GUYENOT Amber

MEYER Thomas

Projet IF2B – Pavage

Introduction :

Pour ce projet, nous avons fait en sorte qu'un maximum de fonctions puisse être utilisé dans tous les cas possible, jouer seul ou à deux, en facile ou difficile, nous avons donc séparé le code en cinq fichiers C et quatre fichier header. Il y bien sûr le *main*, puis les autres associé à celui-ci qui sont : *MultiplayerGame*, *Singleplayergame*, *Utilities* et *save* avec chacun leur header associé.

Le fichier *main*, est le menu du jeu, et permet de commencer une nouvelle partie ou d'en charger une, et dans le premier cas, d'en donner les caractéristiques, mode de jeu, niveau de difficulté, taille de la grille de jeu. Puis la page fait appelle aux autres fichiers en fonction des choix faits.

Il y a ensuite le fichier *Utilities*, qui est l'un des plus important puisqu'il regroupe toutes les fonctions permettant l'affichage, la créations des tuiles, la vérification de placement ou de début de tours, le tour d'un joueur, et d'autre fonction plus petite permettant divers vérification.

Ce fichier est ensuite utilisé par les deux fichiers *MultiplayerGame* et *Singleplayergame*, dont l'utilisation est déterminée par les choix dans le main. Ces deux fonctions initialisent une partie pour une partie débuter puis rentre dans une boucle utilisant les fonctions de *Utilities* jusqu'à la fin de la partie ou l'abandon d'un joueur.

Le dernier fichier C sur lequel nous avons travaillé est *save* qui s'occupe de créer un fichier à partir d'une partie en cours ou de charger une partie qui a été sauvegardée.

Fichier main :

Pour le fichier *main*, la première partie se compose de plusieurs switch case qui permettent de se déplacer dans le menu, avec vérification des commandes mises par l'utilisateur.

Le premier regroupe les différents niveaux de menus qui permet de choisir d'abord si une nouvelle partie est lancée, une ancienne est chargée, ou quitter le jeu. Ensuite pour la création, le joueur peut retourner en arrière ou ouvrir la partie en mode 1 joueur ou 2 joueur.
L'utilisateur initialise ensuite la difficulté du jeu et la taille de la grille avec la fonction

newGame, qui est composé de boucles *while* donnant un message d'erreur en cas d'envois de caractère non attendu. Le joueur est ensuite redirigé vers les fonctions des fichiers *MultiplayerGame* et *Singleplayergame*.

Fichier *Utilities* :

Un point important du de *Utilities*, c'est la fonction *interpretChar* qui permet que tout les char qui sont gérés par les fonctions ne soient pas des nombre négatif tout en ayant un affichage et en faisant les calculs correctement. Pour gérer cela, avec un char en entré, la fonction crée une string avec « \0 » à sa fin et remplace le char en entré par le char réel auquel il se rapporte (0 = -3 / 1 = -2 / 2 = -1 / 3 = 0 / 4 = 1 / 5 = 2 / 6 = 3).

La fonction *createAndInitializeMatrix* prend en paramètres la taille de la grille de jeu pour créer un tableau en deux dimensions à l'aide de *malloc* par ligne puis par colonne sur des boucles *for*.

La fonction *checkIfSpaceIsEmpty* prend simplement en entrée sur quel tableau vérifier et quelle ligne et colonne vérifier pour vérifier si l'emplacement est vide ou s'il y a déjà un char sur celui-ci avec une variable *Boolean* qui renvoie *FALSE* ou *TRUE.*

La fonction *initializeTile* :

Les arguments entrés sont: la tuile à initialiser, le niveau de difficulté, le mode de jeu, et si c'est c'est au joueur 1 de commencer.

La tuile est d'abord mise entièrement vide, puis place 1 ou 2 lettres selon la difficulté, puis place les nombres selon la difficulté de jeu à l'aide de deux boucle *for* séparé.

Voici son algorithme :

Initialiser le Tableau `tile` :

- Boucler à travers chaque cellule dans le tableau `tile` 3x3 et définir sa valeur à '3', indiquant que la cellule est vide.

