Projet Informatique - La chasse au trésor

Justifications techniques

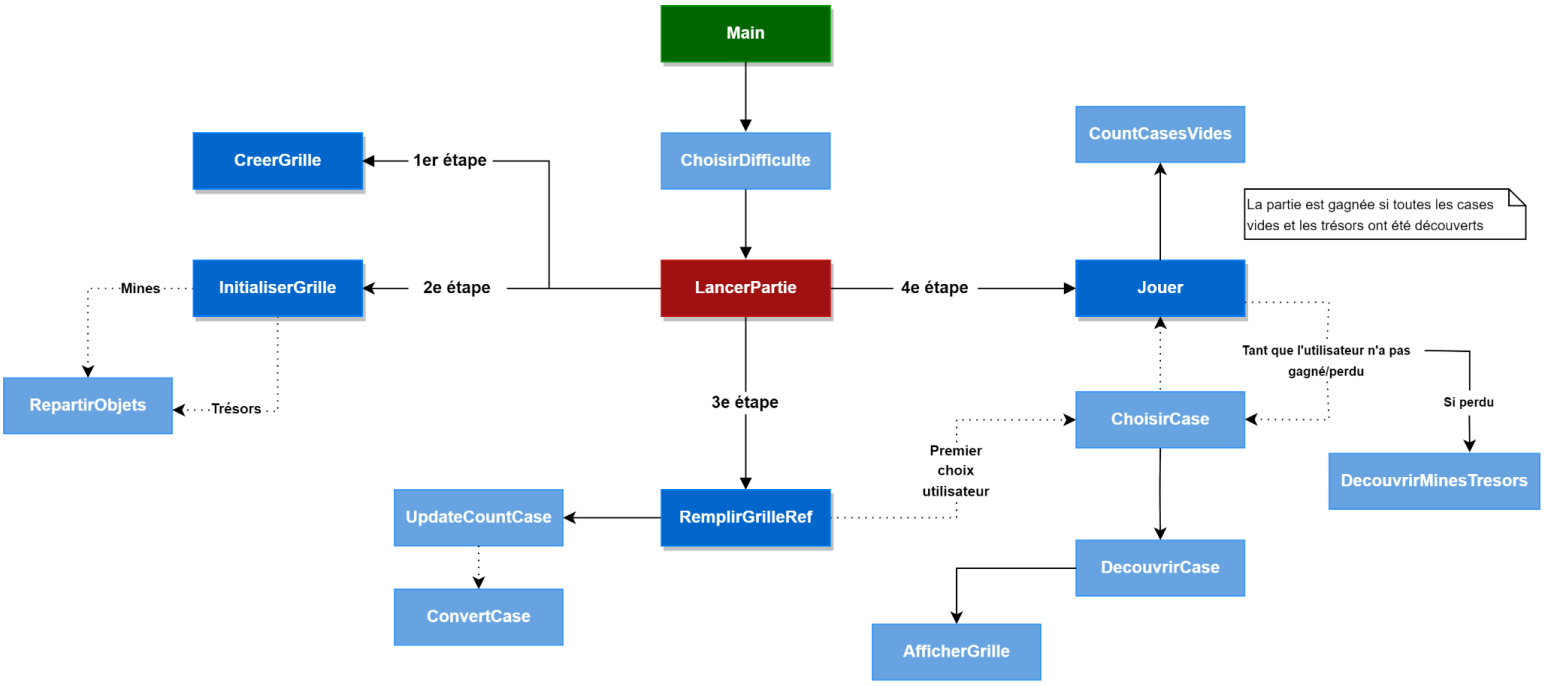
Biret Margo
Grenet Rebecca

17/12/2021

Sommaire :

I. Fonctionnement général du code

Afin de comprendre le fonctionnement général du code, nous avons souhaité créer un schéma reprenant les grandes fonctionnalités du code et le lien entre elles.



Pour que l'utilisateur puisse commencer une partie, nous appelons depuis la fonction principale, à savoir le Main, la procédure ChoisirDifficulte qui permettra par la suite de savoir si le jeu sera Facile, Intermédiaire, ou Difficile. Par la suite, la fonction LancerPartie est appelée. Celle-ci va alors successivement appeler des fonctions pour réaliser le jeu.

En premier lieu, cette fonction va appeler CreerGrille afin de créer la grille de référence qui ne sera pas disponible pour l'utilisateur.

Après avoir élaboré cette grille, la deuxième étape consiste à appeler la fonction InitialiserGrille. Celle-ci permettra de placer les mines et les trésors, à travers la procédure RepartirObjet qui sera appelée deux fois au sein de cette fonction. Après avoir terminé l'initialisation de la grille de référence, nous ferons de même pour la grille de Jeu, qui sera disponible pour l'utilisateur.

Lors de la 3ème étape, la fonction LancerPartie appellera RemplirGrilleRef. Elle demandera à l'utilisateur de choisir sa première case à travers ChoisirCase. Puis, cette procédure utilisera la grille de référence créée au préalable pour y ajouter les numéros pour

chacune des cases. Pour ce faire, les fonctions UpdateCountCase et ConvertCase seront appelées.

