XPFIGHT

Mission n°8 – Portfolio – Antoine Gandelin

XPFIGHT

I. Scripts de création de la base de données

Voir les fichiers xpfight_admin.sql et xpfight_structure.sql

1. Création de la base de données et du rôle d'administrateur

```
create database xpfight;
create role xpfight_admin password 'admin' login;
grant all on database xpfight to xpfight_admin;
```

2. Création de la structure de la base de données

```
begin transaction;

create table score
    (
    id_score    serial primary key,
    pseudo      varchar(20),
    date_score  timestamp,
    ip          inet,
    grille      integer,
    nb_tues     integer,
    perso1      char(1) check (perso1 in ('A', 'M', 'S')),
    perso2      char(1) check (perso2 in ('A', 'M', 'S')),
    perso3      char(1) check (perso3 in ('A', 'M', 'S')),
    pos1        integer,
    pos2        integer,
    pos3        integer
    );
commit;
```

II. Lancement des scripts de création

Lancement des scripts de création de la base. On se connecte en DBA (postgres) :

```
$ su postgres
```

On lance le client de Postgresql :

```
$ psql
```

Sous psql, on lance le script de création de la base et du rôle :

```
postgres=# \i /chemin_ou_se_trouve_le_script/xpfight_admin.sql
CREATE DATABASE
CREATE ROLE
GRANT
```

On quitte le client psql, on quitte la session bash postgres et on lance le client psql avec les valeurs voulues :

```
$ psql xpfight -U xpfight_admin -h 127.0.0.1

xpfight=> \i /chemin_ou_se_trouve_le_script/xpfight_structure.sql
BEGIN
CREATE TABLE
COMMIT
```

On vérifie que la table a été correctement crée :

```
xpfight=> \d
             Liste des relations
 Schéma  |  Nom    |  Type  | Propriétaire
 --------+---------+--------+--------------
 public  | score   | table  | dc_admin
(1 ligne)
```

III. Programmation C++ avec QtCreator

1. Fichier : fenetre.cpp

1) Initialisation du jeu

```
void fenetre::init()
    {
    // Visibilité de l'envoi du score
    // Changement des boutons
    ui->b_envoi->setVisible(false);
    ui->b_enregistrer->setVisible(false);
    ui->t_nom->setVisible(false);
    ui->t_nom->clear();
    ui->b_fight->setVisible(true);
    ui->b_envoi->setVisible(false);
    connect(ui->b_envoi, SIGNAL(clicked()), this, SLOT(envoi()));
    connect(ui->b_enregistrer, SIGNAL(clicked()), this, SLOT(save()));
    // Mise à jour du array
    for(int i=0; i<100; i++)
        {
        neutre * n;
        n = new neutre(theme);
        per.at(i)=n;
        per.at(i)->set_position(i);
        connect(per.at(i), SIGNAL(clicked(int)), this, SLOT(change(int)));
        // Mise en place de la grille
        ui->la_grille->addWidget(per.at(i), i/10, i%10);
        }
    // Distribution des monstres
    int graine;
    graine = ui->t_numero->text().toInt();
    srand(graine);
    int compteur = 0;
    while(compteur<10)
        {
        unsigned int i = rand()%100;
        if(per.at(i)->get_val() == 0)
            {
            monstre * m;
            m = new monstre(theme);
            m->set_position(i);
            per.at(i) = m;
            ui->la_grille->addWidget(per.at(i), i/10, i%10);
            compteur++;
            }
        }
    }
```

2) Distribution d'un numéro de grille aléatoire

```
fenetre::fenetre(QWidget *parent) : QMainWindow(parent), ui(new Ui::fenetre)
    {
    ui->setupUi(this);

    // N° de grille aléatoire
    srand(time(nullptr));
    int gr = rand()%100+1;
    QString st;
    st.setNum(gr);
    ui->t_numero->setText(st);
```

.

