**Exercice**

## La Queue

Pour faire fonctionner le jeu, nous allons utiliser une file (Queue) qui contiendra les actions qui doivent être réalisées. Nous l’appellerons Action Queue. Dans cette Queue, nous y placerons les actions à réaliser.

Cette Queue prendra la forme d’un tableau qui pourra avoir une taille dynamique. Chaque case du tableau étant une action.

### Implémentation de la Queue

Une file ou une queue est une structure de données qui satisfait à l’ordre FIFO : First In, First Out. La première action introduite sera la première à être traitée.

Une Queue peut être implémentée de différentes manières. Dans le cadre de cet exercice, on va utiliser un tableau. Dans une version simplifiée de l’exercice, on peut travailler avec un **tableau de taille fixe**. Dans une version plus complète, on peut travailler avec un tableau de taille dynamique.

**Exemple** : un tableau de 8 éléments

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

Chaque élément de ce tableau est une **structure de type Action**.

Une **Action** comprendra les données suivantes :

* Le nom de l’action : tableau de caractères de taille 20
* Les coordonnées X et Y d’un premier pion en format integer
* Les coordonnées X et Y d’un autre pion en format integer

Initialement, la Queue est vide :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

J’ajoute un élément, Action4 :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Action 4 |  |  |  |  |  |  |  |

J’ajoute un second élément, Action 5 :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Action 4 | Action 5 |  |  |  |  |  |  |

Si je souhaite retirer un élément de la Queue, je dois prendre le premier élément :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Action5 |  |  |  |  |  |  |  |

Dès qu’une action a été traitée, il faut la retirer de la Queue. Cela voudrait dire qu’il faut faire reculer toutes les autres actions vers le sommet de la Queue :

Next

Top

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Action 4 | Action 5 | Action6 | Action 7 | Action 8 |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Action 5 | Action 6 | Action7 | Action 8 |  |  |  |  |

Dans une Queue, il y deux Index qui sont importants :

* L’index qui me donne le début de la Queue : Top
* L’index qui me donne la fin de la Queue : Next

Mais, on peut éviter ces mouvements si on voit ce tableau comme un anneau :

Top

Next

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Action 4 | Action 5 | Action6 |  |  |  |  |  |

Top = 0

Next=3

On enlève un élément de la Queue :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Action 5 | Action6 |  |  |  |  |  |

Top = 1

Next = 3

On ajoute un élément :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Action 5 | Action6 | Action7 |  |  |  |  |

Top = 1

Next = 4

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Action 5 | Action6 | Action7 | Action8 |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Action6 | Action7 | Action8 |  |  |  |

Top = 2

Next = 5

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Action6 | Action7 | Action8 | Action9 |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Action6 | Action7 | Action8 | Action9 | Action10 |  |

Top = 2

Next = 7

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Action6 | Action7 | Action8 | Action9 | Action10 | Action11 |

Top = 2

Next = 0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Action12 |  | Action6 | Action7 | Action8 | Action9 | Action10 | Action11 |

Top = 2

Next = 1

Donc, trois index sont nécessaires :

* **Top** : Index du premier élément dans la Queue
* **Next** : Index du prochain élément libre dans la Queue
* **Size** : Index Maximal de la Queue

Dans la Queue, il faut évidemment aussi y placer le **tableau avec les Actions**.

Trois fonctions sont à écrire :

1. **L’initialisation** qui initialise la Queue
2. La fonction **Add** qui permet d’ajouter une action que l’on passe en paramètre. La fonction Add doit aussi indiquer si l’insertion a pu se faire
3. La fonction **Get** qui retourne une action. Le Get doit aussi indiquer si une action a pu être retournée

Point de vue Fichiers :

* 1 fichier Candy.cpp qui va contenir la fonction main
* 1 fichier Queue.cpp qui va contenir l’implémentation des trois fonctions
* 1 fichier **Queue.h** qui va contenir :
* On va y trouver le prototype de chaque fonction se trouvant dans le fichier Queue.cpp
* Mais en plus, on va y trouver la définition du nouveau type **struct etQueue**.

Pourquoi placer la définition du type Queue dans le fichier .h ?

Parce que la fonction main, va avoir besoin d’utiliser la Queue. La fonction main est la fonction qui va déclarer la Queue utilisée dans le programme.

Dans le fichier Candy.cpp, on va y trouver :

#include « Queue.h »

Main()

{

Struct etQueue stQueueAction ;

InitialiserQueue(&stQueueAction) ;

}

Marche à suivre :

La première question à se poser c’est : quelle est la composition du nouveau type etQueue ?

L’écriture de la fonction InitialisationQueue

Quelle est la composition du nouveau type Action ? (réponse dans le texte ci-dessus). La définition du nouveau type Action, peut aussi se placer dans le fichier Queue.h

L’écriture de la fonction AddQueue

L’écriture de la fonction GetQueue