La dernière étape est d'appeler la fonction permettant à l'utilisateur d'interagir avec notre programme, c'est la fonction Jouer. Celle-ci se répètera tant que l'utilisateur n'est pas tombé sur une mine, ou si celui-ci n'a pas encore trouvé tous les trésors ainsi que les cases dites "vides". En effet, une partie est considérée comme gagnée si l'utilisateur trouve tous les trésors de la partie, ainsi que les cases vides. Pour ce faire, nous avons créé la fonction CountCasesVides qui nous permet d'obtenir le nombre de cases vides présentes dans la grille de référence (et donc la grille de jeu). La fonction ChoisirCase sera appelée, et une fois que l'utilisateur inscrit le numéro de ligne et de colonne souhaités, les cases de sa grille de Jeu seront découvertes à travers la procédure DecouvrirCase. Après chaque itération de choix de la case, l'utilisateur pourra voir sa grille de Jeu, à l'aide de la procédure AfficherGrille. Si l'utilisateur tombe sur une mine pendant le jeu, alors la partie sera terminée et la procédure DecouvrirMinesTresors sera appelée pour afficher toutes les mines et les trésors présents dans la grille.

Nous avons fait le choix de séparer le plus possible nos fonctions suivant leurs objectifs. Cela permet de respecter le S des principes de l'acronyme SOLID qui correspond à la responsabilité unique (*Single Responsibility principle*). En effet, toute fonction doit avoir une seule responsabilité. Notre code est donc plus lisible, mais il sera davantage plus simple à faire évoluer, contrairement à un code qui ne respecte pas ce principe.

II. Spécifications techniques

Tout d'abord, nous avons créé deux constantes correspondant aux valeurs des mines et trésors. En effet, pour des questions de clarté et de compréhension du code nous avons associé 1 à MINE et 2 à TRESOR pour ne pas avoir des chiffres sans signification évidente dans le programme. Ainsi, pour l'évolution et la relecture du code il sera plus simple de comprendre MINE et TRESOR plutôt que 1 et 2.

Ensuite, nous avons ajouté des contraintes lorsque le joueur doit entrer les différentes valeurs nécessaires au jeu.

Pour la taille de la grille, il faut respecter que les valeurs entrées ne soient pas négatives. Nous avons également exclu la possibilité qu'il n'y ait qu'une seule colonne ou qu'une seule ligne car le concept du jeu serait annulé. Effectivement, lors de la découverte d'une case dans ces cas là, nous pouvons directement savoir si les cases adjacentes sont des trésors ou des mines.

Pour les coups du joueur, il faut faire attention que celui-ci entre bien une valeur comprise dans la taille de la grille (supérieure à 1 et inférieure au nombre de lignes ou colonnes).

Afin que le programme ne s'arrête pas si un caractère autre qu'un nombre est entré par mégarde, nous avons aussi créé une boucle qui demande de saisir un nombre tant que la valeur entrée n'en est pas un, à la fois pour la taille de la grille et les coups du joueur. Dans cette même logique, lorsque l'utilisateur entre le niveau de difficulté, il est nécessaire de vérifier que ce soit bien de type entier, et que le numéro soit compris entre 1 et 3 inclus.

De plus, l'indice des tableaux en programmation commence à 0. C'est pourquoi nous avons fait le choix de retirer à la ligne ainsi qu'à la colonne une incrémentation pour correspondre aux index du tableau. Par exemple, si un utilisateur entre le chiffre 3 pour la ligne, celle-ci sera égale à 2, mais correspondra bien à la troisième valeur du tableau.

Finalement, pour aider le joueur à se repérer sur la grille de jeu, nous avons affiché le numéros des lignes et colonnes. Pour l'aider également à distinguer les lignes des colonnes, nous avons ajouté C devant le numéro de colonnes et L devant le numéro de lignes. Voici un tableau reprenant les différentes variables principales utilisées tout au long du programme afin de créer ce jeu :

Nom	Type + Justification	Intérêt
grilleRef	string[,] : permet d'avoir à la	Matrice qui permet d'avoir une grille qui ne

	fois des mines et des trésors qui sont des caractères, mais aussi les valeurs des cases adjacentes qui sont des entiers et des cases vides.	change pas, sur laquelle sont positionnés les mines, trésors et valeurs des cases adjacentes.
grilleJeu	string[,] : comme grilleRef, contenant en plus ND pour les cases non découvertes qui est une chaîne de deux caractères.	Tableau que le joueur voit. Elle est mise à jour à chaque tour selon le choix du joueur.
score	int[] : tableau des nombre de coups pour chaque partie gagnée.	Tableau qui permet de retenir les scores au cours d'une même partie (à savoir dans la même console).
nbMines	int : nombre sans virgule et forcément positif.	Variable qui contient le nombre de mines à positionner pour chaque partie car c'est un nombre aléatoire.
nbTresors	int : nombre sans virgule et forcément positif.	Variable qui contient le nombre de Trésors répartis sur la grille de jeu. Elle permet de savoir si le joueur a trouvé tous les trésors ou non.
countCaseVide	int : nombre sans virgule et forcément positif. Nous l'avons passé en référence afin que sa valeur soit modifiée dans les fonctions qui l'utilisent.	Variable de référence qui compte le nombre de cases vides dévoilées à chaque tour de jeu.
nbCasesVides	int : nombre sans virgule et forcément positif.	Variable qui compte le nombre total de cases vides dans la grille de référence au début d'une partie.