3) Création d'un menu avec plusieurs fonctionnalités

```
    // MENU
    QMenu * menu = new QMenu("XPFight", this);

    QAction * about = new QAction("A &Propos",this);
    connect(about, SIGNAL(triggered()),this, SLOT(a_propos()));
    menu->addAction(about);

    QAction * aide = new QAction("&Aide",this);
    connect(aide, SIGNAL(triggered()),this, SLOT(aide()));
    menu->addAction(aide);

    //separateur
    menu->addSeparator();

    QAction * quit = new QAction("&Quitter",this);
    connect(quit, SIGNAL(triggered()),this, SLOT(quitter()));
    menu->addAction(quit);


    //Ajout du Menu à la barre de menu
    ui->barre_menu->addMenu(menu);
```

4) Construction des thèmes

```
// Emplacement des thèmes - récupération du thème par défaut
theme = fic.readLine();
theme.remove(theme.size()-1,theme.size());
theme.remove(0,6);
fic.close();

// Construction du menu "Thème"
QMenu * menut = new QMenu("Thème", this);
QDir d_theme("themes");
d_theme.setFilter(QDir::NoDotAndDotDot|QDir::Dirs);
QStringList l = d_theme.entryList();
int taille = l.size();
for(int i = 0; i<taille; i++)
    {
    QAction * q = new QAction;
    q->setText(l.at(i));
    connect(q, SIGNAL(triggered()),this, SLOT(changer_theme()));
    menut->addAction(q);
    }
ui->barre_menu->addMenu(menut);
init();
connect(ui->b_init, SIGNAL(clicked()), this, SLOT(init()));
connect(ui->b_fight, SIGNAL(clicked()), this, SLOT(resultat()));
}
```

5) Position des personnages

```
void fenetre::change(int pos)
    {
    if(per.at(pos)->get_val() ==  0)
        {
        archer * a;
        a = new archer(theme);
        a->set_position(pos);
        connect(a, SIGNAL(clicked(int)), this, SLOT(change(int)));
        per.at(pos) = a;
        ui->la_grille->addWidget(per.at(pos), pos/10, pos%10);
        }
    else
        {
        if(per.at(pos)->get_val() == 1)
            {
            mage * m;
            m = new mage(theme);
            m->set_position(pos);
            connect(m, SIGNAL(clicked(int)), this, SLOT(change(int)));
            per.at(pos) = m;
            ui->la_grille->addWidget(per.at(pos), pos/10, pos%10);
            }
        else
            {
            if(per.at(pos)->get_val() == 2)
                {
                soldat * s;
                s = new soldat(theme);
                s->set_position(pos);
                connect(s, SIGNAL(clicked(int)), this, SLOT(change(int)));
                per.at(pos) = s;
                ui->la_grille->addWidget(per.at(pos), pos/10, pos%10);
                }
            else
                {
                neutre * n;
                n = new neutre(theme);
                n->set_position(pos);
                connect(n, SIGNAL(clicked(int)), this, SLOT(change(int)));
                per.at(pos) = n;
                ui->la_grille->addWidget(per.at(pos), pos/10, pos%10);
                }
            }
        }
    }
```

6) Résultat et conformité de la grille

```
void fenetre::resultat()
    {
    // Vérification de la conformité de la grille
    unsigned int nb_monstres = 0;
    unsigned int nb_persos = 0;
    unsigned int type = 0;
    for(int i=0; i<100; i++)
        {
        type = per.at(i)->get_val();
        if(type)
            {
            if (type == 9)
                {
                nb_monstres++;
                }
                else
                {
                nb_persos++;
                }
            }
        }
    if(nb_monstres == 10 && nb_persos == 3)
        {
        int res = 0;
        for(int i=0; i<100; i++)
            {
            // Appel de toutes les fonctions fight des persos
            res += per.at(i)->fight(per);
            }
        // Changement des boutons
        ui->b_fight->setVisible(false);
        ui->b_envoi->setVisible(true);
        }
    else
        {
        QMessageBox mess(QMessageBox::Warning, "ATTENTION", "Pourq
        mess.exec();
        }
    }
```

