Projet Ricochet Robots

Le groupe est constitué de ASSE Romain, FERRE Cedric, PROCCACIA Paul et LEQUEUX Alexis.

Voici le lien menant au dépôt GitHub : https://github.com/ECN-SEC-SMP/tp-note-2023-ricocheh.

Voici le lien menant au dépôt Replit : https://replit.com/join/cuddzfyflh-cedricferre

La documentation technique est disponible via le fichier readme.md ou readme.pdf à la racine du projet ainsi que les fichiers .html et .tex fournis par Doxygen dans les dossiers html et latex disponibles également à la racine du projet.

Introduction

Ce projet a pour but d'implémenter le jeu ricochet. Ce projet sera l'occasion d'appliquer et donc de valider l'ensemble des concepts vues pendant les cours.

Ce projet sera réalisé en C++. Le suivi de version sera avec GIT.

Regle de codage

Attributs

Par défaut, les attributs seront privés, il sera cependant possible de modifier ces derniers avec des accesseurs.

Convention de nommage

- Types start with upper case: `MyClass`.
- Functions and variables start with lower case: `myMethod`.
- Constants are all upper case: const double `PI=3.14159265358979323;`.
- Macro names use upper case with underscores: `INT_MAX`.
- Template parameter names use camel case: `InputIterator`.
- All other names use snake case: `unordered_map`.

Langue

Le code sera rédigé en Anglais, les commentaires seront rédigés en francais.

Doxygen

Les fichiers utiliseront l'entête suivante:

```
/**
 * @file exemple.h
 * @brief La déclaration de la classe Exemple
```

```
 * si besoin auteur, version et date
 */

/******* Native include *******/
#include <iostream>

/******* Project include ******/
#include "MonFichier.h++"

/******* Espace de nommage ****/
using namespace std;
```

- les constantes et/ou macros :

```
/**
 * @def NB
 * @brief Définit le nombre 42 !
*/
#define NB 42 //!< Un nombre NB
```

- les types énumérés :

```
/**
 * @enum TEnum
 * @brief Description du type énuméré ...
 *
 * @var TEnum Val1
 * @brief Description de Val1 ...
 */
enum TEnum //! Un type énuméré ...
{
    Val1,
    Val2    //!< Description de Val2 ...
};
```

- les structures :

```
/**
 * @struct Etat
 * @brief Structure ...
 *
 */
struct Etat
{
    bool present; //!< Membre définissant ...
};
```

- les classes :

```
/**
```

```
 * @class       Exemple exemple.h "exemple.h"
 * @brief       La déclaration de la classe Exemple
 * @details     La classe \c Exemple permet de montrer l'utilisation des \em tags \b Doxyge
 * @author      Thierry vaira <thierr.vaira@gmail.com>
 * @version     0.1
 * @date        2020
 * @note        Une note à l'attention de ceux qui lisent les notes
 * @pre         Initialisez d'abord le système
 * @post        L'objet est initialisé ou pas
 * @bug         La copie est impossible ou illégale
 * @warning     Une mauvaise utilisation peut faire planter votre application (c'est votre
 * @attention   Il faut toujours faire attention
 * @remark      Une remarque à faire ?
 * @copyright   GNU Public License.
 */
class Exemple
{
};
```

- les attributs :

```
class Exemple
{
    private:
        int a; //!< a est ...
};
```

Les méthodes:

```
/**
 * @brief Constructeur par défaut de la classe Exemple.
 *
 * @see   Exemple::Exemple(int a)
 */
Exemple::Exemple() : a(0)
{
}

/**
 * @brief Constructeur de la classe Exemple.
 * @overload
 * @param a la valeur initiale de l'attribut a
 *
 * @see   Exemple::Exemple(int a)
 * @see   https://doc.qt.io/qt-5/qdatetime.html
 */
Exemple::Exemple(int a) : a(a)
{
```

```
    QDateTime maintenant = QDateTime::currentDateTime();
    qDebug() << Q_FUNC_INFO << maintenant.toString("dd/MM/yyyy") << maintenant.toString("hh
}

/**
 * @brief Accesseur de l'attribut a
 * @callergraph
 * @return a la valeur de l'attribut a
 * @retval int la valeur de l'attribut a
 */
int Exemple::getA() const
{
    return a;
}

/**
 * @brief Mutateur de l'attribut a
 * @callgraph
 * @param a ...
 * @exception range_error Si a est négatif
 */
void Exemple::setA(int a)
{
    if(a < 0 || a > NB)
        throw range_error("erreur plage");
    this->a = a;
    qDebug() << Q_FUNC_INFO << "a" << getA();
}

/**
 * @brief Montre le sens des paramètres
 * @param[in]       a1  ...
 * @param[out]      a2  ...
 * @param[in,out]   a3  ...
 */
void Exemple::copy(const int &a1, int &a2, int *a3)
{
    /**
     * @todo Implémenter la méthode
     */
}
```