choixCaseUser	int[] : contient deux nombres entiers, un pour les lignes et un pour les colonnes.	Tableau, qui contient la ligne et la colonne choisies par l'utilisateur pour jouer le prochain coup.
niveau	int : il n'y a que trois niveaux possibles (1, 2 et 3) qui sont entiers.	Variable qui contient le chiffre correspondant au niveau de difficulté, qui peut être égal à 1 (Facile), 2 (Intermédiaire), ou 3(Difficile).
MINE	const int : Constante de type entier : nous avons expliqué ce choix précédemment.	Constante qui correspond à la valeur d'une mine, à savoir 1.
TRESOR	const int : Constante de type entier : nous avons expliqué ce choix précédemment.	Constante qui correspond à la valeur d'un trésor, à savoir 2.

III. Les fonctions implémentées et fonctionnalités supplémentaires

Afin de réaliser ce jeu de chasse au trésor en mode console, nous avons créé différentes fonctions, que nous allons vous présenter ci-dessous :

La fonction ChoisirDifficulte :

Cette fonction demande à l'utilisateur le niveau de difficulté qu'il souhaite et le renvoie (difficulte).

La fonction InitialiserGrille :

Cette fonction prend en entrée la grille de référence (grilleRef) ainsi que le nombre de Mines et de Trésors à placer (nbMines, nbTresors).

On initialise la grille de jeu (grilleJeu) comportant les "ND" (signifiant "Non Découvert") à l'aide de la fonction CreerGrille.

La fonction demande ensuite au joueur son premier coup à l'aide de ChoisirCase. Ce résultat est enregistré dans un tableau de deux entiers nommé choixCaseUser.
Un J est inscrit dans la grille de référence à l'endroit choisi précédemment par l'utilisateur.
Après cela, les mines et les trésors sont répartis aléatoirement sur la grille avec RepartirObjets, en évitant la case du premier coup du joueur grâce au J.
Le J est ensuite enlevé de la grille de référence et la fonction retourne un tableau multidimensionnel qui correspond à la grille de jeu. Elle a également en paramètre de sortie un tableau de dimension 2 comprenant le choix de la case du premier coup du joueur (choixCaseUser).

La fonction CreerGrille :

Cette fonction prend en entrée le nombre de lignes et de colonnes (lignes, colonnes) que la future grille aura ainsi que les caractères qui sont mis dedans initialement (caseJeu). ("ND" pour la grille de jeu et " " pour la grille de référence).
Elle crée un tableau multidimensionnel (grille) dont le nombre de lignes et de colonnes sont données au début. Pour chaque case, la fonction inscrit le caractère entré en paramètre.
Enfin, elle renvoie la grille.

La procédure RepartirObjets :

Cette procédure prend en entrée la grille de références (grilleRef) ainsi que le symbole (objet) et le nombre d'éléments (nbObjets) de l'objet à placer.
Elle initialise deux variables contenant un nombre aléatoire entre 0 et le nombre de lignes pour rndLigne et entre 0 et le nombre de colonnes pour rndColonne. Après ça la fonction regarde si la case correspondante est vide dans la grille de référence. Si c'est le cas, elle pose l'objet à cet endroit, sinon elle redonne un numéro de ligne et de colonne aléatoire et recommence. Cette étape est répétée pour le nombre d'objets à placer.

La procédure LancerPartie :

Cette procédure prend en entrée un nombre entier de lignes et de colonnes qui sont recueillis au préalable auprès de l'utilisateur dans le Main, le niveau choisi par le joueur, le

nombre de cases vides découvertes par référence (ref countCasesVides) ainsi que le tableau des scores (score) et l'indice du score actuel dans ce tableau (indexScore).
A chaque début de partie cette fonction va créer la grille de référence (grilleRef) en appelant la fonction CreerGrille avec le bon nombre de lignes et de colonnes données par le joueur. Les mines et les trésors sont ensuite ajoutés en appelant deux fois RemplirGrille (une fois pour chaque objet). Leur nombre (nbMines, nbTresors) est choisi aléatoirement (le nombre maximum de mines étant différent selon le niveau). La fonction crée ensuite la grille de jeu (grilleJeu) avec InitialiserGrille. Suite à cela, le premier choix de case du joueur (coord) est récupéré. La fonction appelle alors RemplirGrilleRef pour remplir les cases de la grille de référence avec les bonnes valeurs selon le placement des mines et trésors. Enfin elle dévoile la première case jouée (dont les coordonnées ont été récupérées plus haut) à l'aide de DecouvrirCase et lance la suite de la partie avec Jouer.

