Rapport projet Digger

Groupe : Stacey CODJIA, Chanaël SIAR, Alice JACQUES

Objectif du projet : Créer un jeu en 2D s'apparentant au jeu du digger.

Organisation du projet :

Digger/
\--*assets (contenant dossier images)*
\--*CmakeUtils*
\--*lib*
\--*src/*

Nom des fichiers dans le dossier src servant à différentes fonctionnalités du jeu propres à leurs noms :

- *carte.cpp*
- *carte.hpp*
- *draw_scene.cpp*
- *draw_scene.hpp*
- *enemy.cpp*
- *enemy.hpp*
- *flow_field.cpp*
- *flow_field.hpp*
- *main.cpp*

Fonctionnalités implémentées

Génération de la carte :

Ici l'idée a été de créer une génération de carte de manière procédurale. Pour cela, nous avons utilisé un algorithme de type cellular automata. Le principe est dans un premier temps de remplir chaque case avec des 1 et des 0 de manière aléatoire, puis de compter les cases ayant une valeur de 1 autour d'une cellule. Ensuite, si une cellule de valeur 1 a au moins 4 cases pleines (de valeur 1) alors elle reste pleine, sinon elle devient vide. Inversement, si une case est vide (de valeur 0) a plus de 4 cases pleines voisines elle devient pleine. On répète l'action plusieurs fois pour avoir un résultat concluant.

On a aussi fait en sorte que la map soit constituée d'un seul chemin que le joueur peut parcourir. Cela permet de ne pas créer de zones où le joueur n'a pas accès alors qu'il est censé récupérer des choses à cet endroit.

Pour ce qui est de son contenu, il est possible de trouver :

- Des blocs vides représentés par du sable
- Des blocs plein représentés par de l'eau
- Des objets à collecter représentés par des diamants
- Des Obstacles représentés par des pierres

Le joueur :

Le joueur est positionné grâce à deux systèmes de coordonnées :

- **Coordonnées de grille**: utilisées pour la logique de déplacement et la détection de collision.

- **Coordonnées OpenGL**: utilisées pour l'affichage à l'écran.

Nous utilisons des fonctions de conversion (gridToWorld() et worldToGrid()) qui permettent de passer d'un système à l'autre. Le joueur se déplace à l'aide des flèches du clavier.

Le contrôle du joueur est géré dans la fonction key_callback() qui est déclenchée lors de la pression d'une touche. Seuls les déplacements vers des cases valides sont pris en compte. Sinon, la position du joueur reste inchangée.

Pour un déplacement plus fluide, nous manipulons plutôt des flottants. Le joueur pouvant donc se trouver entre deux cases, la vérification des collisions se fait aux quatres coins du joueur.

Les ennemis et le flow field pathfinding :

Le mouvement des ennemis est aléatoire pour que chacun soit indépendant. Un ennemi suit le joueur lorsqu'il est proche de lui d'environ 5 cases adjacentes. Lorsqu'un ennemi touche le joueur, la partie est perdue.

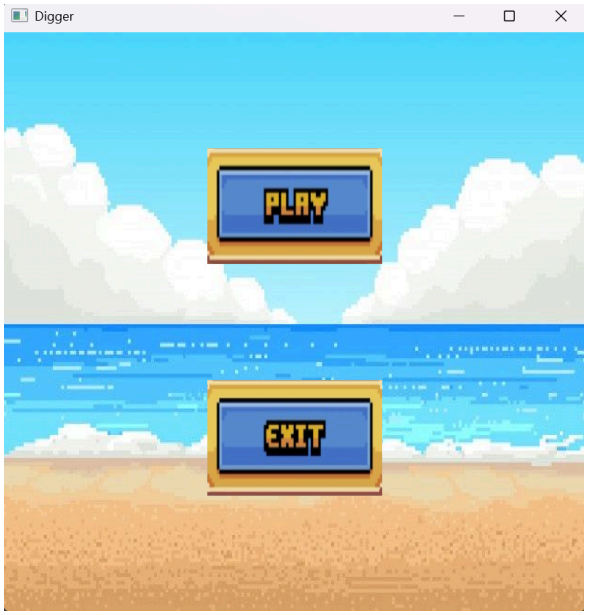
La fonction canMoveEnemy permet de vérifier s' il peut se déplacer à une position donnée, sans collisions. Pour cela, il calcule les 4 coins de la zone de l'ennemi et le convertit en indice de grille, et si un des coins du joueur se

retrouve en dehors de la carte ou dans une case qui n'est pas traversable, alors le mouvement est interdit.