Déterminer le Nombre de Chiffres à Placer :

- Selon le niveau de difficulté :
    - Si `isHardDifficulty` est `true`, définir `howManyNumbers` à une valeur aléatoire entre 2 et 4.
    - Si `isHardDifficulty` est `false`, définir `howManyNumbers` à une valeur aléatoire entre 1 et 3.

Placer les Lettres :

- Déterminer le nombre de lettres à placer :

    - Si `isHardDifficulty` est `true`, placer 2 lettres.
    - Si `isHardDifficulty` est `false`, placer 1 lettre.
- Pour chaque lettre à placer :
    - Générer des coordonnées aléatoires `(x, y)` dans les limites du tableau 3x3.
    - Vérifier si la cellule à `(x, y)` est vide (`'3'`).
    - Si la cellule est vide :
        - Si `isMultiplayer` est `true` :
            - Si `isPlayer1` est `true`, placer une lettre commençant par 'A' et incrémenter pour chaque lettre suivante.
            - Si `isPlayer1` est `false`, placer une lettre commençant par 'X' et incrémenter pour chaque lettre suivante.
        - Si `isMultiplayer` est `false`, placer une lettre aléatoire de 'A' à 'Z'.
    - Répéter jusqu'à ce qu'une lettre soit placée avec succès.

Placer les Chiffres :

- Pour chaque chiffre à placer (`howManyNumbers` fois) :
    - Générer des coordonnées aléatoires `(x, y)` dans les limites du tableau 3x3.
    - Vérifier si la cellule à `(x, y)` est vide (`'3'`).
    - Si la cellule est vide :
        - Si `isPlayer1` est `true`, placer un chiffre aléatoire de '4' à '6'.
        - Si `isPlayer1` est `false`, placer un chiffre aléatoire de '0' à '2'.
    - Répéter jusqu'à ce qu'un chiffre soit placé avec succès.

La fonction *printHand* permet l'affichage des tuiles actuelles du joueur appelé avec la fonction, et la fonction *printLevel* avec *printTurn* affiche la grille au moment de la partie où la fonction est appelée, mais aussi le joueur à qui c'est le tours et son score.

Pour vérifier s'il est possible pour le joueur de jouer, il y deux fonctions utilisées pour faire la principale.

Algorithme pour `locateTileAnchor`

Description

La fonction `locateTileAnchor` parcourt une tuile à la recherche de la première lettre majuscule ('A' à 'Z'). Une fois trouvée, elle stocke les coordonnées de cette lettre dans les variables `anchorX` et `anchorY`.

Étapes de l'Algorithme

1. **Initialiser les Variables** :
    - Initialiser une variable `found` à `FALSE` pour suivre si une lettre a été trouvée.
    - Initialiser une variable `y` à 0 pour commencer la recherche depuis la première ligne.
2. **Boucler à Travers les Cellules du Tableau** :
    - Tant que `found` est `FALSE` et que `y` est inférieur à 3 :
        - Initialiser une variable `x` à 0 pour commencer la recherche depuis la première colonne de la ligne courante.
        - Tant que `found` est `FALSE` et que `x` est inférieur à 3 :
            - Si la cellule `tile[x][y]` contient une lettre majuscule ('A' à 'Z') :
                - Affecter les coordonnées `x` à `anchorX` et `y` à `anchorY`.
                - Mettre `found` à `TRUE`.
            - Incrémenter `x`.
        - Incrémenter `y`.

La fonction suivante était assez compliquée à mettre en place car les conditions pour placer une tuile sont très nombreuses.

Algorithme pour `isTilePlaceable`

Description

La fonction `isTilePlaceable` vérifie si une tuile (`tile`) peut être placée sur la grille de jeu en s'assurant qu'il n'y ait pas de collision avec d'autres lettres déjà placées et que les lettres ne soient pas placées en dehors des limites du niveau.

Paramètres

- `level`: tableau représentant le niveau.
- `levelX`: largeur du niveau.
- `levelY`: hauteur du niveau.
- `tile`: tableau représentant la tuile.
- `anchorLevelX`: coordonnée X de l'ancre dans le niveau.
- `anchorLevelY`: coordonnée Y de l'ancre dans le niveau.
- `anchorTileX`: coordonnée X de l'ancre dans la tuile.

- `anchorTileY`: coordonnée Y de l'ancre dans la tuile.