7) Envoi du score

```
void fenetre::envoi()
    {
    // Réseau
    QEventLoop eventLoop;
    QNetworkAccessManager manager;
    connect(&manager, SIGNAL(finished(QNetworkReply*)), &eventLoop, SLOT(quit()));

    // Construction de la requête http

    //DEBUG
    std::cerr<<"#"<<url.toStdString()<<"#";

    QString adresse = url;
    adresse += "xpfight_server.cgi?";
    // Récupération des paramètres dans la grille
    std::ostringstream o;
    o<<"grille="<<ui->t_numero->text().toStdString();
    int nb = 0;
    for(int i=0; i<100; i++)
        {
        if(per.at(i)->get_val() == 10)
            {
            nb++;
            }
        }
    o<<"&nb_tues="<<nb;
    for(int i=0; i<100; i++)
        {
        if(per.at(i)->get_val() == 1 || per.at(i)->get_val() == 2 || per.at(i)->get_val() == 3)
            {
            // On récupère les valeurs
            o<<"&perso="<<per.at(i)->get_val();
            o<<"&pos="<<per.at(i)->get_position();
            }
        }
    adresse += o.str().c_str();
    //DEBUG
    std::cerr<<adresse.toStdString();

    QUrl url(adresse);
    QNetworkRequest req(url);
    QNetworkReply * reply = manager.get(req);
    eventLoop.exec();
    QString reponse = (QString)reply->readAll();
    if(reponse.size() < 2 )
        {
        QMessageBox mess(QMessageBox::Warning, "ATTENTION", "L'envoi n'a pas été valide.");
        mess.exec();
        }
    else
        {
        ui->b_envoi->setVisible(false);
        ui->b_enregistrer->setVisible(true);
        ui->t_nom->setVisible(true);
        jeton=reponse.toStdString();
        }
```

8) Enregistrement du score

```
void fenetre::save()
    {
    // Réseau
    QEventLoop eventLoop;
    QNetworkAccessManager manager;
    connect(&manager, SIGNAL(finished(QNetworkReply*)), &eventLoop, SLOT(quit()));

    // Construction de la requête http
    QString adresse = url;
    adresse += "xpfight_server.php?jeton=";
    // Passage du jeton
    adresse += jeton.c_str();
    adresse += "&nom=";
    adresse += ui->t_nom->text();
    QUrl url(adresse);
    QNetworkRequest req(url);
    QNetworkReply * reply = manager.get(req);
    eventLoop.exec();
    QString reponse = (QString)reply->readAll();
    // Lecture du score en xml

    //DEBUG
    std::cerr<<reponse.toStdString();



    //DEBUG
    /*
    if(reponse !="0")
        {
        QXmlStreamReader xml(reponse);
        QMessageBox mess(QMessageBox::Information, "Information", "Votre score est enregistré.");
        mess.exec();
        }
    else
        {
        QMessageBox mess(QMessageBox::Warning, "ATTENTION", "Votre score n'a pas été enregistré.");
        mess.exec();
        }
    */
    init();
    }
```

9) Quitter le jeu

```
void fenetre::quitter()
    {
    QMessageBox mess;
    int r=0;
    mess.setText("Merci d'avoir utilisé XPFIGHT.");
    mess.setInformativeText("Mais êtes-vous sûr de vouloir quitter le jeu ?");
    mess.setStandardButtons(QMessageBox::Cancel | QMessageBox::Ok);
    mess.setDefaultButton(QMessageBox::Cancel);
    mess.setIcon(QMessageBox::Question);
    r = mess.exec();
    if(r == QMessageBox::Ok)
        {
        QApplication::quit();
        }
    }
```

10) Aide du jeu

```
void fenetre::aide()
    {
    class aide a;
    a.exec();
    }
```