La fonction ChoisirCase :

Cette fonction prend en entrée un nombre de lignes et un nombre de colonnes (ligneTable, colonneTable).
Elle demande au joueur quelle case sera son prochain coup et vérifie que la valeur soit acceptable dans le jeu. Sinon elle lui en demande une autre.
Avant de renvoyer le résultat, elle retire 1 aux deux valeurs car l'indexation des tableaux commence à 0. Enfin, elle retourne un tableau de dimension 2 contenant le choix du joueur (choix).
Dans cette fonction, nous laissons la possibilité à l'utilisateur d'inscrire le chiffre 0. Nous testons alors s'il fait cela, pour renvoyer un tableau de deux entiers égaux à -1, cela sera utile dans la fonction Jouer pour arrêter la partie.

La fonction CountCasesVides :

Elle prend en entrée la grille de référence (grilleRef). Le nombre de cases vides est initialement à 0 (nbCasesVides). La fonction regarde ensuite pour chaque case de la grille si elle est vide ou non. Si c'est le cas, on implémente nbCasesVides qu'elle retourne à la fin.

La fonction Jouer :

Cette fonction prend en entrée les grilles de référence et de jeu (grilleRef, grilleJeu), le nombre de Mines et de Trésors pour la partie (nbMines, nbTresors), le nombre de cases vides découvertes par référence (ref countCasesVides) ainsi que le tableau des scores (score) et l'indice du score actuel dans ce tableau (indexScore).

Elle affiche tout d'abord la grille de jeu et crée trois variables initialisées à 0 : countTresor pour compter le nombre de trésors découverts, countMine et nbCoups.

Elle enregistre alors le nombre de cases vides dans la grille de référence à l'aide de CountCasesVides.

Après avoir affiché une première fois la grille de jeu avec AfficherGrille, la fonction démarre le chronomètre à l'aide de StopWatch pour le temps de la partie qui est enregistré dans tempsPartie.

Tant que tous les trésors et les cases vides n'ont pas été découverts (nbTresors>countTresor), que le joueur n'est pas tombé sur une mine ou sur une case déjà découverte, et que le booléen stop est faux, la fonction appelle la fonction ChoisirCase, et lui demande son prochain coup que l'on enregistre dans le tableau d'entiers choixCaseUser.

Nous avons ajouté la contrainte suivante : si la case est déjà découverte, alors un message apparaîtra pour lui demander le nouveau coup qu'il souhaite effectuer. De plus, si le tableau d'entiers choixCaseUser est égale à [-1,-1] alors cela signifie que l'utilisateur souhaite arrêter la partie. Un message sera affiché sera tout d'abord affiché pour confirmer son choix de quitter la partie. S'il répond "oui", alors le booléen stop passe à vraie et, à l'aide d'un break, nous sortons de l'itération.

Une fois que la case choisie n'a pas encore été révélée, la fonction regarde à l'aide de la grille de référence (grilleRef) où tombe le joueur. Si c'est une mine, on augmente le countMine. Si c'est un trésor, on incrémente countTresor et on affiche que le joueur en a découvert un et la case est révélée à l'aide de DecouvrirCase. Sinon la case est juste révélée à l'aide de DecouvrirCase et countCasesVides est incrémenté du nombre de cases vides révélées.

La fonction implémente ensuite le nombre de coups effectués et affiche la grille de jeu.

Si elle sort du while il y a deux options :

- Soit countMine vaut 1 et le joueur a perdu. La fonction affiche alors la grille avec la mine découverte. Il indique par la suite la défaite, ainsi que le temps de partie.
- Soit tous les trésors ont été trouvés et toutes les cases vides révélées, la partie est remportée. Le joueur peut alors voir en combien de temps et de coups il a gagné. Son score (équivalent au nombre de coups sur la partie) est ensuite ajouté au tableau des scores si cette valeur n'existe pas encore. Après avoir trié le tableau par ordre décroissant (du moins bon au meilleur score), si le nombre de coups de la partie jouée est égale au meilleur score la fonction le montre au joueur ainsi que le classement des trois premiers meilleurs scores.

Dans tous les cas, la fonction propose au joueur de commencer une nouvelle partie. S'il accepte, on rappelle la fonction ChoisirDifficulte puis LancerPartie. La console sera alors vidée à l'aide de Console.Clear() pour plus de lisibilité pour l'utilisateur. Si l'utilisateur ne souhaite pas continuer, alors il sera écrit dans la console le message suivant : "Merci et à bientôt".

La procédure DecouvrirCase :

Cette procédure prend en paramètres d'entrée les grilles de jeu et de référence (grilleJeu, grilleRef) ainsi que les coordonnées du choix de la case du joueur (ligne, colonne) et le nombre de cases vides découvertes par référence (ref countCasesVides).
Si la grille de jeu comporte un "ND" sur cette case, on regarde ce que comporte la grille de référence.

Si elle est vide, la même case de la grille de jeu prend la même valeur et countCasesVides est incrémentée. La fonction vérifie ensuite la valeur de chaque case autour (même ligne et colonne de droite, même ligne et colonne de gauche, même colonne et ligne d'au-dessus, même colonne d'en-dessous) en rappelant DecouvrirCase de façon récursive. Elle sort de la récursivité lorsque la case rappelée est en dehors du tableau ou lorsque la case appelée n'est pas vide dans la grille de référence. Dans ce dernier cas, elle affiche simplement la valeur de la case dans la grille de référence avant de sortir.

