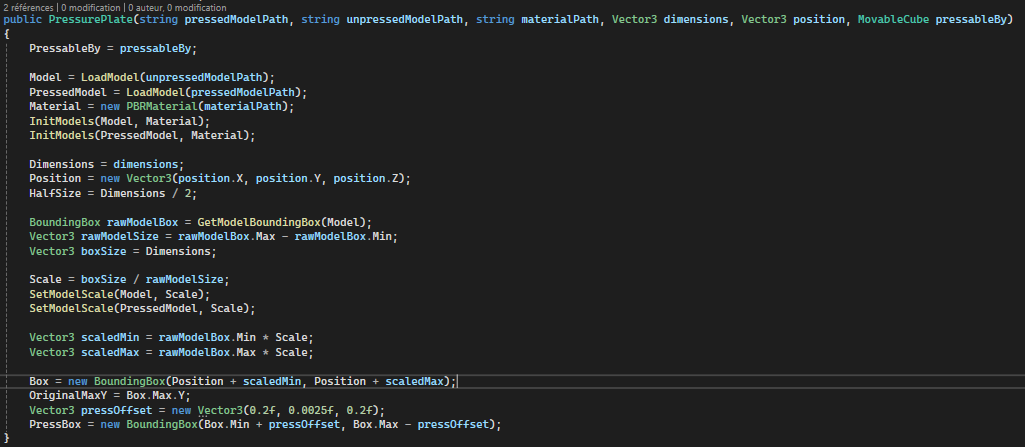
### PressurePlate

PressurePlate représente des plaques au sol qui réagissent à la présence d’un cube. La propriété « PressableBy » définit quel bloc est autorisé à presser la plaque.

#### Propriétés



#### Constructeur



#### Méthodes

La classe PressurePlate n’a pas de nouvelle méthode elle utilise également « InitModels » et « SetModelScale » qui ont été expliquée précédemment.   
C’est comme un booléen mais un peu plus poussé, puisqu’il est déclenché par la pression d’un cube spécifique et il est représenté par une plaque qui s’abaisse ou qui s’élève.

### MovableCube

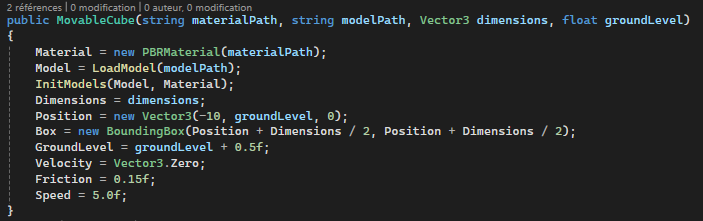
MovableCube permet de manipuler des blocs, les pousser et les placer pour activer les plaques.

Cette classe a le même comportement que le joueur il suffit de changer les caractéristiques comme suit :

* Déplacement par les touches 🡪 déplacement par une poussée extérieure
* Changement de vue avec la caméra 🡪 ne peut pas interagir avec la caméra,   
  mais subit les effets de la 2D.
* Peut presser toutes les plaques 🡪 peut presser uniquement la plaque de sa couleur.
* Ne peut pas récupérer de clés

#### Propriétés

#### Constructeur



#### Méthodes

Les cubes déplaçables imitant le joueur dans les méthodes :

* Update
* Brake
* HandleCollsiion
* HandlePlate

Il est inutile de réexpliquer leur fonctionnement. Le mouvement ne venant pas des entrées utilisateurs mais d’une poussée extérieure il y a uniquement une seule nouvelle méthode :

##### HandlePush

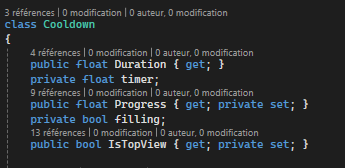
Cette méthode gère la logique de poussée entre le joueur et les cubes mobiles, ainsi que la gestion des collisions avec d'autres cubes mobiles.

