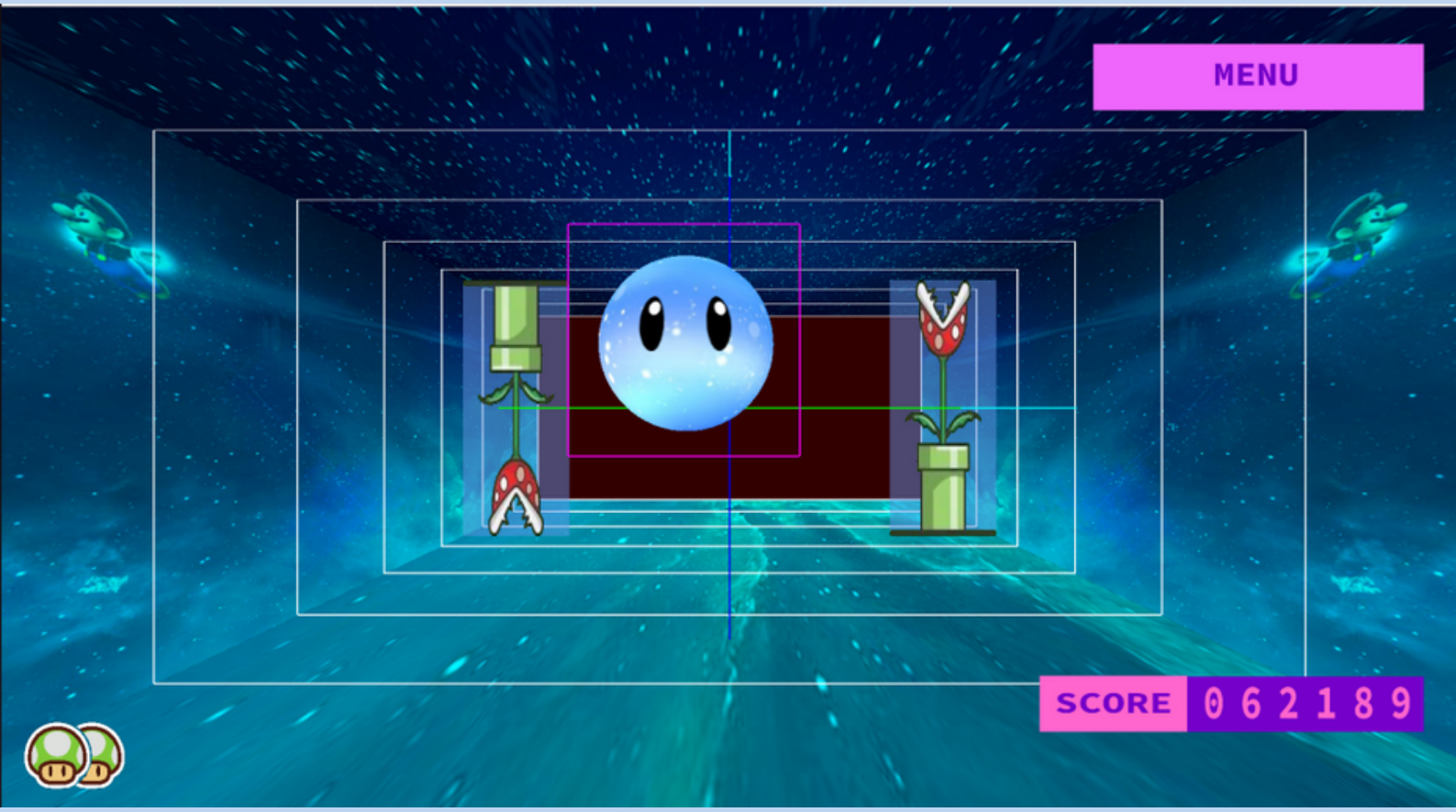
SUPER JEU DE LA MORT QUI TUE



Maureen Grandidier - Tanya François Gaillard Nina

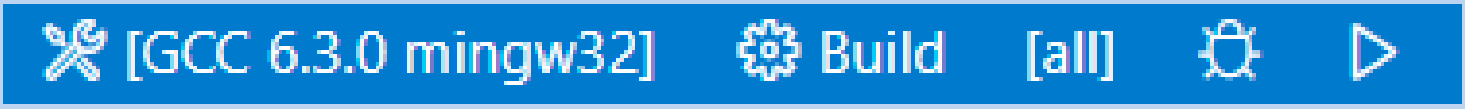
SOMMAIRE

MODE D'INSTALLATION

Nous étions trois sur le projet, Tanya et Maureen utilisant Linux, elles peuvent compiler et exécuter en ligne de commande avec Cmake :

```
$ make project && ./bin/project.out
```

Nina travaillait avec la template d'Enguerrand sur Windows, ce qui lui permettait de compiler et exécuter avec Cmake en utilisant le kit GCC de mingw32.



Par ailleurs nous utilisons la bibliothèque SOIL pour le chargement des PNG ce qui demande d'installer la bibliothèque et des commandes en plus.

```
Ajoutez le chemin du répertoire contenant libSOIL.so.1 à la variable d'environnement
LD_LIBRARY_PATH de manière temporaire (à faire dès qu'on ouvre un terminal). Vous
pouvez le faire en exécutant la commande suivante dans un terminal :

export LD_LIBRARY_PATH=/usr/local/lib:${LD_LIBRARY_PATH}
```

LE JEU

Lorsque l'on exécute le code, c'est le menu qui s'affiche en premier lieu. Grâce à la souris, on peut choisir si on veut jouer, choisir un niveau ou quitter le menu. Si l'on clique sur jouer, le jeu commence au niveau 1. On appuie sur clic droit pour lancer la balle. Le joueur ne peut pas avancer tant qu'il n'a pas lancé la balle. Il ne peut également pas avancé si la raquette se trouve devant un obstacle. Le joueur avance en restant appuyé sur clic gauche. La raquette se déplace avec les mouvements de la souris.

FONCTIONNALITÉS

Nous avons travaillé avec des classes. Nous avons donc une classe Game, Corridor, Ball, Enemy, Menu et Racket.

