Lycée TOUCHARD WASHINGTON

Section des Techniciens Supérieurs

Systèmes Numériques option Informatique et Réseaux

académie Nantes

Projet Chasse au trésor multi-joueurs

~ TP Ctrl 3 ~
Codage Qt
Interfaces utilisateurs - Signaux & Slots
Sockets TCP - Flux de données

Version 2.0 – décembre 2020

Conditions de réalisation

Travail individuel

Durée : 4h

Travail à déposer dans votre dépot privé **chasse_au_tresor** de votre GitHub. Vous inviterez votre enseignant en tant que collaborateur.

Après chaque étape, transférez votre réalisation sur votre GitHub en indiquant le nom de l'étape dans le commentaire de votre commit.

Vous rendrez également dans un document README.md une présentation du dépôt .

Ressources utilisées :

Matériel	**Logiciel**
Un ordinateur	Qtcreator Git Doxygen

1. PRÉSENTATION DU PROJET

On se propose de créer un jeu de chasse au trésor multijoueur.
Dans une grille de jeu de 20x20, un trésor est caché.
Chaque joueur est placé de façon aléatoire sur la grille en début du jeu.
Il ne possède comme information que la distance le séparant du trésor.
Les déplacements possibles sont: haut, bas, gauche et droite.
Les bords de la grille sont particuliers :

- Si le joueur est sur le bord gauche et qu'il essaye d'aller à gauche, cela le fera reculer d'une case à droite.
- Si le joueur est sur le bord haut et qu'il essaye d'aller en haut, cela le fera reculer d'une case vers le bas.
- Si le joueur est sur le bord droit et qu'il essaye d'aller à droite, cela l'amènera sur le bord gauche.
- Si le joueur est sur le bord bas et qu'il essaye d'aller en bas, cela l'amènera sur le bord haut.
-

Si le joueur rencontre un autre joueur, les deux joueurs sont renvoyés à des positions aléatoires.
Le jeu se termine lorsqu'un joueur a trouvé le trésor.

2. PROTOCOLE DE COMMUNICATION

Le client envoie toujours des trames ayant le format suivant:

quint16	QChar
taille de la trame	commande

Les commandes possibles sont:

- 'U' : Pour aller vers le haut
- 'D' : Pour aller vers le bas
- 'L' : Pour aller à gauche
- 'R' : Pour aller à droite

Le serveur envoie toujours des trames ayant le format suivant:

quint16	QPoint	QString	double
taille de la trame	coordonnées du joueur sur la grille	message	distance par rapport au trésor

Les messages possibles sont:

- "start" : Trame reçue lors de la connexion
- "vide" : La case sur laquelle vous êtes est vide
- "collision" : Vous avez rencontré un autre joueur
- "Victoire de X.X.X.X" : Un joueur a trouvé le trésor (X.X.X.X est l'adresse du client ayant trouvé le trésor).

En cas de victoire d'un joueur, c'est le point ayant pour coordonnées (-1,-1) qui sera envoyé à tous les joueurs.

3. TRAVAIL DEMANDÉ

Réalisez dans votre GitHub un dépôt privé nommé **Chasse_au_tresor**. Il servira de base pour votre projet sous Qt et votre documentation.

Sous QtCreator, réalisez un projet nommé **Chasse_au_tresor** de type ***Qt Widgets Application***. Il sera suivi par le gestionnaire de version **Git**.

Vous réaliserez au choix soit le client soit le serveur.

Le serveur peut très bien fonctionner sans rien afficher.

L'aspect graphique est donc à traiter en dernier.

4. LE CLIENT

4.1.1.Interface utilisateur

Cette dernière doit afficher la grille de jeu et la position du joueur sur cette dernière.

Avoir une zone permettant de saisir l'adresse ip du serveur ainsi que le numéro de port de ce dernier.

Un bouton de connexion/deconnexion au serveur.

Avoir une rose des vents minimaliste permettant de gérer les déplacements du joueur, uniquement si ce dernier est effectivement connecté au serveur.