11) Changer le thème de la grille

```
void fenetre::changer_theme()
    {
    QString s = qobject_cast<QAction *>(sender())->text();
    theme = s;
    init();
    }
```

2. Fichier : fenetre.h

```
    private:
        Ui::fenetre *ui;
        std::array <perso*, 100> per;
        std::string jeton;
        QString url;
        QString theme;

    private slots:
        void init();
        void change(int);
        void resultat();
        void envoi();
        void save();
        // menu
        void quitter();
        void aide();
        void a_propos();
        void changer_theme();
```

3. Fichier : archer.cpp

```
#include "archer.h"

archer::archer(QString t) : perso()
    {
    theme = t;
    QPixmap pic;
    QString fic_im = "archer.png";
    fic_im = "themes/" + t + "/" + fic_im;
    pic.load(fic_im);
    setPixmap(pic);
    set_val(1);
    }
```

```
unsigned int archer::fight(std::array <perso *, 100>& a)
    {
    unsigned int nb_tues = 0;
    unsigned int centre = get_position();
    int lig = centre/10;
    int col = centre %10;
    std::vector <unsigned int> v;
    int p = 0;
    if(lig-3 >= 0)
        {
        p = (lig-3)*10+(col);
        v.push_back(p);
        }
    if(lig-2 >= 0)
        {
        p = (lig-2)*10+(col);
        v.push_back(p);
        }
    if(col-3>=0)
        {
        p = (lig)*10+(col-3);
        v.push_back(p);
        }
    if(col-2>=0)
        {
        p = (lig)*10+(col-2);
        v.push_back(p);
        }
    if(col+3 < 10)
        {
        p = (lig)*10+(col+3);
        v.push_back(p);
        }
```

4. Fichier : archer.h

```
#ifndef ARCHER_H
#define ARCHER_H

#include "perso.h"

class archer : public perso
    {
    private:
        QString theme;

    public:
        archer(QString ="classique");
        unsigned int fight(std::array <perso *, 100>&);
    };

#endif // ARCHER_H
```

5. Fichier : monstre.cpp

```
#include "monstre.h"

monstre::monstre(QString theme) : perso()
    {
    QPixmap pic;
    QString fic_im = "monstre.png";
    fic_im = "themes/" + theme + "/" + fic_im;
    pic.load(fic_im);
    setPixmap(pic);
    set_val(9);
    }

unsigned int monstre::fight(std::array <perso *, 100>& a)
    {
    return 0;
    }
```

6. Fichier : monstre.h

```
#ifndef MONSTRE_H
#define MONSTRE_H

#include "perso.h"

class monstre : public perso
    {
    public:
        monstre(QString ="classique");
        unsigned int fight(std::array <perso *, 100>&);
    };

#endif // MONSTRE_H
```

7. Fichier : neutre.cpp

```
#include "neutre.h"

neutre::neutre(QString theme) : perso()
    {
    QPixmap pic;
    QString fic_im = "neutre.png";
    fic_im = "themes/" + theme + "/" + fic_im;
    pic.load(fic_im);
    setPixmap(pic);
    }

unsigned int neutre::fight(std::array <perso *, 100>& a)
    {
    return 0;
    }
```

8. Fichier : neutre.h

```
#ifndef NEUTRE_H
#define NEUTRE_H

#include "perso.h"

class neutre : public perso
    {
    public:
        neutre(QString ="classique");
        unsigned int fight(std::array <perso *, 100>&);
    };

#endif // NEUTRE_H
```

9. Fichier : perso.cpp

```
#include "perso.h"
#include <iostream>

perso::perso() : QLabel()
    {
    setFixedSize(40,40);
    val = 0;
    pos = 0;
    }

void perso::set_position(unsigned int i)
    {
    pos = i<1000?i:0;
    }

void perso::set_val(unsigned int i)
    {
    val = i<100?i:0;
    }

void perso::mousePressEvent(QMouseEvent * e)
    {
    if (e->button() == Qt::LeftButton)
        {
        emit clicked(pos);
        }
    }

perso::~perso()
    {

    }

unsigned int perso::get_val()
    {
    return val;
    }

unsigned int perso::get_position()
    {
    return pos;
    }
```