* Interaction avec le joueur : Si le cube entre en collision avec le joueur (CheckCollisionBoxes), la direction de la poussée est calculée en fonction de la vitesse du joueur dans les axes X et Z. Cette direction est normalisée et appliquée au cube pour le pousser dans la même direction.
* Collision avec d'autres cubes : Ensuite, la méthode vérifie si le cube entre en collision avec d'autres cubes mobiles dans la scène. Si tel est le cas, la direction de poussée est calculée entre le centre du cube actuel et celui avec lequel il entre en collision. Cette direction est ensuite utilisée pour appliquer une poussée au cube en collision, tout en inversant la direction de la poussée pour l'autre cube.

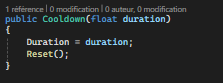
### Cooldown

Le cooldown permet l’affichage d’un temps de recharge pour le mode 2D. Lorsque le temps est écoulé le mode 2D se désactive et le temps se recharge automatiquement.

#### Propriétés



#### Constructeur



#### Méthodes

##### 1. Update

Cette méthode met à jour le statut du cooldown, en ajustant la progression et en gérant le passage d'un état à l'autre en fonction du temps écoulé (deltaTime).

* Gestion en vue de dessus : Si IsTopView est vrai, la méthode augmente le timer avec deltaTime et diminue la Progress (la progression du cooldown) jusqu'à ce qu'elle atteigne zéro, signalant que le cooldown est "rempli". Si timer atteint la durée totale (Duration), la vue passe à un autre état et le timer est réinitialisé.
* Gestion du remplissage : Si IsTopView est faux et que l'état de remplissage est actif (filling), la progression (Progress) est augmentée. Une fois que la progression atteint 100% (valeur de 1.0f), l'état de remplissage se termine, et la barre de cooldown est de nouveau prête à être utilisée.

##### 2. ToggleView

Cette méthode permet de basculer entre les états de vue ("top view" et "normal view"). Elle gère également la logique pour ne pas basculer tant que le cooldown n'est pas terminé.

* Basculement de la vue : Si IsTopView est actuellement désactivée et que la progression est au maximum (Progress >= 1.0f), la vue passe en mode "top view", et le timer est réinitialisé. Si la vue est déjà en mode "top view", elle bascule en mode normal, et le cooldown commence à se remplir à nouveau.

##### 3. Reset

Cette méthode réinitialise toutes les valeurs de l'état du cooldown.

* Réinitialisation : Elle remet à zéro le timer, redonne une progression maximale de 1.0f à Progress et réactive l'état de non-remplissage (filling = false), ce qui permet de recommencer le processus de cooldown.

## Programme principal

Le rôle du programme principal est le suivant, il s’occupe de :

* La navigation entre les différentes fenêtres
* La déclaration et instanciation de tous les éléments du niveau
* L’interaction entre les classes
* L’affichage et animation des objets 3D
* Jouer les sons interactifs
* Jouer la musique ambiante

# Résultats

Le projet final correspond aux attentes initiales que je m’étais fixées : proposer une base de jeu fonctionnelle qui intègre une mécanique de changement de dimension (2D/3D) et qui repose sur des interactions simples mais cohérentes. Le résultat obtenu m’a permis de valider la faisabilité de ce concept en C avec Raylib.

* Le changement de caméra fonctionne sans bug visible. Le joueur peut basculer à tout moment entre la vue 3D libre et une vue orthographique figée, ce qui permet de voir le niveau sous deux angles très différents. Cela ouvre la porte à des mécaniques de puzzle plus poussées à l’avenir, comme cacher ou révéler des chemins uniquement visibles dans une dimension.
* Le système de mouvement est fluide, avec une bonne réactivité. Le saut et la gravité sont bien gérés, et les collisions avec les murs ou le sol sont respectées. Le joueur ne peut pas sortir de la scène, ni traverser les objets solides. Les cubes peuvent être poussés dans toutes les directions, et leur comportement reste stable. On ne peut pas les pousser à travers les murs ni les bloquer entre deux obstacles.
* Les interactions avec les plaques de pression fonctionnent également comme prévu : elles s’activent lorsqu’un cube ou le joueur reste dessus. Elles déclenchent ensuite un événement, par exemple la disparition d’un mur, simulant une porte qui s’ouvre. Ce système est modulaire et peut être réutilisé dans plusieurs scénarios.
* La caméra orbitale autour du joueur ajoute un confort de jeu en 3D, surtout pour observer la disposition des éléments. Elle n’est pas indispensable mais améliore l’expérience.
* Le visuel est simple mais propre. Les textures sont rudimentaires, mais les modèles 3D importés donnent un peu plus de personnalité au niveau. Les illustrations faites à la main dans Krita m’ont permis de clarifier mes idées et d’expliquer certains éléments dans la présentation.
* Enfin, le code source est bien structuré, réparti dans plusieurs fichiers avec des commentaires pour faciliter la lecture. Le projet est versionné sur GitHub, avec des commits réguliers qui retracent l’évolution du travail.