La procédure RemplirGrilleRef :

Cette procédure prend en entrée la grille de référence (grilleRef). Pour chaque case, on regarde sa valeur. Si la case est une mine ou un trésor on appelle UpdateCountCase avec l'objet correspondant.

La procédure UpdateCountCases :

Cette procédure prend comme paramètres d'entrée la grille de référence (grilleRef), une ligne, une colonne et un nombre (ligne, colonne, nb). Pour chaque case autour, s'il elle existe dans la grille, et qu'elle n'est pas occupée par une mine ou un trésor, on récupère sa valeur à l'aide de ConvertCase et on lui ajoute le nombre (nb). Enfin on le convertit en caractère avant de le remettre dans la case correspondante.

Prenons l'exemple suivant, nous regardons la case placée avec le "X". La récursivité nous sera utile pour tester les cases adjacentes. Il est alors nécessaire de parcourir chacune des cases c'est pourquoi nous avons utilisé deux boucles for. La première concerne les lignes, il faut donc itérer sur 3 lignes, qui ont comme coordonnées par rapport à X, -1, 0 et 1. Il sera nécessaire de faire de même pour les colonnes.

(-1,-1)	(-1,0)	(-1,1)
(0, -1)	X	(0,1)
(1,-1)	(1,0)	(1,1)

Dans chaque cas, la fonction vérifie que le numéro de ligne ou de colonne ne sort pas du tableau avant d'effectuer le reste.

La fonction ConvertCase :

Elle prend en entrée une grille (grilleRef) ainsi qu'une ligne et une colonne (ligne, colonne). Une variable nombre est créée. Si la case est vide nombre vaut 0. Si elle comporte déjà un nombre on le convertit en entier et on l'associe à nombre.

La procédure DecouvrirMinesTresors :

Cette procédure prend comme paramètres d'entrée les grilles de jeu et de référence (grilleJeu et grilleRef).

Elle est appelée lorsque l'utilisateur tombe sur une mine au cours de la partie dans la fonction Jouer. Elle va parcourir la grille de référence, à savoir un tableau dimensionnel, afin de tester si des cases T (trésors) ou M (mines) sont présentes. Si la condition est vraie, alors nous affectons cette valeur (à savoir T ou M) sur la grille de jeu. Cela permettra à la fin de la partie d'afficher à l'utilisateur où se situent les mines et les trésors.

La fonction AfficherGrille :

Cette fonction prend en paramètre une grille.

Tout d'abord, une première ligne comprenant le numéro des colonnes sera créée ; cela permettra à l'utilisateur de se retrouver lors du choix de la case. Ensuite pour chaque ligne, nous affichons d'abord le numéro de la ligne puis les valeurs pour chacune des cases.

Entre chaque colonne est ajouté "|" afin de mieux distinguer les cases. De même, les lignes seront séparées à l'aide de tirets.

Nous avons également ajouté de la couleur dans la console afin d'obtenir un grille de jeu plus claire pour l'utilisateur. Lorsqu'il découvrira un trésor, la case sera alors colorée en jaune, contrairement à une mine qui sera colorée en rouge.

Nous avons également ajouté des fonctionnalités au jeu demandé :

- *Affichage du temps* à la fin de chaque partie. Pour cela nous créons une instance de la classe StopWatch importée à l'aide de System.Diagnostics. A chaque début de partie, le chronomètre est lancé.
- *Affichage du nombre de coups* lorsqu'une partie est gagnée. Pour cela nous avons initialisé une variable nbCoups qui s'incrémente à chaque tour. Elle se réinitialise à 0 à chaque début de partie et ne s'affiche que si le joueur gagne. Nous avons fait ce choix d'affichage pour ne pas décourager le joueur s'il perd.

