**Rappels**

Création du type de données et **etAction** 🡪 QueueAction.h

Création du type **etQueueAction** 🡪 QueueAction.h

Créé trois fonctions qui permettent de manipuler notre Queue :

* Une fonction d’initialisation 🡪 sera appelée une seule fois
* Une fonction AddQUeue pour ajouter une Action dans la Queue 🡪 sera appelée plusieurs fois
* Une fonction GetQueue pour retirer une Action de la Queue 🡪 sera appelée plusieurs fois
* Les prototypes de ces fonctions 🡪 QueueAction.h
* L’implémentation des trois fonctions 🡪 QueueAction.cpp

**Fichier Candy.cpp**

Fichier qui va contenir :

#include « QueueAction.h »

Main()

{

Struct etQueueAction stQueueAction ;

InitialisationQueue(&stQueueAction) ;

}

**Matrice de jeu (grille du jeu)**

On va se créer un nouveau type de données que l’on va appeler :

Struct etMatrice 🡪 Matrice.h

On va également placer dans ce fichier les prototypes des fonctions que nous aurons besoin

L’implémentation des fonctions va se trouver dans le fichier Matrice.cpp

**La Matrice de Jeu**

Cela sera un tableau à deux dimensions qui va permettre de stocker les pions et la gélatine. En clair, c’est la grille du jeu.

La grille du jeu va se composer de cases qui vont contenir les pions et la gélatine.

On va se créer un nouveau type de données que l’on va appeler :

Struct etCase

{

Couleur du pion qui occupe la case 🡪 être un caractère {‘J’, ‘V’, ‘B’, ‘R’, ‘M’}

Présence ou non de Gélatine sur la case

}

Cette nouvelle structure va aussi se trouver dans le fichier Matrice.h

Ensuite, on va y trouver la définition de notre Matrice de jeu :

Struct etMatrice

{

Struct etCase tMatrice[TAILLE1][TAILLE2] ;

}

**Fonctions**

On va commencer par une première fonction qui sera la fonction d’Initialisation.

InitialisationMatrice()

Que va faire cette fonction ?

Elle va initialiser chaque case de la Matrice avec :

1. Une couleur de pion choisie aléatoirement
2. Présence ou non de gélatine sur la case

**TAILLE1 et TAILLE2**

Dans un programme (dans un fichier .cpp), quand une donnée constante se répété plusieurs fois ET que cette constante est susceptible de changer dans le futur, alors, on a tout intérêt à définir cette constante une seule fois.

#define **NomMacro** **Texte**

Texte source

Le pré-processeur va examiner, le texte source et chaque fois qu’il va rencontrer une chaîne de caractères qui correspond à la Macro alors il va le substituer par le texte associé.

Exemple :

**Dans le fichier Matrice.h :**

#define **TAILLE1** **10**

#define **TAILLE2** **10**

Struct etMatrice

{

Struct etCase tMatrice[**TAILLE1**][**TAILLE2**] ;

}

Dans le fichier **Matrice.cpp** :

#include « Matrice.h »

Fonction(…)

{

Int dI, dJ ;

For (dI = 0 ; dI < **TAILLE1** ; …)

}

Quand on va lancer la compilation, elle ne s’exécute pas de suite. Il y a d’abord le travail du pré-processeur. Il va traitre toutes les commandes précédées par le caractère **‘#’**.

Ce n’est QUE lorsque toutes les commandes précédées du ‘#’ auront été traitées que la compilation peut réellement commencer.

Pour les commandes #define, il va simplement substituer un texte par un autre. Exemple :

**Dans le fichier Matrice.h :**

#define **TAILLE1** **10**

#define **TAILLE2** **10**

Struct etMatrice

{

Struct etCase tMatrice[**10**][**10**] ;

}

Dans le fichier **Matrice.cpp** :

~~#~~include « Matrice.h »

Struct etMatrice

{

Struct etCase tMatrice[**10**][**10**] ;

}

Texte source qui va être compilé

Fonction(…)

{

Int dI, dJ ;

For (dI = 0 ; dI < **10** ; …)

}