La classe Game représente le jeu dans son ensemble. Elle contient entre autre une balle, un corridor, une raquette et un menu. L'objet game représente un peu le contrôleur de notre jeu. C'est lui qui contient la vie du joueur ou bien encore le score du jeu. Les méthodes getter permettent d'accéder aux objets individuels du jeu, tels que la balle, le corridor et la raquette. C'est lui également qui gère le gameOver (écran de fin) et le fait de perdre des vies.

La classe Corridor défini les coordonnées du corridor, le point de départ, la distance parcourue, le kilométrage et la vitesse. La classe fournit des méthodes pour dessiner le corridor et ce qu'il contient (les murs, les ennemis, les bonus...). C'est ici aussi qu'on réalise le chargement des ennemis et en soit des levels. En effet, tout nos ennemis sont stockés sur un fichier texte.
Le premier élément correspond au "type" d'ennemis, et le deuxième pour savoir où il est placé dans le corridor.
Comme on peut le voir dans la fonction loadEnemys, si on commence du level 1 alors tout le jeu sera chargé (de la ligne 0 à la ligne 100 du fichier txt) mais si nous commençons du level 2 alors on chargera à partir de la ligne 51 soit à partir de 1479km.

```
V_M_L 35
V_M_R 35
V_L_R 82
H_M_U 114
```

C'est également le corridor qui permet au joueur d'avancé (on soustrait la variable m_walk du corridor aux objets qui doivent bouger quand le joueur avance comme les obstacles et les bonus. m_walk est égale à 0 quand le joueur est immobile et on lui additionne une vitesse quand le joueur est en marche.

```
end = 101;

if(level == 1)
{
    start = 0;
    adapt = 0;
}
else
{
    start = 51;
    adapt = 1440;
}
```