Étapes de l'Algorithme

1. **Calculer les Coordonnées Absolues** :
    - Calculer `absoluteX` comme étant `anchorLevelX` moins `anchorTileX`.
    - Calculer `absoluteY` comme étant `anchorLevelY` moins `anchorTileY`.
2. **Boucler à Travers les Cellules de la Tuile** :
    - Pour chaque cellule `(x, y)` dans la tuile (3x3) :
        - **Vérifier les Limites du Niveau** :
            - Si la cellule `tile[x][y]` contient une lettre majuscule ('A' à 'Z') et que les coordonnées absolues `(absoluteX + x, absoluteY + y)` sont en dehors des limites du niveau (`levelX` ou `levelY`), retourner `FALSE`.
        - **Vérifier les Collisions dans le Niveau** :
            - Si les coordonnées absolues `(absoluteX + x, absoluteY + y)` sont à l'intérieur des limites du niveau et que la cellule `tile[x][y]` n'est pas vide ('3') :
                - Si la cellule correspondante dans le niveau `level[absoluteX + x][absoluteY + y]` contient déjà une lettre majuscule ('A' à 'Z'), retourner `FALSE`.
                - Si la cellule `tile[x][y]` contient une lettre majuscule ('A' à 'Z') et que la cellule correspondante dans le niveau est vide ('3'), retourner `FALSE`.
3. **Retourner le Résultat** :
    - Si aucune des conditions ci-dessus n'a été violée, retourner `TRUE`.

La fonction principale n'a plus qu'à vérifier que chaque tuile est placable ou non sur la grille de jeu.

Algorithme pour `canPlayerPlay`

Description

La fonction `canPlayerPlay` vérifie si un joueur peut jouer au moins une des tuiles dans sa main (`hand`) sur le niveau (`level`). Pour chaque tuile dans la main, elle utilise la fonction `isTilePlaceable` pour déterminer si la tuile peut être placée à une position quelconque sur le niveau.

Paramètres

- `level` : tableau représentant le niveau.
- `levelX` : largeur du niveau.
- `levelY` : hauteur du niveau.
- `hand` : tableau de pointeurs représentant les tuiles dans la main du joueur (chaque tuile est un tableau 3x3).

Étapes de l'Algorithme

1. **Parcourir Chaque Tuile dans la Main** :
    - Pour chaque tuile `i` dans la main (de 0 à 4) :
        - Initialiser les variables `anchorTileX` et `anchorTileY` pour les coordonnées de l'ancre de la tuile.
        - Appeler la fonction `locateTileAnchor` pour obtenir les coordonnées de l'ancre de la tuile courante et les stocker dans `anchorTileX` et `anchorTileY`.
2. **Vérifier Toutes les Positions sur le Niveau** :
    - Pour chaque position `(x, y)` dans le niveau :
        - Appeler la fonction `isTilePlaceable` pour vérifier si la tuile courante peut être placée à la position `(x, y)` avec les coordonnées de l'ancre obtenues.
        - Si la tuile peut être placée (`isTilePlaceable` retourne `TRUE`), retourner `TRUE`.
3. **Retourner le Résultat** :
    - Si aucune tuile ne peut être placée à aucune position, retourner `FALSE`

La fonction suivante place une tuile spécifié dans la fonction :

Algorithme pour `placeTile`

Description

La fonction `placeTile` place une tuile sur le niveau à la position spécifiée par les coordonnées d'ancrage de la tuile et du niveau.

Paramètres

- `level` : tableau représentant le niveau.
- `levelX` : largeur du niveau.
- `levelY` : hauteur du niveau.
- `tile` : tableau 3x3 représentant la tuile.
- `anchorLevelX` : position x de l'ancre sur le niveau.

- `anchorLevelY` : position y de l'ancre sur le niveau.
- `anchorTileX` : position x de l'ancre sur la tuile.
- `anchorTileY` : position y de l'ancre sur la tuile.

Étapes de l'Algorithme

1. **Calculer les Coordonnées Absolues** :
    - Calculer `absoluteX` comme la différence entre `anchorLevelX` et `anchorTileX`.
    - Calculer `absoluteY` comme la différence entre `anchorLevelY` et `anchorTileY`.
2. **Parcourir Chaque Cellule de la Tuile** :
    - Pour chaque cellule `(x, y)` dans la tuile (de 0 à 2) :
        - Vérifier si la position absolue `(absoluteX + x, absoluteY + y)` est dans les limites du niveau (`levelX`, `levelY`) et si la cellule de la tuile n'est pas vide (`'3'`).
3. **Placer la Tuile** :
    - Si les conditions sont remplies, copier la valeur de la cellule de la tuile dans la position correspondante du niveau.