Choix technique

Dans un premier temps, définissons les besoins fonctionnels de notre projet: Diagramme de cas d'utilisation

Pour respecter le cahier des charges, notre main respectera la séquence ci dessous.

Diagramme de flux

Afin de mieux s'organiser en équipe, nous avons préalablement découpé notre projet pour diviser le travail. Voici le diagramme de classe de notre projet. Diagramme de classe

Classe Sablier

Cette classe est utilisée pour la gestion du temps dans le jeu. Elle utilise la librairie . Dans ce projet, le sablier est utilisé pour décompter des temps de 1 minute. Celui-ci est utilisé après qu'un joueur ait annoncé le nombre de déplacement qu'il compte faire pour atteindre l'objectif. Pendant son écoulement, les autres joueurs peuvent proposer un nombre de déplacement en deçà du premier joueur si c'est possible. Quand le sablier est écoulé, le joueur avec le nombre de déplacement le moins élevé peut jouer.

Pour répondre à ce cahier des charges, nous avons créé une classe Sablier avec une méthode de qui lance le sablier, une qui l'arrête, une qui le remet à 0 et une qui permet de savoir si le temps est écoulé ou non.

Classe Robot

Cette classe est utilisée pour créer et gérer les déplacements des robots. Dans ce projet, 4 robots sont présents. Ces robots peuvent se déplacer dans quatre directions possibles (haut, bas, gauche et droite) jusqu'à rencontrer un autre robot ou bien un mur.

Pour répondre à ce cahier des charges, nous avons créé une classe Robot qui permet d'instancier les quatre robots. Dans cette classe, il y a deux méthodes qui permettent de savoir quelle sont les positions X et Y, deux méthodes qui mettent à jour les positions que ça soit en X ou en Y tout en mettant à jour le statut d'occupation sur le plateau de jeu. Pour finir, nous avosn créé une méthode déplacement qui a pour objectif de déplacer un robot dans une des quatre directions possibles selon le choix effectué par le joueur. Pour ce faire, la fonction passe de case en case et s'assure de pouvoir les traverser en fonction des murs et des robots présents sur les cases qu'il parcoure.

Classe Joueur

Cette classe est utilisée pour créer et gérer les joueurs qui intéragissent avec le jeu. Les joueurs ont un prénom et un surnom. Chaque joueur annonce une estimation d'un nombre de coups nécéssaires pour gagner la manche, ce nombre de coup est donc propre au joueur et son accès est rendu possible par un accesseur.

Cette classe comporte des accesseurs sur le prénom, le surnom et le nombre de coups. De même, elle comporte des mutateurs sur le prénom, le surnom et le nombre de coups. Elle comporte également une méthode *chooseRemStrokes()* qui permet de demander au joueur son estimation et qui utilise *setRemStrokes()* pour renseigner la donnée dans l'attribut *remStrokes* de l'objet courant.

Classe MapManager

Cette classe est utilisée pour faire des opérations sur le plateau et nottament pour l'afficher au travers de la fonction *displayBoard()*.

Cette classe comporte un constructeur privé pour empecher plusieurs instanciations de l'objet MapManager. A la place on utilise un pointeur sur MapManager qui est *static* afin d'avoir un MapManager commun pour tous les objets (cela n'a pas de sens d'avoir plusieurs instance de ce composant). On y retrouve donc également un accesseur sur le pointeur instance qui permet d'accéder à cette instance unique de MapManager. Enfin cette classe comporte les méthodes *loadBoard()* qui permet de charger un plateau et de lui ajouter des murs et des robots, *displayBoard()* qui affiche le plateau avec les murs et les robots qu'il contient dans un terminal, et *addBoardLimit()* qui pose des murs sous tout le pourtour du plateau de jeu.