Lorsqu'on récupére le nom/type de l'ennemi grâce au fichier texte, on va aller utiliser un dictionnaire pour créer un objet ennemi avec des caractéristiques précises (c'est plus pratique que de retaper de longues lignes de variable pour construire correctement les obstacles car ils sont de taille et d'orientation différente). Le dictionnaire se trouve dans dataEnemy.

```
#define H 12
#define L 25

void initializeEnemyMap(
    std::map<std::string, std::tuple<int, int, int, int, int, int>> &enemyMap) {
  // int w, int h, int d, int points, int left, int up
  /* ********** V E R T I C A L ********** */
  enemyMap["V_L_L"] = std::make_tuple(7, H, 10, 90, 1, -1);
  enemyMap["V_L_R"] = std::make_tuple(7, H, 10, 90, 0, -1);
  enemyMap["V_M_L"] = std::make_tuple(10, H, 10, 70, 1, -1);
  enemyMap["V_M_R"] = std::make_tuple(10, H, 10, 70, 0, -1);
  enemyMap["V_B_L"] = std::make_tuple(L, H, 10, 40, 1, -1);
  enemyMap["V_B_R"] = std::make_tuple(L, H, 10, 40, 0, -1);

  /* ********** H O R I Z O N T A L ********** */
  enemyMap["H_L_U"] = std::make_tuple(L, 4, 10, 90, -1, 1);
  enemyMap["H_L_B"] = std::make_tuple(L, 4, 10, 90, -1, 0);
  enemyMap["H_M_U"] = std::make_tuple(L, 8, 10, 70, -1, 1);
  enemyMap["H_M_B"] = std::make_tuple(L, 8, 10, 70, -1, 0);
  enemyMap["H_B_U"] = std::make_tuple(L, H, 10, 40, -1, 1);
  enemyMap["H_B_B"] = std::make_tuple(L, H, 10, 40, -1, 0);

  /* ********** S Q U A R E ********** */
  enemyMap["L_L_B"] = std::make_tuple(7, 7, 10, 100, 1, 0);
  enemyMap["L_R_B"] = std::make_tuple(7, 7, 10, 100, 0, 0);
  enemyMap["L_L_U"] = std::make_tuple(7, 7, 10, 100, 1, 1);
  enemyMap["L_R_U"] = std::make_tuple(7, 7, 10, 100, 0, 1);

  enemyMap["M_L_B"] = std::make_tuple(10, 10, 10, 80, 1, 0);
  enemyMap["M_R_B"] = std::make_tuple(10, 10, 10, 80, 0, 0);
  enemyMap["M_L_U"] = std::make_tuple(10, 10, 10, 80, 1, 1);
  enemyMap["M_R_U"] = std::make_tuple(10, 10, 10, 80, 0, 1);

  enemyMap["B_L_B"] = std::make_tuple(15, 15, 10, 50, 1, 0);
  enemyMap["B_R_B"] = std::make_tuple(15, 15, 10, 50, 0, 0);
  enemyMap["B_L_U"] = std::make_tuple(15, 15, 10, 50, 1, 1);
  enemyMap["B_R_U"] = std::make_tuple(15, 15, 10, 50, 0, 1);
}
```

De plus, game contient le score qui est une addition des km parcouru et des bonus ramassés. Un bonus vaut 100 points. Un joueur ne pourra jamais dépassé les 5 vies mais il continue à gagner des points s'il ramasse les bonus vie.

La classe Ball défini les coordonnées de la balle, sa taille et ses vitesses dans les directions X, Y et Z. La classe fournit des méthodes pour dessiner la balle, gérer les collisions avec la raquette, les murs du corridor et les enemis.

La classe Enemy défini la largeur, la hauteur, la distance, la position de l'ennemi. La méthode "setD" permet de modifier la distance de l'ennemi, tandis que la méthode "setDWithWalk" permet de mettre à jour la distance en fonction de la marche effectuée.

Nous avons mis un nombre de points aux obstacles car nous partions sur le fait de leur donner une vie et de les faire disparaitre au bout de x collisions avec la balle. Cependant, par manque de temps nous n'avons pas pu implémenter cette fonctionnalité.

