Licence 3 Informatique
Module : théorie des langages
et compilation

Projet
city (part 2) : Interface graphique

Ce document contient les informations nécessaires pour l'utilisation de l'interface graphique et une description du jeu de tests. En addition de ce document vous pouvez récupérer le jeu de tests (qui sera utilisé pour la correction des projets) sur moodle.

1 Éléments à ajouter pour la compilation

Pour utiliser l'interface il y a plusieurs pré-requis. Tout d'abord récupérer le dossier compressé `IG.zip`.

- Avoir Qt 5 installé sur sa machine ;
- Décompresser `IG.zip` (à récupérer sur moodle) dans le répertoire de votre projet (attention celui-ci va écraser le fichier `parser/main.cc`) ;
- Modifier le `CMakeList` avec les instructions en annexe ;
- Ajouter un objet VilleGUI à vos fichiers driver.hh et driver.cpp (exemple en annexe).

2 Méthodes permettant la manipulation de l'interface graphique

L'interface graphique propose des méthodes simples permettant uniquement de modifier la taille d'une ville, de construire ou de détruire des maisons et des routes et de modifier les paramètres d'une maison. Les méthodes à votre disposition et leur description sont les suivantes :

- `void setRayon(unsigned int x)` : Modifie le rayon de la ville (et donc sa taille). La carte de la ville est toujours affichée dans une fenêtre de 1000x1000 pixels et les éléments de la ville sont redimensionnés en fonction du rayon de la ville. Attention, si une ville est redimensionnée les maisons et les routes dépassant le nouveau rayon de la ville doivent être détruites. L'appel à la méthode ne fait que changer la taille de la carte c'est à vous de gérer le reste.
- `unsigned int getRayon()` : Renvoie le rayon (la taille) de la ville.
- `void construireMaison(int x, int y, int z)` : Ajoute une maison sur l'interface graphique aux coordonnées indiquées. Attention la méthode ne vérifie pas s'il y a une erreur sur les coordonnées entrées. Les maisons sont gérées à l'aide d'un vecteur et ajoutées par push_back, elles ont donc en mémoire une numérotation suivant l'ordre de construction.
- `void detruireMaison(int x, int y, int z)` : Supprime la maison, située aux coordonnées indiquées, du vecteur de maisons et disparaît sur l'interface graphique. Il n'est pas possible de détruire une maison autrement que par ses coordonnées. S'il existe une route reliant cette maison elle ne sera pas détruite.
- `void construireRoute(int x1, int y1, int z1, int x2, int y2, int z2)` : Ajoute une route sur l'interface graphique entre les coordonnées (x1,y1,z1) et (x2,y2,z2). Une route est ajoutée même s'il n'existe pas de maison à l'emplacement indiqué ou d'obstacles sur la route. Les routes sont aussi gérées par un vecteur et ajoutées dans l'ordre des appels à la méthode.
- `void detruireRoute(int x1, int y1, int z1, int x2, int y2, int z2)` : Supprime une route entre les coordonnées (x1,y1,z1) et (x2,y2,z2) si elle existe. La route disparaît de l'affichage dans l'interface graphique et le vecteur de routes est mis à jour. Il reste judicieux de traiter

l'existence de la route avant l'appel à cette méthode. Les maisons reliées par cette route ne sont pas détruites.

— `void setCouleur(int x, int y, int z, int r, int g, int b)` : Modifie la couleur au format RGB de la maison aux coordonnées (x,y,z). Par défaut la maison a une couleur rouge (255,0,0). La méthode ne vérifie pas si les valeurs sont bien comprises dans le spectre RGB ni s'il existe une maison à la position donnée à vous de gérer les erreurs.
— `CouleurGUI getCouleur(int x, int y, int z,)` : renvoie la couleur de la maison en position (x,y,z) au format CouleurGUI composé de trois `int` (rouge,vert,bleu). Pour accéder aux couleurs il faut utiliser `getCouleur(int x, int y, int z,).rouge`, `getCouleur(int x, int y, int z,).vert` `getCouleur(int x, int y, int z,).bleu`.
— `void setOrientation(int x, int y, int z, int o)` : Modifie l'orientation de la maison en position (x,y,z). Par défaut l'orientation est de 0° (maison orientée vers le bas). L'orientation s'effectue dans le sens des aiguilles d'une montre (90° correspond à une maison face à gauche). La méthode ne vérifie pas si les coordonnées correspondent à une maison ni la validité de l'orientation.
— `int getOrientation(int x, int y, int z)` : Renvoie l'orientation en degrés de la maison aux coordonnées (x,y,z).