# Conclusion

Ce projet m’a permis de découvrir le développement d’un petit moteur de jeu en C avec Raylib. Je suis parti d’un environnement complètement vide pour arriver à un prototype fonctionnel, avec plusieurs classes, des interactions entre objets et une mécanique centrale bien intégrée.

Problèmes rencontrés :

* L’un des plus gros défis a été la gestion des collisions et du mouvement. Il fallait que le joueur se déplace de manière fluide, mais qu’il respecte les limites du décor. De même, pousser un cube peut paraître simple sur le papier, mais en pratique cela demande une gestion fine des angles, des murs et des doubles collisions. J’ai dû faire pas mal de tests pour obtenir un résultat cohérent.
* Un autre aspect complexe a été le changement de dimension. Passer d’une vue 3D à une vue 2D sans tout casser au passage a demandé une bonne compréhension de la caméra dans Raylib. J’ai dû bien réfléchir à ce que je voulais conserver d’une dimension à l’autre, comme la position du joueur ou les objets visibles.

Même si le résultat final reste simple, je pense avoir bien posé les bases pour un jeu plus complet. Les éléments mis en place sont modulables et pourraient facilement être réutilisés pour construire de vrais niveaux, avec plus d’objets, des énigmes plus complexes, ou même un système de sauvegarde.

Ce projet m’a aussi permis d’améliorer ma gestion de projet. J’ai mieux organisé mon temps, structuré mon code et appris à documenter mon travail. Grâce au journal de travail et aux visuels que j’ai produits en parallèle, j’ai pu suivre mes progrès semaine après semaine et corriger mes erreurs plus rapidement.

Au final, je suis satisfait de ce que j’ai produit. J’ai réussi à aller au bout d’un concept qui me motivait, tout en apprenant à jongler entre design, programmation et technique. Ce pré-TPI m’a bien préparé pour le TPI final, autant sur le plan technique que méthodologique.

# Annexes

Journal de travail complet  
→ Voir document joint : JT-PRE-TPI.docx  
Contient le détail de toutes les étapes de développement, les problèmes rencontrés et les solutions apportées. Chaque tâche y est décrite clairement avec des commentaires personnels sur la progression.

Présentation PowerPoint  
→ Voir document joint : Présentation\_DIMEN.pptx  
Présentation visuelle du projet, avec captures d’écran du jeu, schémas explicatifs, croquis dessinés à la main, et structure utilisée pour la soutenance orale.

Illustrations et croquis personnels  
– Croquis du fonctionnement de la caméra dimensionnelle  
– Dessin explicatif de la plaque de pression et du cube  
– Vue schématique de l’environnement de test (niveau)

Liens externes  
– Référentiel GitHub du projet : (à compléter selon ton lien)  
– Lien vers les modèles 3D utilisés (Sketchfab)  
– Documentation [Raylib](https://www.raylib.com/cheatsheet/cheatsheet.html)

Liste des fichiers du projet  
– main.c  
– player.c / player.h  
– cube.c / cube.h  
– obstacle.c / obstacle.h  
– camera.c / camera.h  
– pressureplate.c / pressureplate.h  
– assets/ (dossier contenant les modèles, textures et sons)

Logiciels utilisés  
– Visual Studio 2022  
– Raylib (librairie graphique C)  
– Krita (dessin des croquis et visuels)  
– GitHub (versionnage)  
– PowerPoint (présentation finale)