La classe Racket définie les coordonnées de la raquette, sa longueur, sa position, sa vitesse et son mode d'affichage. La méthode "drawRacket" permet de dessiner la raquette en utilisant les coordonnées et le mode d'affichage. Les méthodes getter permettent de récupérer les valeurs des attributs, tandis que les méthodes setter permettent de modifier la position, le mode d'affichage et de mettre à jour la position de la raquette.

FONCTIONNALITÉS ANNEXES

- Nous proposons deux packs de texture. Il faut pour cela appuyer sur la touche A en Azerty et Q en Qwerty pour changer de texture.

- Le joueur peut choisir de commencer au niveau 1 ou 2. S'il meurt, il reprendra de niveau choisi.
- Pouvoir charger des données grâce à des fichiers txt. Bien pratique pour les ennemis ou les textures.

AMELIORATION

- Mettre une vie aux obstacles
- Faire bouger les obstacles
- Rajouter du son
- Esthétique du jeu

MENU

JOUER

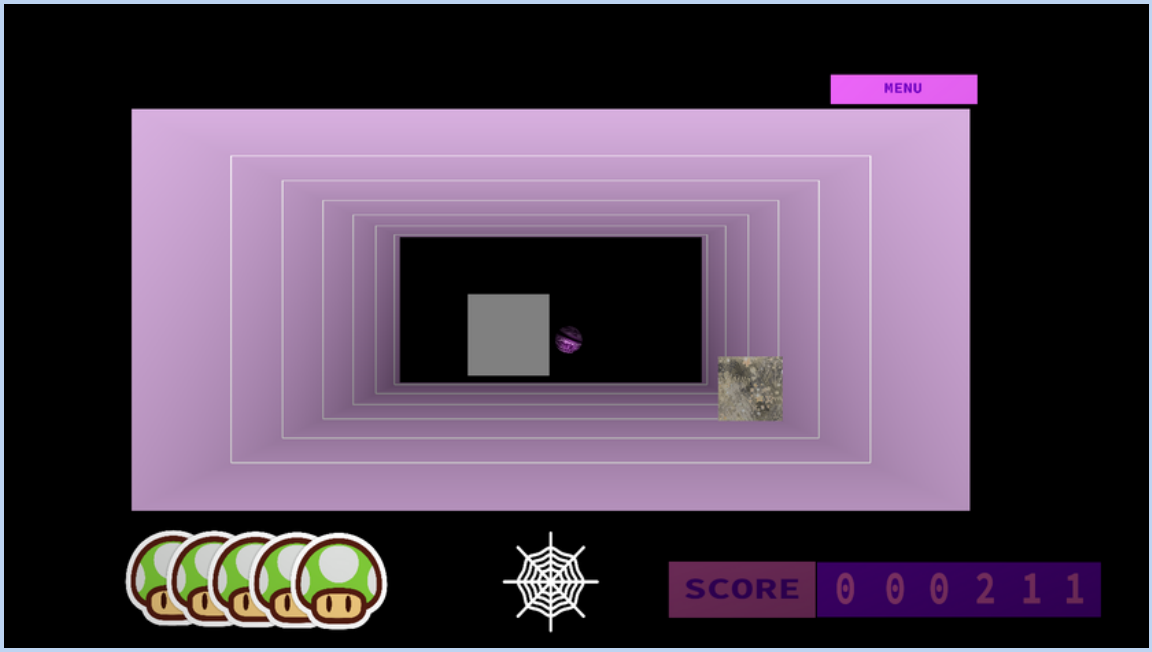
NIVEAUX

QUITTER

NIVEAU 1

NIVEAU 2

RETOUR



FIN

SCORE

0 6 2 1 8 9

REJOUER

QUITTER

RÉPARTITION DU TRAVAIL

Nous avons utilisé Trello afin de se répartir le travail. À chaque fois qu'une de nous commençait à travailler sur quelque chose, elle créait une carte à son nom. Nous avions six colonnes, dont 3 (à faire, en cours, terminé) pour le design et les trois mêmes pour la partie dev.

Nous avions également un serveur discord dans lequel nous postions nos avancées et éventuels problèmes.

