TP 13bis – Classes et objets

Dans ce TP, vous allez créer vos premières classes dans le contexte d'un programme d'affichage graphique. Vous apprendrez également à organiser vos fichiers dans un cadre de programmation orientée objet.

Les instructions notées « **facultatives** » sont à faire à la fin du TP, une fois que celles qui sont obligatoires ont été réalisées avec succès !

L'objectif de ce TP est d'implémenter un moteur de résolution de labyrinthe

Contexte

Nous souhaitons réaliser un logiciel de résolution de labyrinthe 2D. Le labyrinthe sera composé d'un plateau en deux dimensions constitué de cases. Il sera alors possible de passer d'une case à l'une de ses voisines seulement si aucun mur n'est présent entre ces deux cases. Le logiciel devra implémenter un algorithme de résolution qui parcourra le labyrinthe jusqu'à trouver la sortie. Au cours du parcours, les cases du labyrinthe parcourues s'afficheront en jaune. Lorsque l'algorithme devra rebrousser chemin, les cases parcourues dans la mauvaise direction s'afficheront en rouge. La figure 1 représente différentes étapes successives de la résolution d'un labyrinthe.

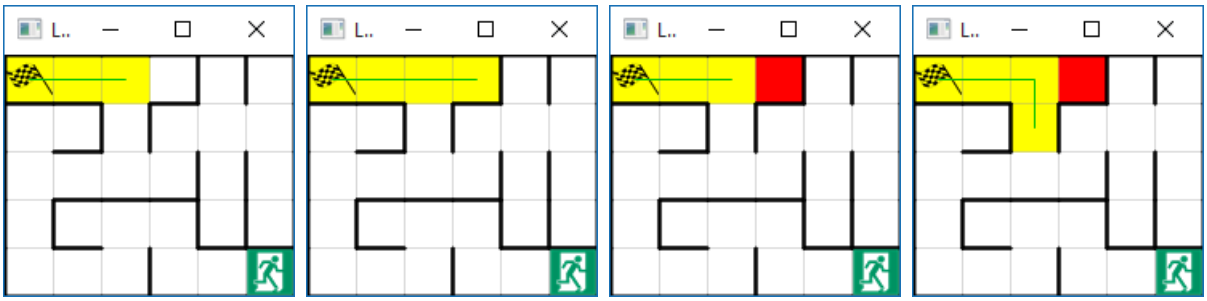


Figure 1 : Capture d'écran de l'avancement dans le labyrinthe

Nous voulons que le programme ait les fonctionnalités suivantes :

- Lorsque l'utilisateur appuie sur la barre d'espace, un pas de parcours est effectué et la représentation graphique est mise à jour.
- Le chemin de résolution du labyrinthe doit s'afficher par une ligne verte reliant toutes les cases utiles entre le point de départ et la position courante de l'algorithme (comme sur la figure 1).

Exercice 0 Mise en place du TP

Récupérez le fichier **`VEtudiant.zip`** sur l'ENT. Il contient une solution Visual Studio contenant un squelette de programme principal que vous pouvez utiliser pour démarrer.

Exercice 1 Votre première classe et vos premiers objets

Nous allons implémenter dans cet exercice la première version de la classe **CCase** de gestion d'une case du labyrinthe.

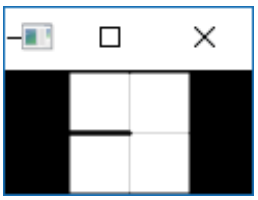
1.1 Gestion d'une case

Dans le contexte exprimé en préambule, nous souhaitons implémenter la classe de gestion d'une case du labyrinthe.

- Écrivez la déclaration de cette classe dans le fichier nommé **Case.h**. Elle aura pour membres :
  - Données :
    - Sa position sur le plateau (2 entiers)
    - Une donnée pour savoir si elle a été visitée (case jaune : 1 booléen)
    - Une donnée pour savoir si elle correspond à un mauvais parcours (case rouge : 1 booléen)
    - Quelles sont ses éventuelles 4 voisines (4 pointeurs vers des cases pouvant être `nullptr` s'il n'y a pas de connexion avec la voisine dans la direction considérée)
  - Comportements :
    - Des méthodes pour définir ses cases voisines
    - Une méthode pour définir qu'elle a été visitée
    - Une méthode pour définir qu'elle se situe sur un mauvais chemin
    - Des accesseurs en lecture aux différentes données internes
- Écrivez la définition de cette classe dans le fichier nommé **Case.cpp**
  - Les méthodes de définition des cases voisines doivent définir le voisinage dans les deux sens. Par exemple la méthode `setRightCase(CCase* pCase)` doit :
    - Vérifier si la case de droite est déjà définie. Si c'est le cas, la méthode ne doit rien faire.
    - Définir la case de droite à celle passée en paramètre.
    - Définir la case de gauche de celle passée en paramètre à la case courante.
      (« A – B » Si A est à gauche de B, alors B est à droite de A)
- Ecrivez un programme de test dans le fichier **prog.cpp** de façon à ce que (des indications sont fournies dans le code) :
  - Quatre cases A, B, C, D soient disposées selon la configuration suivante :

