Olivier Fabre, uapv2104042

Projet – Programmation parallèle Résolution d'un labyrinthe par backtracking avec collecte d'objets

Je reste à votre dispsoition pour vous expliquer mon code et le choix de tel verrou, de telle valeur, ma vision de ma conception de ces 3 implémentations. J'ai biensur relis le cours et effectué des recherches pour trouver le meilleur modèle de conception d'une implémentation séquentielle. J'ai donc tout recommencé mes scripts pour mieux les comprendre.

Les labyrinthes

Nous avons un fichier de format texte avec 4 labyrinthes:

```
####################
#D    #        #   #
####  #  ####  # # #
#  #     #  #    # #
#M #######  ###### #
#                  #
###### # ####### # #
#    # #     #   # #
# ## # ### # # ### #
# #    #   # #     #
# ###### ### ##### #
#      E   #     # #
# #### ### ###### ##
#       #          #
##### # ### ## # # #
#   # #      # # # #
# # ### #### # ### #
# #     #    #     #
# ###### # #######1#
####################

####################
#     #    1   #   #
####  #  ####  # # #
#  #     #  #    # #
#  #######  ###### #
#                  #
###### # ####### # #
#    # #     #   # #
# ## # ### # # ### #
# #    #   # #B    #
# ###### ### ##### #
#          #       #
```

```
# #### ### ###### ##
#       #         T#
##### # ### #### ###
#   # #      # # # #
# # ### #### # ### #
# #     #    #     #
# ###### # #######2#
####################

####################
#     #    1   #   #
####  #  ####  # # #
#  #     #  #    # #
#  #######  ###### #
#                   #
###### # ####### # #
#    # #     #   # #
# ## # ### # # ### #
# #    #   # #B    #
# ###### ### ##### #
#          #       #
# #### ### ###### ##
#       #         T#
##### # ### ####   #
#   # #      # #   #
# # ### #### # ### #
# #     #    #     #
# ###### # #######2#
####################

####################
#     #  M     #   #
####  #  ####  # # #
#2 #     #  #    # #
#  #######  ###### #
#                   #
###### # ####### # #
#  M # #     #   # #
# ## # ### # # ### #
# #    #   # #     #
# ###### ### ##### #
#          #     # #
# #### ### ######  #
#       #         C#
##### # ### ## # # #
#   # #      # # # #
# # ### #### # ### #
# #     #    #     #
# ###### # #######A#
```

```
83 ####################
```

Les labyrinthes sont numérotés de 0 à 3, rangés en verticale pour les charger plus rapidement.

Dans un labyrinthe, on trouve:

- un point de départ nommé par une lettre (D, 1, T, 2)
- un point d'arrivé nommé par une lettre (1, T, 2, A)
- des murs de symbole #
- des objets à ramasser ou non (E, B, C)
- des mines (M) à éviter

les contraintes obligatoires pour résoudre le 1er labyrinthe (numero 0):

- le départ se fait du point **D** et se termine au point **1**
- Éviter les mines **M** si présentes
- Passer obligatoirement par le point **E** (ramassage objet)
- on ne peut pas passer à travers un mur **#**

les contraintes obligatoires pour résoudre le 2ème labyrinthe (numéro 1):

- le départ se fait du point **1** et se termine au point **T**
- Éviter les mines **M** si présentes
- Passer obligatoirement par le point **B** (ramassage objet)
- on ne peut pas passer à travers un mur **#**

les contraintes obligatoires pour résoudre le 3ème labyrinthe (numéro 2):

- le départ se fait du point **T** et se termine au point **2**
- Éviter les mines **M** si présentes
- il n y a aucun point de passage obligatoire (aucun ramassage objet)
- on ne peut pas passer à travers un mur **#**

les contraintes obligatoires pour résoudre le 4ème labyrinthe (numéro 3):

- le départ se fait du point **2** et se termine au point **A**
- Éviter les mines **M** si présentes
- Passer obligatoirement par le point **C** (ramassage objet)
- on ne peut pas passer à travers un mur **#**

Implémentation d'un backtracking séquentielle (v1)

Mon programme C++ 11 lit plusieurs labyrinthes décrits dans « labyrinthe.txt », chaque grille étant séparée par une ligne vide. Il transforme chaque caractère en une structure `Case` avec des coordonnées (sous forme de matrice lignes x colonnes) indiquant murs, mines, départ, arrivée et objets à récupérer, puis applique un parcours en profondeur récursif (backtracking) limité aux quatre directions cardinales pour chercher un chemin valide: la récursion s'arrête (les conditions d'arret) sur un mur, une mine, une case déjà visitée ou, au contraire, lorsqu'on atteint l'arrivée après avoir, si nécessaire, ramassé l'objet requis.

La fonction `resoudreLabyrinthe` enregistre le tracé au fil de la recherche, de sorte qu'en cas de succès le chemin est restitué en remplaçant les espaces par des `*`. L'affichage terminal est enrichi de codes ANSI pour distinguer murs, mines, points de départ/arrivée, objets et chemin trouvé en couleurs, ce qui facilite la lecture visuelle du résultat.