Nous avons tout d'abord créé trois branches sur le git de Tanya afin d'y mettre chacune nos avancées. Maureen a fait un premier commit du corridor dans la main que nous avons pu récupérer dans le but d'avoir une base.

Nina utilisant le template d'Enguerrand, elle n'a pu faire que des commits dans sa branche qui ont été récupérés par Maureen et Tanya par la suite, elle s'est donc occupée de tout ce qui était de l'affichage, c'est-à-dire Menu, score, ball, le tout relié à des textures et un début de light afin d'avancer plutôt indépendamment de Maureen et Tanya.

Tanya s'est occupée de la raquette, de gérer les collisions, et Maureen a donc créé le corridor en premier lieu, géré les ennemis et les bonus, ainsi qu'importé le code de Nina dans le main.

DÉTAILS TECHNIQUES

La gestion de nos collisions se fait dans les classes Ball, Corridor et Game.

Les fonctions de gestion des collisions de la classe Ball gèrent les collisions entre la balle et différents éléments du jeu.

La fonction collisionRacket vérifie si la balle entre en collision avec la raquette, en comparant les positions et les dimensions des deux objets. Si la collision a lieu, la fonction ajuste la vitesse de la balle en fonction de l'angle de collision et active un bonus de "colle" si celui-ci est activé. L'angle est calculé selon la distance avec le centre de la raquette.

La fonction collisionCorridor vérifie si la balle entre en collision avec le corridor, en comparant les positions et les dimensions. Elle renvoie un code indiquant quelle partie du corridor a été touchée.

La fonction collisionEnemy vérifie si la balle entre en collision avec l'un des ennemis, en comparant les positions et les dimensions. Si la collision a lieu, la vitesse de la balle est ajustée en fonction de l'angle de collision.

Enfin, la fonction collision regroupe toutes les vérifications de collision et renvoie un code indiquant quel type de collision s'est produit.

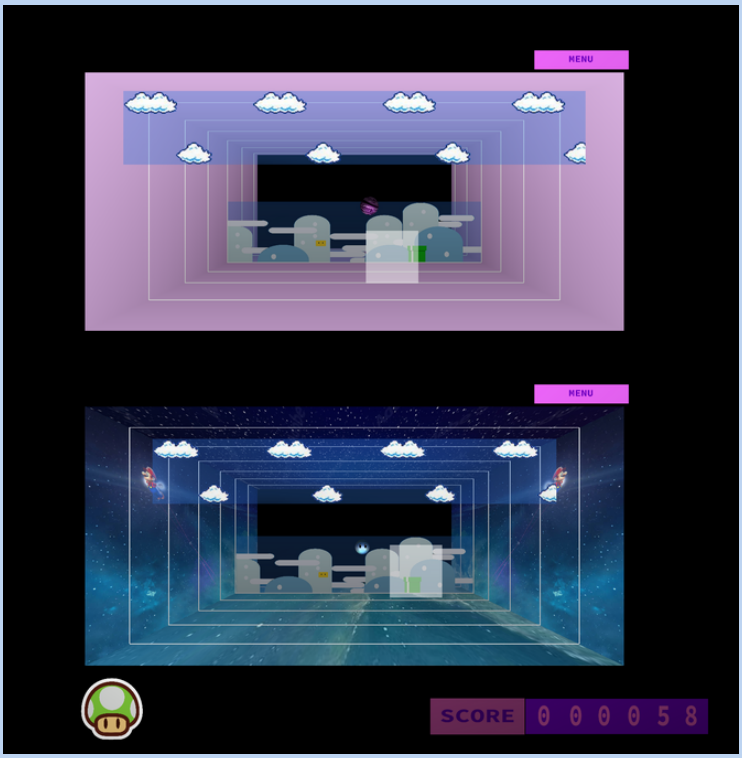
Les fonctions de gestion des collisions de la classe Corridor gèrent les collisions entre le corridor et d'autres éléments du jeu.

La fonction collisionRacket vérifie si la raquette entre en collision avec l'un des ennemis présents dans le corridor. Elle compare les positions et les dimensions des objets pour détecter la collision.