Une zone affichant la distance du joueur par rapport au trésor.

Une zone affichant les messages en provenance du serveur.

- En cas de collision le message devra être affiché en magenta.
- En cas de victoire d'un joueur, si le vainqueur est le joueur, le message devra être affiché en vert, sinon il devra s'afficher en rouge.
- Dans les autres cas, le message s'affiche en noir.

Les widgets sont placés sur un QGridLayout, voir en annexe l'extrait de code.

Il est possible de changer la couleur de font d'un bouton de la grille en fonction de sa position.

On suppose que l'on a un objet de type "QGridLayout *grille" et que toutes les cases contiennent un objet de type "QToolButton *".

Pour changer la couleur d'un bouton en (5,8) en noir, on peut faire ainsi:

```
grille->itemAtPosition(8,5)->widget()->setStyleSheet("background-
color : black");
```

Pour avoir l'abscisse et l'ordonnée d'un objet de type QPoint nommé pos:

```
QPoint pos;
int x=pos.x();
int y=pos.y();
```

4.1.2. Gestion des communications

Les communications avec le serveur se font en mode TCP.

Les signaux impérativement associés à un slot sont:

- **connected**
- **disconnected**
- **error**
- **readyRead**

Chaque slot devra afficher un message compréhensible avec qDebug() << this.

Il est fortement recommandé de coder une méthode :

```
void EnvoyerCommande(QChar commande);
```

Cette méthode sera en charge de formater la trame et d'envoyer cette dernière via la socket.

La classe visuelle devra s'appeler : **ClientCrawler**

5. LE SERVEUR

5.1.1. Interface

Elle ne contient que la grille (générée dynamiquement), un bouton quitter, une zone permettant de saisir le numéro de port et un bouton lancement.

La grille affichera:

- Le trésor en rouge.
- Les clients en noir.

La grille sera mise à jour :

- Au lancement du serveur.
- A chaque connexion de client
- A chaque réception de commande de la part d'un client.
- A chaque déconnexion de client.

La classe visuelle devra s'appeler : **ServeurCrawler**

5.1.2. Gestion des communications

Les communications avec le serveur se font en mode TCP.

Le signal impérativement associé à un slot pour la socket d'écoute serveur est:

- `newConnection`

Les signaux impérativement associés à un slot pour les sockets de communication client sont:

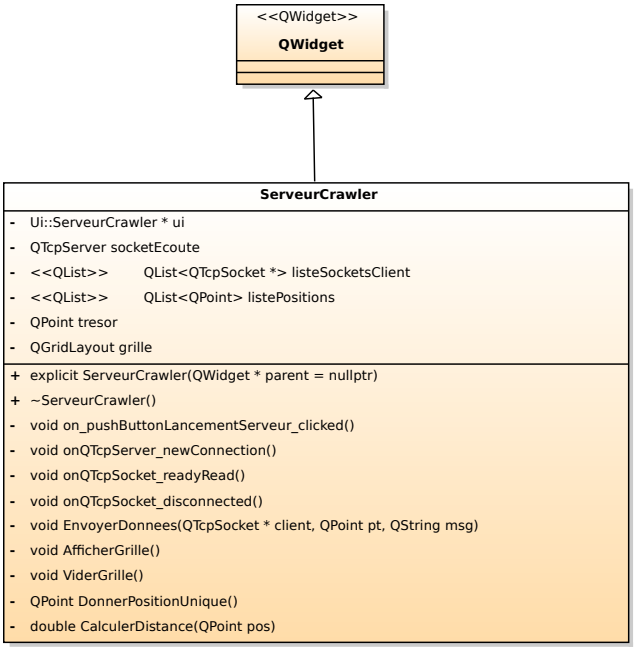
- `disconnected`
- `error`
- `readyRead`