- *Possibilité de rejouer*. Pour cela nous demandons si le joueur souhaite recommencer à chaque fin de partie. Si c'est le cas, nous rappelons la fonction LancerPartie, sinon le jeu s'arrête.
- *Possibilité d'arrêter la partie en cours*. Pour cela, nous avons dit à l'utilisateur que s'il entrait 0 lors du choix de la case, il pouvait quitter la partie. Si c'est le cas, nous lui demandons si c'est vraiment son souhait. Il sort alors de l'itération, et affiche que sa partie est terminée, avec le temps de partie et le nombre de trésors restants à trouver.
- *Création de niveaux*. Pour cela nous avons changé le nombre de mines selon le niveau sélectionné par le joueur au début de la partie. Pour le niveau 1 (le plus simple) le nombre de mines doit être inférieur à la taille de la grille divisé par quatre. Dans le cas où la grille à une taille de deux ou trois, on ne met qu'une seule mine car sinon il y en a 0 à cause de la division euclidienne. Pour le niveau 2, nous avons fait le même système avec le nombre de mines inférieur à la taille de la grille divisé par trois. Enfin, pour le niveau 3 nous avons gardé le nombre de mines maximum demandé dans l'énoncé. Dans tous les cas, nous avons choisi de garder le maximum au nombre de lignes divisé par deux.
- *Affichage des règles*. Pour cela, dans le main au tout début du jeu, nous avons demandé au joueur s'il souhaite voir les règles du jeu. Elles sont donc affichées uniquement s'il répond oui.
- *Affichage en jaune d'un trésor trouvé*. Lorsqu'un trésor est trouvé, celui-ci s'affiche en jaune sur la grille afin que le joueur le repère plus facilement.
- *Affichage en rouge d'une mine trouvée*. Lorsque le joueur tombe sur une mine, celle-ci s'affiche en rouge, accompagnée du message "La partie est terminée... Vous êtes tombé(e) sur une mine.".
- *Affichage en couleur de la grille*. Pour cela nous utilisons ConsoleColor pour la couleur de fond et la couleur d'écriture.
- *Sauvegarde du meilleur score tant que la console n'est pas fermée*. Pour cela nous avons mis en place un tableau des scores que nous implémentons à chaque partie, si le joueur gagne. Le nombre de coups est rentré uniquement si celui-ci n'est pas déjà présent dans le tableau. Puis, à chaque fois, nous le trions dans l'ordre

croissant. Si le nombre de coups de la partie actuelle est égale au meilleur score on affiche “Vous obtenez le meilleur score !” et on affiche le top 3 des résultats.
Cependant, nous avons initialisé un tableau de dimension 100 (On suppose que l’utilisateur ne fera pas plus de 100 parties dans la même console). Pour éviter d’obtenir un 0 (qui est la valeur par défaut d’un tableau d’entier), nous testons lors du parcours du tableau si le résultat est différent de 0.
Cependant, la complexité de cette étape est assez importante suite au parcours du tableau. Il serait préférable d’utiliser un tableau redimensionnable (à savoir un ArrayList).

IV. Difficultés rencontrées et pistes d’amélioration

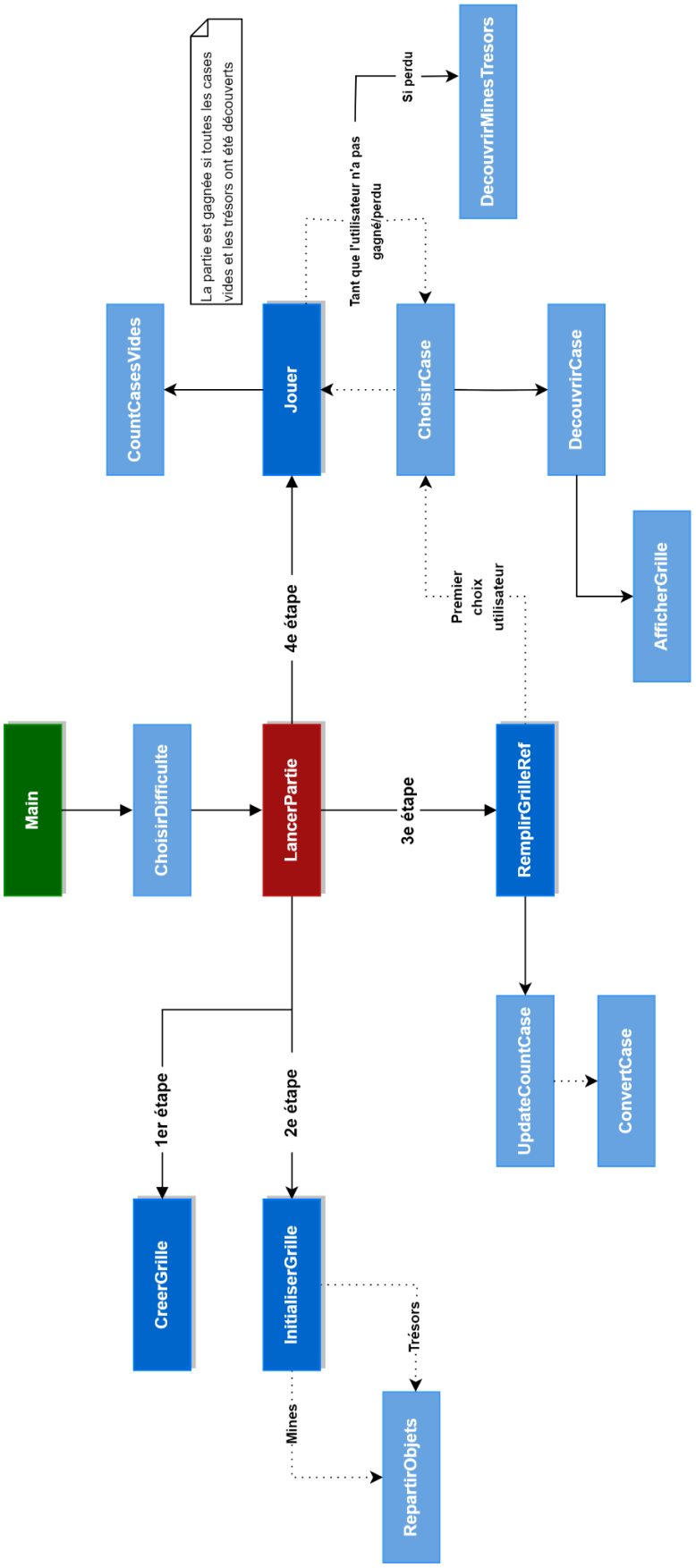
La programmation d’un projet en binôme est parfois assez complexe. En effet, notre environnement de développement Visual Studio ne permet pas de travailler à plusieurs. C’est pourquoi, afin de travailler sans être sur le même ordinateur, tout en ayant les mêmes implémentations, nous avons souhaité utiliser l’outil Git en créant un repository. Le logiciel git permet d’obtenir une version locale et en ligne. Nous avons créé deux branches, pour pouvoir implémenter notre code. Afin de regrouper notre travail, nous avons fait plusieurs “merge” de branches, ce qui permettait de fusionner nos fonctionnalités.