La fonction collision appelle la fonction collisionRacket pour vérifier si la raquette entre en collision avec les ennemis. Si aucune collision n'est détectée, la fonction met à jour l'état du corridor.

La fonction collision de la classe Game gère les collisions globales du jeu. Elle appelle les fonctions de collision de la balle et du corridor, en passant les ennemis et les positions pertinentes en paramètres. Si un mouvement est autorisé, la fonction gère les collisions de la balle avec les autres éléments et met à jour les positions.

D'ailleurs, nous avons mis la possibilité de mettre des textures ou non au mur (Touche A). Et les textures sont modifiables en changeant le fichier loadImg.txt



DIFFICULTÉS

Light

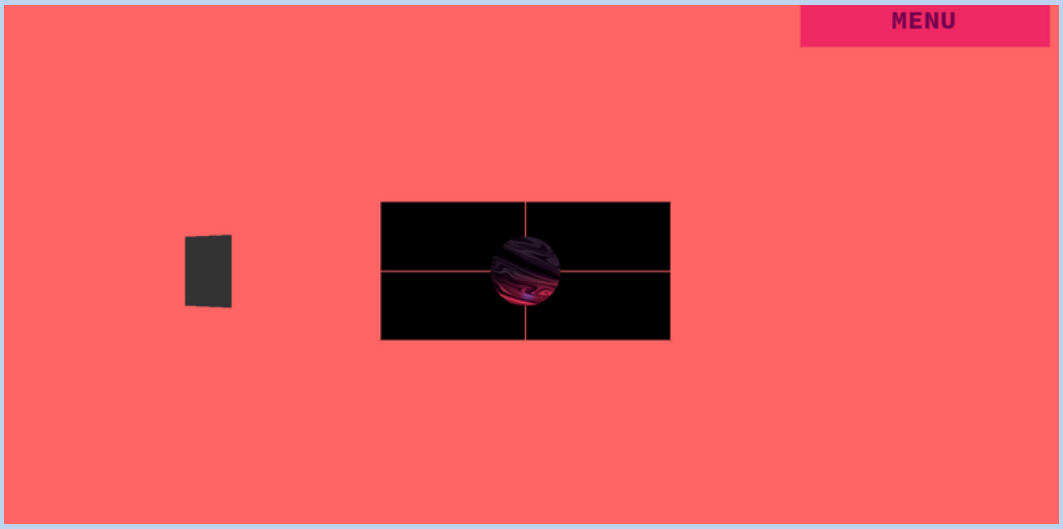
Nous avons eu des difficultés à gérer la lumière. Tout d'abord, ce code nous permettait de gérer la couleur ambiante, la diffusion et la brillance.

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);

    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, light_direction);
    GLfloat globalAmbientColor[] = { 1.0f, 1.0f, 1.0f, 1.0f };  // Couleur ambiante globale (RGB)
    GLfloat globalDiffuseColor[] = { 1.0f, 1.0f, 1.0f, 1.0f };  // Couleur diffuse globale (RGB)
    GLfloat globalSpecularColor[] = { 1.0f, 1.0f, 1.0f, 1.0f }; // Couleur spéculaire globale (RGB)
    GLfloat globalShininess = 90.0f; // Brillance globale du matériau