10. Fichier : perso.h

```
#ifndef PERSO_H
#define PERSO_H

#include <QLabel>
#include <QWidget>
#include <QObject>
#include <QMessageBox>
#include <QMouseEvent>
#include <QPixmap>
#include <array>
#include <QString>

class perso : public QLabel
    {
    Q_OBJECT

    signals:
        void clicked(int);

    public:
        perso();
        ~perso();
        virtual unsigned int fight(std::array <perso *, 100>&)=0;
        void set_val(unsigned int);
        void set_position(unsigned int);
        unsigned int get_val();
        unsigned int get_position();

    protected:
        void mousePressEvent(QMouseEvent *);

    private:
        unsigned int val;
        unsigned int pos;
    };

#endif // PERSO_H
```

IV. Affichage graphique sur QtCreator

1. Affichage de la fenêtre au lancement du jeu

2. Affichage du fichier d'interface utilisateur (fenetre.ui)

3. Liste des éléments graphiques

Objet	Classe
fenetre	QMainWindo
centralwidget	QWidget
frame	QFrame
la_grille	QGridLay
la_titre	QHBo...yc
l_titre	QLabel
label	QLabel
la_commande	QVBoxLay
b_enregistrer	QPushButton
b_envoi	QPushButton
b_fight	QPushButton
b_init	QPushButton
l_numero	QLabel
t_nom	QLineEdit
t_numero	QLineEdit
verticalSpacer	Spacer
barre_menu	QMenuBar
statusbar	QStatusBar

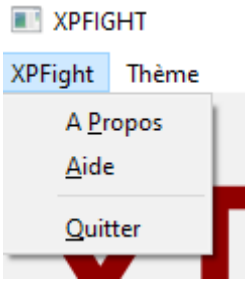
4. Placement des personnages

5. Message d’erreur

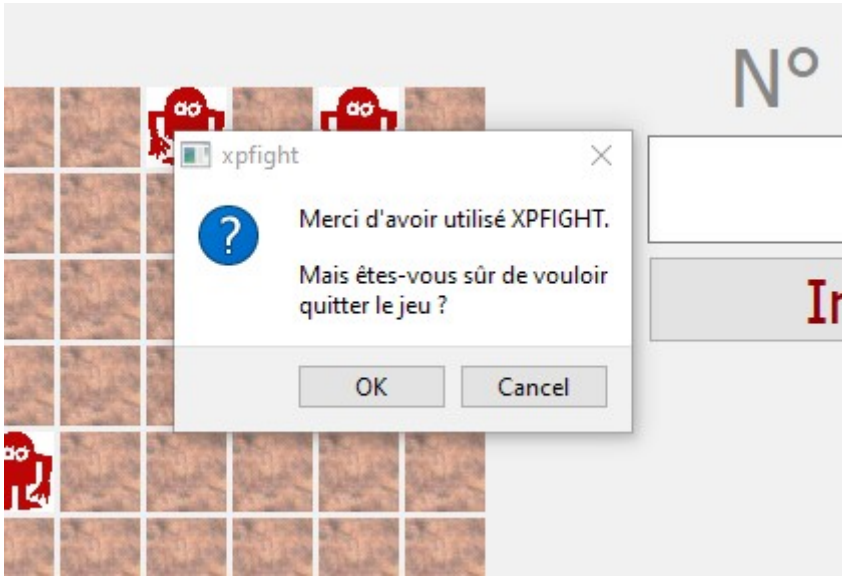
6. Lancement du combat

7. Vérification du score puis enregistrement du score

8. Affichage du menu et ses fonctionnalités



9. Quitter le jeu



10. Affichage des différents thèmes

1) Thème « classique »

2) Thème « comics »

3) Thèmes « lettres »