A chaque frame, nous faisons une update de l'ennemi pour qu'il puisse se déplacer sur la carte au fil du temps et en fonction des actions du joueur. La méthode Enemy::update() nous permet de faire cela. Il y a un calcul de cases entre l'ennemi et le joueur et comme mentionné plus haut, si il y a un écart de 5 cases ou moins, l'ennemi poursuit le joueur (grâce à un vecteur pointant vers le joueur). Sinon, on choisit une direction aléatoire parmi bas, gauche, haut et droite.

Interface graphique :

L'interface graphique permet de visualiser le menu. Au départ, il y a une option pour jouer et une autre pour quitter.



Quand on joue, il est possible de quitter la partie à tout moment en faisant la touche echap.

Le joueur peut également mettre le jeu en pause en cliquant sur "Entrée".

De plus, il y a un indicateur de nombre d'objets à collecter qui s'actualise en même temps que la partie dans la console.

Nous avons également implémenté des écrans de victoire (lorsque le joueur collectionne toutes les germes) et de défaite (lorsqu'il est touché par un ennemi).

Sprites utilisés :

Pour ce projet, nous avons pris des sprites libres de droits sur la thématique de la piraterie.

Joueur Ennemi Diamant Téléportation Eau Sable

Bonus - Téléportation :

Pour notre fonctionnalité bonus, nous avons implémenté des portes spéciales qui permettent de téléporter le joueur aléatoirement sur la carte. Pendant la génération de la carte, les portes sont placées aléatoirement sur certaines cases vides avec une faible probabilité.

Le fonctionnement du mécanisme est assez simple : lorsque le joueur entre en contact avec une porte, il est automatiquement téléporté vers une case vide choisie aléatoirement parmi celles disponibles sur la carte.

Ergonomie :

Nous avons essayé de prendre en compte l'ergonomie dans notre projet. Cela passe par :

- la répartition des fonctions dans plusieurs fichiers en fonction de leurs rôles et responsabilités.
- Le code est commenté, cela permet de rendre nos codes plus accessibles.

- Les noms des fonctions sont explicites en fonction de leur utilité afin de pouvoir s'y retrouver plus facilement.

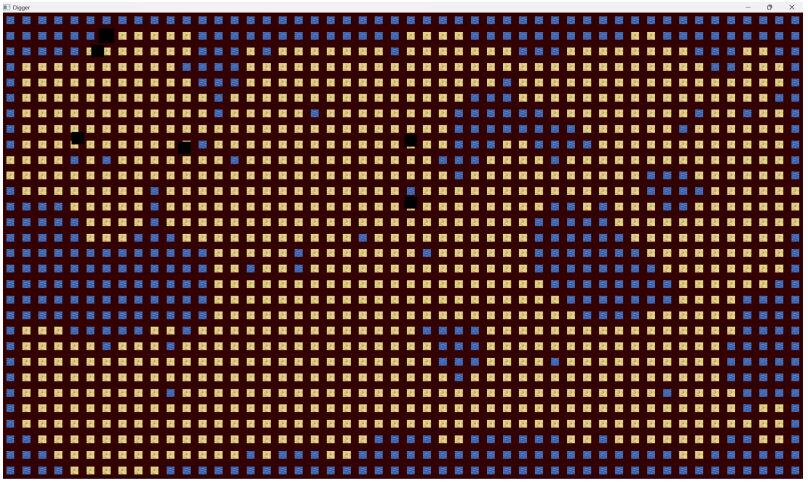
- Nous avons essayé d'utiliser des structures relativement simples comme les std::vector, les boucles for, etc.