A	B
C	D

    - A est en position $(0,0)$, B en $(1,0)$, C en $(0,1)$ et D en $(1,1)$
    - A est voisine de B, B de D et D de C. Cela implique qu'aucun chemin n'est possible entre A et C.
  - Implémentez la partie affichage de façon à ce que les cases s'affichent de la façon ci-contre.

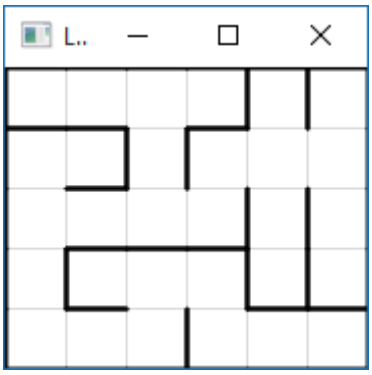


    - Pour cela, créez une procédure **drawCase()** prenant en paramètre la case à dessiner. Cette méthode affichera une case comme étant un carré de taille spécifiée par la constante globale g_szCase. Le carré aura son fond blanc par défaut, jaune si la case a été visitée et rouge si elle se situe sur un chemin menant à un cul-de-sac. Enfin, la fonction affichera également les murs sous forme de traits gris et fins si une connexion existe entre la case dessinée et sa voisine (exemple entre A et B) ou de traits noirs et épais si aucun passage n'est possible vers sa voisine (exemple entre A et C)

Exercice 2 Programme complet : résolution d'un labyrinthe

L'objectif est ici de réaliser le labyrinthe.

- Le code de la classe **CMaze** prendra place dans les fichiers **Maze.h** et **Maze.cpp**. Cette classe devra être munie de :
  - Données :
    - Un tableau 2D de cases
    - Une liste de pointeur vers les cases constituant le chemin vers la sortie (se construisant au fur et à mesure des pas de l'algorithme)
    - Un pointeur vers la case de départ
    - Un pointeur vers la case d'arrivée
  - Comportements :
    - Effectuer un pas de l'algorithme de résolution
    - Accesseurs en lecture aux données internes
- Implémentez dans un premier temps la construction du labyrinthe.
  - Dans le constructeur, vous devez initialiser toutes les cases du tableau de façon à ce que le labyrinthe corresponde à celui représenté ci-dessous :



    - Réaliser cette initialisation manuellement dans le code est fastidieux, peu portable (en cas de modification du labyrinthe) et susceptible d'erreurs. Je vous recommande de réaliser un petit algorithme simple d'initialisation à partir d'une chaîne de caractères. Le labyrinthe ci-dessus peut être modélisé par la chaine suivante :

```
string strMaze = "_ _     _| | |"
                 "  _| |      |"
                 "   _ _ _| | |"
                 " |_     |_|_|"
                 "_ _ _|_ _ _|";
```

Chaque case y est représentée par deux caractères. Les caractères d'indices pairs (0, 2, 4, …) correspondent à la liaison avec la case d'en dessous. Un espace ' ' correspond à une liaison alors qu'un '_' correspond à un mur. Les caractères d'indices impairs (1, 3, 5, …) correspondent à la liaison avec la case de droite. Un espace ' ' correspond à une liaison alors qu'un '|' correspond à un mur.