Pour la compréhension de ce que nous faisions séparément, il a été important de bien commenter notre code de façon simple. Chaque fonction était expliquée rapidement et nous avons utilisé des noms de variables explicites.

Une autre difficulté fut d’essayer d’avoir le moins de répétition de code possible devant toutes les possibilités qu’il y avait, notamment pour les fonctions UpdateCountCase et DecouvrirCase. Il nous a fallu du temps et de nombreuses erreurs pour réussir à faire la fonction récursive pour DecouvrirCase et prendre en compte toutes les sorties de tableaux dans UpdateCountCase.

V. Annexe

- **Annexe 1 : *schéma de relation des fonctions***



Captures d'écran de la console :

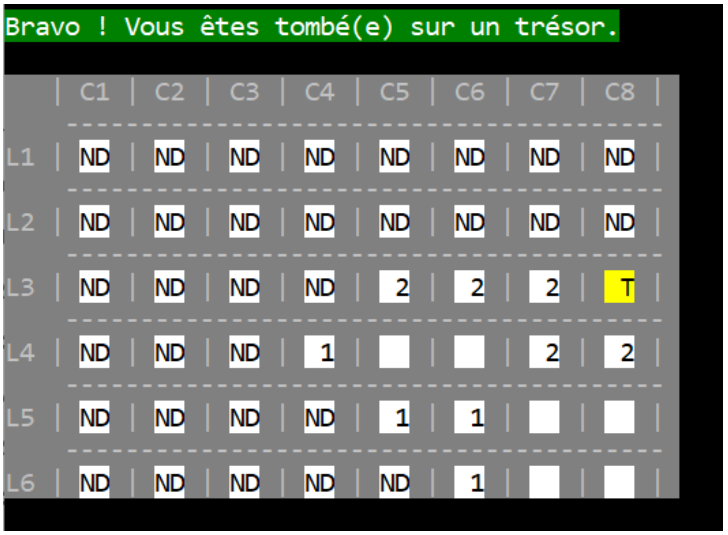
- **Annexe 2 : *Début de partie***

```
---------------------------- Bienvenue sur le jeu de la chasse au trésor ! --------------------

Souhaitez-vous afficher les règles du jeu ? (Oui/Non)
oui

********************************** Règles du jeu **********************************

Objectif : Trouver tous les trésors et les cases vides sur la carte sans tomber sur une mine !

A chaque tour, vous devez choisir une case à réveler pour afficher un décompte.
Ce décompte est calculé à partir du nombre de cases adjacentes minées ou ayant un trésor.
Une case trésor prend la valeur 2 tandis qu'une case mine prend la valeur 1.
Attention : il peut y avoir 1, 2 ou 3 trésors présents sur la carte.
Bon courage !

***********************************************************************************

Veuillez entrer le numéro du niveau souhaité :
- n°1: Facile
- n°2: Intermédiaire
- n°3: Difficile
2
```