La boucle principale parcourt toutes les grilles, initialise dynamiquement les rôles des symboles (`D`, `1`, `T`, `2`, `A`) selon l'index du labyrinthe, exécute la résolution puis mesure le temps total d'exécution avec `std::chrono`.

Voici les résultats visuels:

```
Labyrinthe #1 (20x20)
  ####################
  #D****#********#***#
  #### *#* #### *#*#*#
  #  # *** #**# ***#*#
  #M #######**#######*#
  #       ********* *#
  ###### #*########*#*#
  #    # #***  #***#*#
  # ## # ###*# #*###*#
  # #    #***# #*****#
  # ######*### ##### #
  #     *E*  #     # #
  # ####*### ###### ##
  #     **#  ********#
  ##### #*###*## # #*#
  #   # #***** # # #*#
  # # ### #### # ###*#
  # #     #    #    *#
  # ###### # #######1#
  ####################

Labyrinthe #2 (20x20)
  ####################
  #     #    1***#***#
  ####  #  #### *#*#*#
  #  #     #**# ***#*#
  #  #######**#######*#
  #       ********* *#
```

```
  ###### #*#######*#*#
  #    # #*****#***#*#
  # ## # ### #*#*###*#
  # #    #   #*#B****#
  # ###### ###*##### #
  #          #****** #
  # #### ### ######*##
  #       #        *T#
  ##### # ### #### ###
  #   # #      # # # #
  # # ### #### # ### #
  # #     #    #     #
  # ###### # #######2#
  ####################

Labyrinthe #3 (20x20)
  ####################
  #     #    1   #   #
  ####  #  ####  # # #
  #  #     #  #    # #
  #  #######  ###### #
  #                  #
  ###### # ####### # #
  #    # #     #   # #
  # ## # ### # # ### #
  # #    #   # #B    #
  # ###### ### ##### #
  #          #       #
  # #### ### ###### ##
  #       #         T#
  ##### # ### ####  *#
  #   # #      # #  *#
  # # ### #### # ###*#
  # #     #    #    *#
  # ###### # #######2#
  ####################

Labyrinthe #4 (20x20)
  ####################
  #     #  M     #   #
  ####  #  ####  # # #
  #2*#     #**#    # #
  # *#######**###### #
  # *****************#
  ###### # ####### #*#
  #  M # #     #   #*#
  # ## # ### # # ###*#
  # #    #   # #    *#
  # ###### ### #####*#
```

```
  #          #     #*#
  # #### ### ###### *#
  #       #         C#
  ##### # ### ## # #*#
  #   # #      # # #*#
  # # ### #### # ###*#
  # #     #    #    *#
  # ###### # #######A#
  ####################

Temps total pour résoudre tous les labyrinthes : 206 ms
```

Implémentation d'un backtracking parallèle variante 1 (v2)

Cette variante 1 parallèle conserve exactement le parcours en profondeur récursif backtracking de la version séquentielle, mais distribue la résolution des différentes grilles sur plusieurs threads C++ 11 : le vecteur `threads` lance pour chaque labyrinthe une fonction `traiterLabyrinthe`.

`traiterLabyrinthe` convertit la grille en structure `Case`, exécute le backtracking, prépare un affichage coloré, puis écrit le résultat mais l'accès à `cout` l'affichage est protégé par un `mutex` global pour éviter la race condition des sorties d'ffiches des résultats.

Chaque labyrinthe est résolu indépendamment.

La mesure chrono englobe la création des threads, leur exécution et leur synchronisation par `join`, fournissant un indicateur clair du gain de performance obtenu grâce au parallélisme.

Voici les résultats visuels des grilles selon la vitesse de leurs threads respectifs donc dans le désordre:

```
Labyrinthe #3 (20x20):
  ####################
  #     #    1   #   #
  ####  #  ####  # # #
  #  #     #  #    # #
  #  #######  ###### #
  #                  #
  ###### # ####### # #
  #    # #     #   # #
  # ## # ### # # ### #
  # #    #   # #B    #
  # ###### ### ##### #
  #          #       #
  # #### ### ###### ##
  #       #         T#
  ##### # ### ####  *#
  #   # #      # #  *#
  # # ### #### # ###*#
  # #     #    #    *#
```

```
  # ###### # #######2#
  ####################

Labyrinthe #4 (20x20):
  ####################
  #     #  M     #   #
  ####  #  ####  # # #
  #2*#     #**#    # #
  # *########**###### #
  # *****************#
  ###### # ####### #*#
  #  M # #     #   #*#
  # ## # ### # # ###*#
  # #    #   # #    *#
  # ###### ### #####*#
  #          #     #*#
  # #### ### ###### *#
  #       #         C#
  ##### # ### ## # #*#
  #   # #      # # #*#
  # # ### #### # ###*#
  # #     #    #    *#
  # ###### # #######A#
  ####################

Labyrinthe #1 (20x20):
  ####################
  #D****#********#***#
  #### *#* #### *#*#*#
  #  # *** #**# ***#*#
  #M #######**#######*#
  #       ********* *#
  ###### #*########*#*#
  #    # #***  #***#*#
  # ## # ###*# #*###*#
  # #    #***# #*****#
  # ######*### ##### #
  #     *E*  #     # #
  # ####*### ###### ##
  #     **#  *********#
  ##### #*###*## # #*#
  #   # #***** # # #*#
  # # ### #### # ###*#
  # #     #    #    *#
  # ###### # #######1#
  ####################

Labyrinthe #2 (20x20):
  ####################
```

```
#     #    1***#***#
####  #  #### *#*#*#
#  #     #**# ***#*#
#  #######**#######*#
#       ********* *#
###### #*########*#*#
#    # #*****#***#*#
# ## # ### #*#*###*#
# #    #   #*#B****#
# ###### ###*##### #
#          #****** #
# #### ### ######*##
#       #        *T#
##### # ### #### ###
#   # #      # # # #
# # ### #### # ### #
# #     #    #     #
# ###### # #######2#
####################

Temps total : 66 ms
```