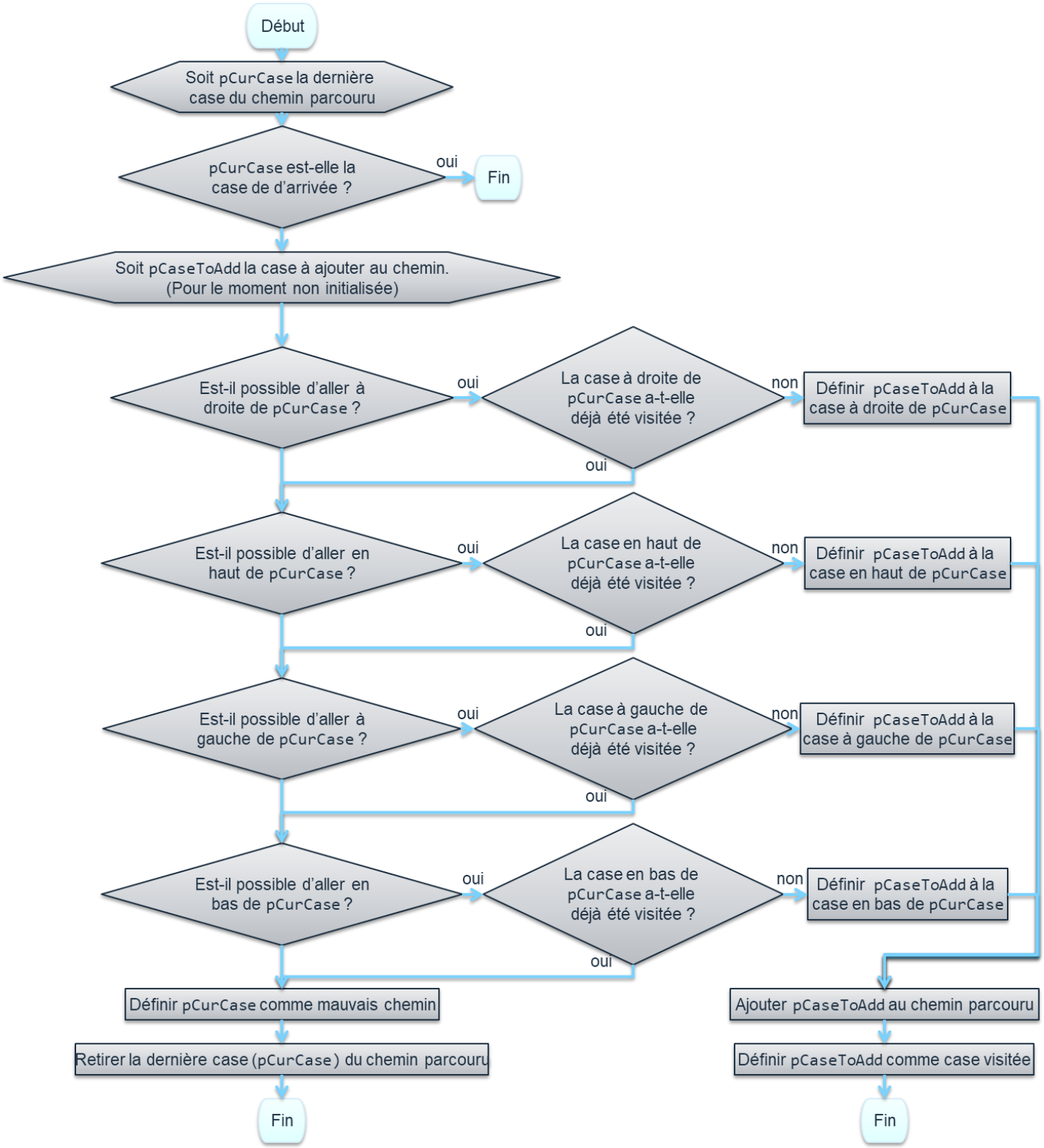
En parcourant la chaîne de caractères en même temps que l'on parcourt toutes les cases du tableau, il est possible de réaliser l'initialisation de façon automatique.

- Créez dans le programme principal un objet de type **CMaze**. Référez-vous aux commentaires du code.
  - Faites en sorte qu'il s'affiche comme sur la figure ci-dessus.

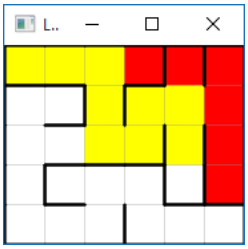
➡ Testez

Implémentez maintenant l'algorithme de résolution du labyrinthe dans la méthode **`step()`**.

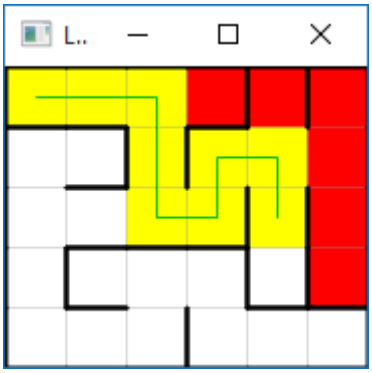
➡ Un diagramme de l'algorithme est représenté sur la page suivante.



- Modifiez le programme principal pour intégrer cette fonctionnalité sur l'appui de la barre d'espace.
- Modifiez la section de l'affichage pour que les cases visitées s'affichent en jaune et, si elles correspondent à un mauvais chemin, en rouge.
  - Testez (le labyrinthe doit pouvoir s'afficher comme ci-contre)



- Modifiez la section de l'affichage pour que le chemin idéal de résolution s'affiche en une ligne verte reliant les cases du chemin.



- (FACULTATIF) Faites en sorte d'afficher un drapeau (**`start.png`**) sur la case de départ, un bonhomme (**`me.png`**) sur la case courante et le sigle d'une sortie (**`exit.png`**) sur la case d'arrivée.