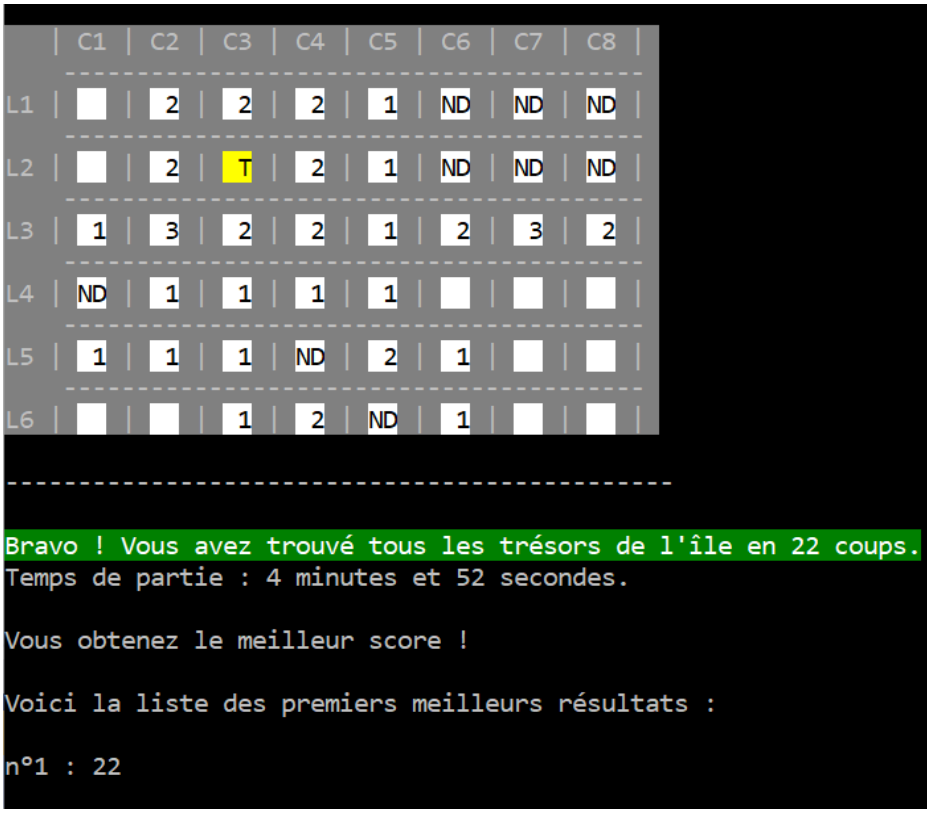
- **Annexe 3 : *Initialisation de la grille***

```
Entrer le nombre de lignes de la grille de jeu souhaitées :
6
Entrer le nombre de colonnes de la grille de jeu souhaitées :
5

   | C1 | C2 | C3 | C4 | C5 |
   ------------------------
L1 | ND | ND | ND | ND | ND |
   ------------------------
L2 | ND | ND | ND | ND | ND |
   ------------------------
L3 | ND | ND | ND | ND | ND |
   ------------------------
L4 | ND | ND | ND | ND | ND |
   ------------------------
L5 | ND | ND | ND | ND | ND |
   ------------------------
L6 | ND | ND | ND | ND | ND |

Entrer le numéro de la ligne souhaitée pour jouer :
2
Entrer le numéro de la colonne souhaitée pour jouer :
1

   | C1 | C2 | C3 | C4 | C5 |
   ------------------------
L1 | ND | ND | ND | ND | ND |
   ------------------------
L2 |  2 | ND | ND | ND | ND |
   ------------------------
L3 | ND | ND | ND | ND | ND |
   ------------------------
L4 | ND | ND | ND | ND | ND |
   ------------------------
L5 | ND | ND | ND | ND | ND |
   ------------------------
L6 | ND | ND | ND | ND | ND |

-----------------Début de la partie-----------------

Entrer le numéro de la ligne souhaitée pour jouer :
```

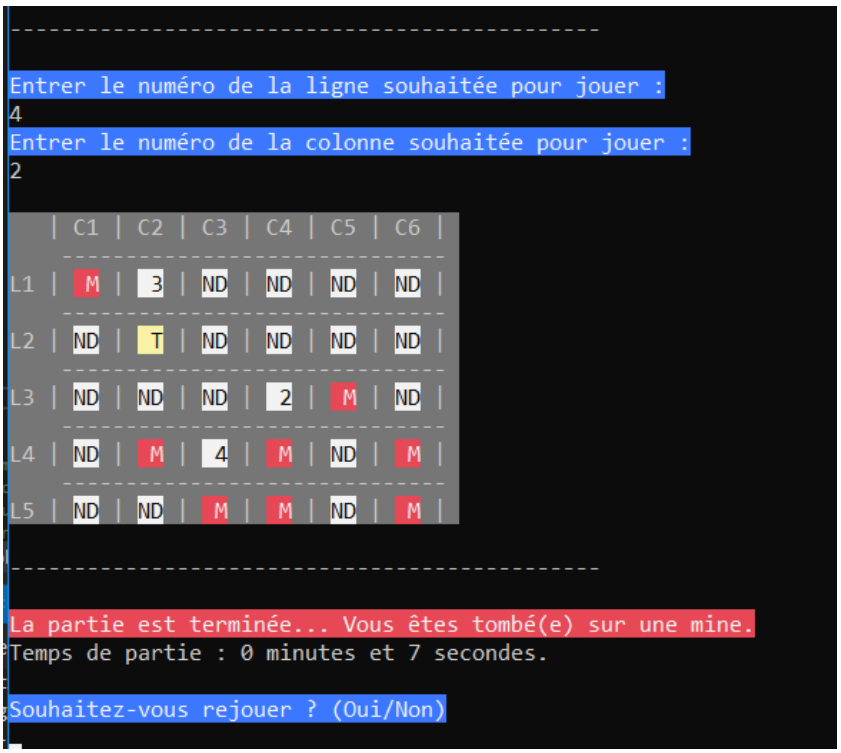
- **Annexe 4 : *Trésor trouvé***



- **Annexe 5 : *Partie gagnée***



- **Annexe 6 : *Partie perdue***



- **Annexe 7 : *Quitter la partie***