Implémentation d'un backtracking parallèle variante 2 (v3)

Cette seconde variante introduit un parallélisme à l'intérieur du labyrinthe: pour chaque grille, le programme lance quatre threads (par une méthode lambda que nous avons appris dans notre cours que j'ai appelé « sniffers ») qui explorent simultanément les quatre premières directions possibles à partir de la case de départ.

Un `atomic<bool> solutionTrouve` permet d'indiquer à tous les threads qu'un chemin valide a déjà été découvert. je peux la modifier avec la methode `store`

Le vecteur `solutionChemin` est partagé et protégé par un `mutex` pour n'être rempli qu'une seule fois.

chaque thread travaille sur sa propre copie du tableau `isCaseVisitee` grâce à la méthod elambda qui permet de faire cela plus facilement, ce qui évite la synchronisation pendant le parcours en profondeur mais cela consomme plus de mémoire.

Aussitôt qu'un thread trouve l'arrivée (en respectant éventuellement la contrainte d'objet à ramasser), il verrouille le mutex, enregistre le chemin et déclenche un flag booléen atomique. Les autres threads terminent leurs tâches sans perturber la solution qui a été déjà trouvée.

Les labyrinthes sont toujours traités l'un après l'autre: le gain de temps vise donc surtout les labyrinthes individuels complexes, et non l'ensemble comme dans la variante « un thread par labyrinthe ».

Le chemin retenu est ensuite colorisé et imprimé comme auparavant, puis le chrono global mesure le temps total, permettant de comparer facilement l'accélération obtenue par ce parallélisme plus précis et ccomplexe.

Voici le résultat visuel:

```
Labyrinthe #1 (20x20)
  ####################
  #D****#********#****#
  #### *#* #### *#*#*#
  #  # *** #**# ***#*#
  #M #######**######*#
  #       ********* *#
  ###### #*########*#*#
  #    # #***  #***#*#
  # ## # ###*# #*###*#
  # #    #***# #*****#
  # ######*### ##### #
  #     *E*  #     # #
  # ####*### ###### ##
  #     **#  *********#
  ##### #*###*## # #*#
  #   # #***** # # #*#
  # # ### #### # ###*#
  # #     #    #    *#
  # ###### # #######1#
  ####################

Labyrinthe #2 (20x20)
  ####################
  #     #    1***#***#
  ####  #  #### *#*#*#
  #  #     #**# ***#*#
  #  #######**######*#
  #       ********* *#
  ###### #*########*#*#
  #    # #*****#***#*#
  # ## # ### #*#*###*#
  # #    #   #*#B****#
  # ###### ###*##### #
  #          #****** #
  # #### ### ######*##
  #       #        *T#
  ##### # ### #### ###
  #   # #      # # # #
  # # ### #### # ### #
  # #     #    #     #
  # ###### # #######2#
  ####################

Labyrinthe #3 (20x20)
  ####################
  #     #    1   #   #
```

```
  ####  #  ####  # # #
  #  #     #  #    # #
  #  #######  ###### #
  #                  #
  ###### # ####### # #
  #    # #     #   # #
  # ## # ### # # ### #
  # #    #   # #B    #
  # ###### ### ##### #
  #          #       #
  # #### ### ###### ##
  #       #         T#
  ##### # ### ####  *#
  #   # #      # #  *#
  # # ### #### # ###*#
  # #     #    #    *#
  # ###### # #######2#
  ####################

Labyrinthe #4 (20x20)
  ####################
  #     #  M     #   #
  ####  #  ####  # # #
  #2 #     #**#    # #
  #**########**###### #
  # *****************#
  ###### # ####### #*#
  #  M # #     #   #*#
  # ## # ### # # ###*#
  # #    #   # #    *#
  # ###### ### #####*#
  #          #     #*#
  # #### ### ###### *#
  #       #         C#
  ##### # ### ## # #*#
  #   # #      # # #*#
  # # ### #### # ###*#
  # #     #    #    *#
  # ###### # #######A#
  ####################

Temps total pour résoudre tous les labyrinthes : 103 ms
```

Vous pouvez retrouver mon dépôt des codes poru l'implémentation V1, V2, V3 à l'adresse suivante: https://github.com/olfabre/LabyrinTrack/tree/main