L'utilisation de la classe VilleGUI s'effectue dans les fichiers `driver.cc` et `driver.hh`, l'interface graphique de la ville est initialisée lors de l'appel au constructeur de Driver. Un pointeur V vers une VilleGUI est créé. Vous pouvez alors faire appel aux différentes méthodes ci-dessus dans n'importe quelle méthode de driver de cette façon :

```
1 void Driver::ConstruireUneMaison(int x, int y, int z) {
2     V->construireMaison(x,y,z);
3 }
```

Vous pouvez ensuite faire appel aux méthodes de `Driver` dans votre fichier `parser.yy`

Modifications sur le fichier CMakeList (lignes à ajouter en rouge)

```
project(bison_expressions)
cmake_minimum_required(VERSION 3.1.0)

find_package(Qt5 COMPONENTS Widgets REQUIRED)

set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)
set(CMAKE_AUTOUIC ON)

set(CMAKE_CXX_STANDARD 14)
set(CMAKE_CXX_STANDARD_REQUIRED on)
set(CMAKE_CXX_EXTENSIONS off)
if(CMAKE_CXX_COMPILER_ID MATCHES "GNU")
    add_compile_options(-Wall -Wpedantic)
endif()

find_package(BISON)
find_package(FLEX)

include_directories(parser expressions ${CMAKE_CURRENT_BINARY_DIR})
file(GLOB EXPRESSION_HEADERS
    expressions/contexte.hh
    expressions/expression.hh
    expressions/expressionBinaire.hh
    expressions/expressionUnaire.hh
    expressions/constante.hh
    expressions/variable.hh
    )

add_library(expressions
    expressions/contexte.cc
    expressions/expressionBinaire.cc
    expressions/expressionUnaire.cc
    expressions/constante.cc
    expressions/variable.cc
    ${EXPRESSION_HEADERS}
    )

add_library(GUI
    IG/main.cc
    IG/villeGUI.cc
)
target_link_libraries(GUI Qt5::Widgets)

BISON_TARGET(parser parser/parser.yy ${CMAKE_CURRENT_BINARY_DIR}/parser.cpp VERBOSE ${
    CMAKE_CURRENT_BINARY_DIR}/automate.txt)
FLEX_TARGET(scanner parser/scanner.ll  ${CMAKE_CURRENT_BINARY_DIR}/scanner.cpp)
ADD_FLEX_BISON_DEPENDENCY(scanner parser)
add_executable(${PROJECT_NAME}
    parser/main.cc
    parser/driver.hh
    parser/driver.cc
    parser/scanner.hh
    parser/scanner.ll
    parser/parser.yy
    ${BISON_parser_OUTPUTS}
    ${FLEX_scanner_OUTPUTS}
)
target_link_libraries(${PROJECT_NAME} expressions GUI)

add_executable(exemple_expression
    exemple_expression/main.cc)

target_link_libraries(exemple_expression expressions)
```

Modifications sur du fichier driver.hh (lignes à ajouter en rouge)

```
#ifndef DRIVER_H
#define DRIVER_H

#include <string>
#include "../IG/villeGUI.hh"
#include "contexte.hh"

class Driver {
private:
    Contexte variables;
    VilleGUI* V;
public:
    Driver();
    ~Driver();

   /* TODO
    const   Contexte& getContexte() const;
    double  getVariable(const std::string& name) const;
    void    setVariable(const std::string& name, double value);
  */
    VilleGUI* getVille() {return V;}
};

#endif
```

Modifications sur du fichier driver.cc (lignes à ajouter en rouge)

```
#include "driver.hh"
#include <iostream>

Driver::Driver() {V = new VilleGUI();}
Driver::~Driver() {}

/*
const Contexte& Driver::getContexte() const {
    //TODO Retourne le contexte pour qu'il soit accessible en dehors de la classe Driver
}

double Driver::getVariable(const std::string & name) const {
    //TODO Retourne la valeur de la variable name
}

void Driver::setVariable(const std::string & name, double value) {
    //TODO Affecte une valeur à une variable avec l'utilisation du contexte variables
}
*/
```