    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, globalAmbientColor);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, globalDiffuseColor);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, globalSpecularColor);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, globalShininess);
    drawFrame();
```

Seulement, le couloir et les textures du menu prenaient la couleur de la lumière, qu'elle soit blanche ou colorée. Cependant, nous avons réussi à insérer un minimum de lumière dans notre scène. Mais, nous n'avons pas pu mettre de source de lumière depuis la balle.

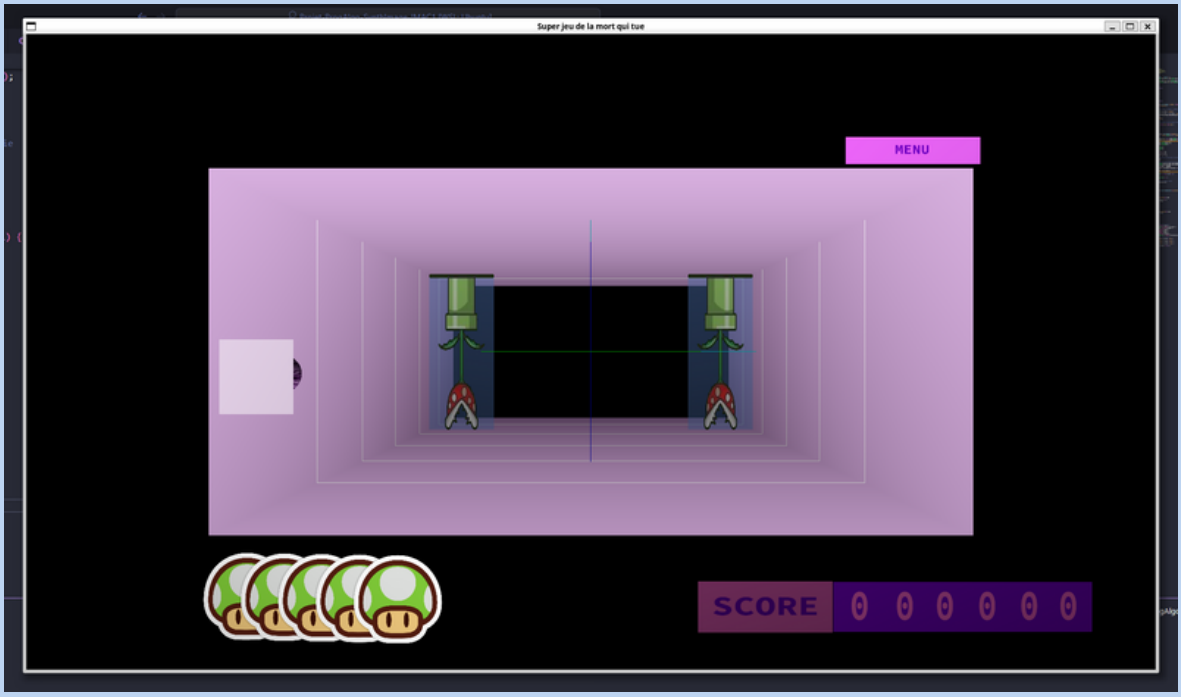


Collisions

Nous avons eu beaucoup de difficultés avec les collisions. En effet, dans la configuration de départ, c'était assez compliqué d'atteindre les vrais coins du couloir, et donc la balle sortait.

Pour pallier ce problème, nous avons changé notre angle de vue, et diminué la zone de jeu afin d'avoir le couloir en entier d'affiché.

L'espace en plus, permet de pouvoir afficher les éléments comme les vies, le score et le menu.



De plus, nous avions des soucis de détections de collisions, en effet la balle avait tendance à traverser les murs lorsque l'on prenait beaucoup de vitesses.

Au final le résultat reste satisfaisant malgré encore quelques soucis de collisions.

GIT

Comme d'habitude avec GIT, les soucis de merge prennent beaucoup de temps à être débogué.

CE QU'ON RETIENT

Nina

J'ai beaucoup appris durant ce projet et durant ce semestre en général. Faire des choses artistique me donne d'avantage envie de coder et m'a fait progresser en programmation de manière générale. J'ai l'impression d'avoir compris et réussi à faire beaucoup plus de choses qu'avant.

Maureen

J'ai beaucoup aimé ce projet mais je suis un peu frustrée. D'habitude je personnalise un maximum mes projets (esthétisme, histoire, musique...) mais ici je n'ai malheureusement pas eu le temps (alors que c'est sûrement un des projets sur lequel j'ai travaillé le plus régulièrement. Je pense que la charge de travail était trop importante au vu de mes capacités mais j'ai tout de même hâte de recommencer !

Tanya

C'était la première que je faisais de la programmation 3D, malgré les difficultés que j'ai eues durant le projet (et le semestre), cela m'a vraiment donné envie de continuer de mon côté.