Problèmes rencontrés :

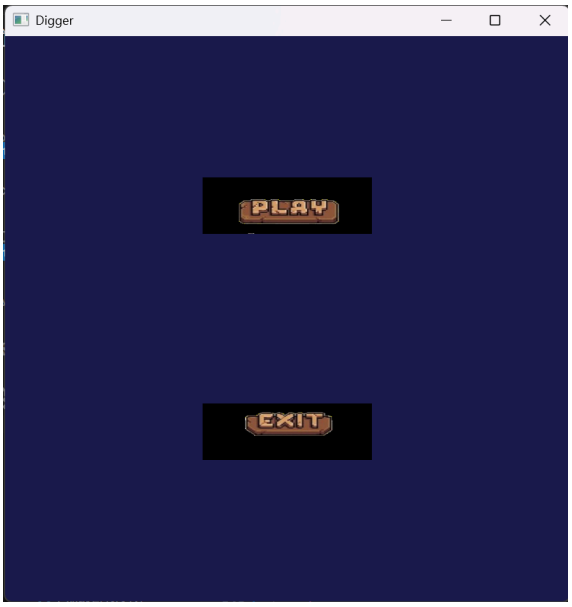
La carte : Le principal problème que nous avons eu au début a été l'affichage de la carte dans la fenêtre de rendu et pas seulement dans la console. Une des raison était qu'un fond et les cases de la carte n'étaient pas appelées dans le bon ordre.
Toujours concernant la carte, nous n'avions pas les bonnes dimensions de cases à cause d'un calcul qui divisait par 2 l'aire de la case.

Les textures : Tout au long du projet, nous n'avons pas réussi à afficher les textures.
Sur un projet de test à part, on avait réussi à les afficher en utilisant BasicRect qui était spécifique pour les textures, mais dans notre projet on dessine un carré et c'est pas du même type. Cela n'a pas fonctionné malgré beaucoup de tentatives. Grâce à l'aide des professeurs et de Océane, nous avons pu enfin réussir à les implémenter. Pour ce faire, on charge les images et on créer la texture. On les initialise dans le InitScene et on les utilise dans le drawMap. On utilise ensuite les mesh pour mapper la texture correctement sur les éléments. On a également utilisé glEnable(GL_BLEND) pour la transparence de certains fonds de sprite.

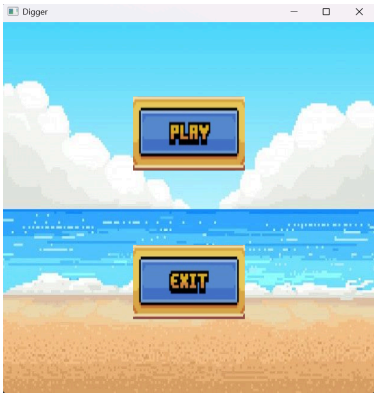
les problèmes à moitié résolues :

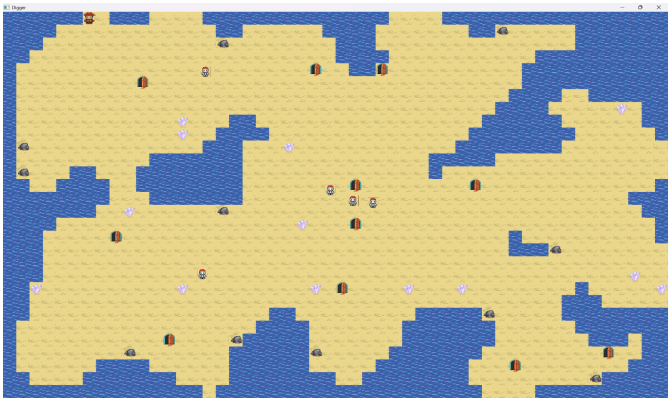


Sans texture :



Avec texture :





Axes d’amélioration :

Il y a certainement des améliorations possibles dans la structure et la propreté du code. Certaines fonctions auraient pu être optimisées.

Nous pourrions également améliorer la probabilité d'apparition de certains objets pour éviter de se retrouver avec une carte vide de temps en temps.