La dernière fonction rassemble toutes les fonctions décrites plus haut et permet aussi au joueur de décider quelle action il veut faire à son tour.

Algorithme pour `playerTurn`

Description

La fonction `playerTurn` gère le tour d'un joueur dans le jeu, lui permettant de placer une tuile, d'abandonner ou de sauvegarder la partie. Le score du joueur est mis à jour en fonction de ses actions.

Paramètres

- `level` : tableau représentant le niveau.
- `sizeX` : largeur du niveau.
- `sizeY` : hauteur du niveau.
- `hand` : tableau de pointeurs représentant les tuiles dans la main du joueur.
- `score` : score actuel du joueur.
- `isFirstTurn` : booléen indiquant si c'est le premier tour.
- `isHardMode` : booléen indiquant si le mode difficile est activé.
- `isMultiplayer` : booléen indiquant si le jeu est en mode multijoueur.
- `isPlayer1` : booléen indiquant si c'est le joueur 1.

Étapes de l'Algorithme

1. **Initialiser les Variables** :
    - `endTurn` à `FALSE` pour indiquer si le tour est terminé.
    - `proceed` et `cantPlace` à `FALSE`.
    - `anchorTileX`, `anchorTileY`, `placementX`, `placementY`, `tileIndex` pour stocker les coordonnées et l'index.
    - `input` pour stocker les entrées utilisateur.
2. **Boucle Principale du Tour** :
    - Tant que `endTurn` est `FALSE` :
        - Afficher le menu des options : "1 - Placer une tuile", "2 - Abandonner", "3 - Sauvegarder".
        - Lire l'entrée utilisateur.
3. **Gérer l'Option Sélectionnée** :
    - **Option 1 : Placer une tuile** :
        - Initialiser `tileIndex` à -1.
        - Demander au joueur de sélectionner une tuile (1-5) jusqu'à ce qu'une entrée valide soit donnée.
        - Appeler `locateTileAnchor` pour obtenir les coordonnées de l'ancre de la tuile sélectionnée.
        - Demander au joueur la colonne et la ligne où il souhaite placer la tuile.
        - Vérifier si la tuile peut être placée à l'emplacement donné :
            - Si oui, appeler `placeTile` pour placer la tuile, incrémenter le score et initialiser une nouvelle tuile.
            - Si non, afficher un message d'erreur.
    - **Option 2 : Abandonner** :
        - Demander confirmation au joueur (y/n).
        - Si le joueur confirme, multiplier le score par -1 et terminer le tour.
    - **Option 3 : Sauvegarder** :
        - Demander confirmation au joueur (y/n).
        - Si le joueur confirme, ajouter 1000 au score et terminer le tour.
4. **Fin du Tour** :
    - Afficher "End of turn".
    - Retourner le score mis à jour.

Enfin, il y a deux fonctions qui permettent de free les différentes allocations sur la grille de jeux et sur les tuiles : *freeMatrix* et *free3dMatrix*

Fichier Singleplayergame et Multiplayergame :

La différence entre les deux est faible, et la version deux joueurs double les actions et les variables, et les scores sont comparés à la fin pour désigner un vainqueur.

Bilan:

La majorité du projet est fonctionnelle, ainsi il est possible de jouer en solo et multijoueur et de sauvegarder dans un fichier texte.

La partie la plus difficile du programme a été la vérification et le placement de tuiles dans le plateau en raison des nombreuses conditions.

Aussi le débogage a été très long et aurait rendu le projet impossible à terminer dans les temps sans l'utilisation de copilot pour trouver les problèmes.

Ce qui ne marche pas:

L'ouverture de sauvegardes cause le crash du programme.

Pistes d'améliorations:

Avoir une ouverture fonctionnelle des sauvegardes

Sauvegarder les ancres des tuiles dans la structure principale afin de ne pas appeler la fonction findTileAnchors() 5 fois par tours, ce qui optimiserait le code