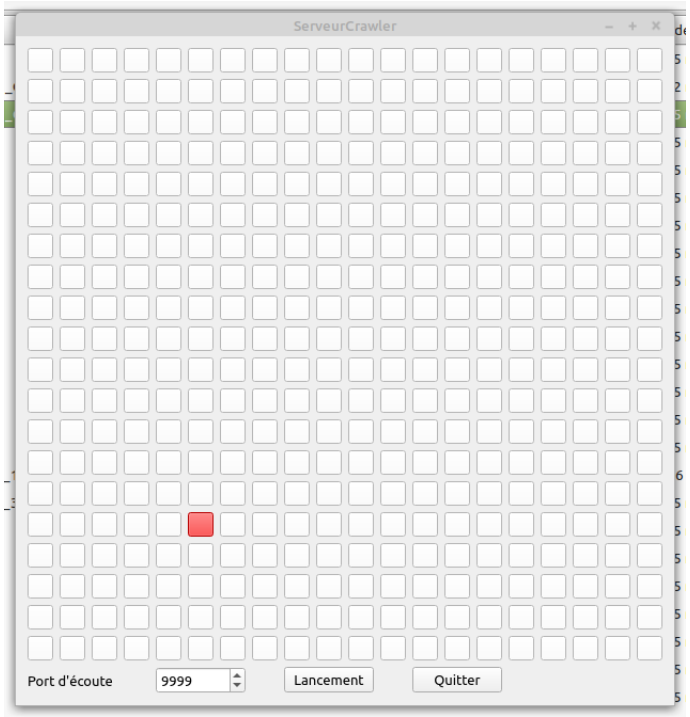
Lors d'une demande de connexion en provenance d'un client:

1. La socket de communication de ce dernier est ajoutée à une liste.
2. Une position unique est ajoutée à une liste de QPoint.

3. Une trame, contenant ce point de départ et le message "start", est envoyée au client.
4. La grille est mise à jour.

Vous pouvez vous baser sur le diagramme de classe et le visuel suivant:





Les méthodes **DonnerPositionUnique** et **CalculerDistance** sont fournies:

```
void qtPause(int millisecondes)
{
    QTimer timer;
    timer.start(millisecondes);
    QEventLoop loop;
    QObject::connect(&timer,&QTimer::timeout,&loop,&QEventLoop::quit);
    loop.exec();
}
```

```
QPoint ServeurCrawler::DonnerPositionUnique()
{
    QRandomGenerator gen;
    QPoint pt;
    gen.seed(QDateTime::currentMSecsSinceEpoch());
    int ligne;
    int colonne;
    do
    {
        ligne = gen.bounded(TAILLE);
        qtPause(20);
        colonne = gen.bounded(TAILLE);
        qtPause(20);
        pt = QPoint(colonne,ligne);
    }while (listePositions.contains(pt));

    return pt;
}

double ServeurCrawler::CalculerDistance(QPoint pos)
{
    double distance;
    int xVecteur = tresor.x()-pos.x();
    int yVecteur = tresor.y()-pos.y();

    distance = sqrt((xVecteur*xVecteur + yVecteur*yVecteur));
    return distance;
}
```

L'utilisation d'une méthode **EnvoyerDonnees** est fortement recommandée.

Elle jouera le même rôle que celle du client, mais avec des paramètres d'entrée différents.

Lors de la réception d'une commande en provenance d'un client il faudra:

1. Retrouver l'index de la position du client.
2. En fonction de la commande, modifier x ou y.

- Si la nouvelle position est celle d'un autre joueur
  - Retrouver l'index de l'autre joueur
  - Remplacer la position du joueur et de l'autre joueur par une nouvelle position uniquement
  - Envoyer une trame avec le message "collision" aux deux joueurs.
- Si la position est celle du trésor,
  - Envoyer à tous les joueurs le message "Victoire de AdrDuGagnant" et la position (-1,-1)
  - Deconnecter tous les joueurs
  - donner une nouvelle position au trésor
- S'il n'y a ni collision, ni découverte du trésor
  - Remplacer la position du joueur avec sa nouvelle position dans la liste des positions
  - Envoyer un message avec la nouvelle position et le message "vide"

ANNEXES

Méthodes utiles concernant les QList.

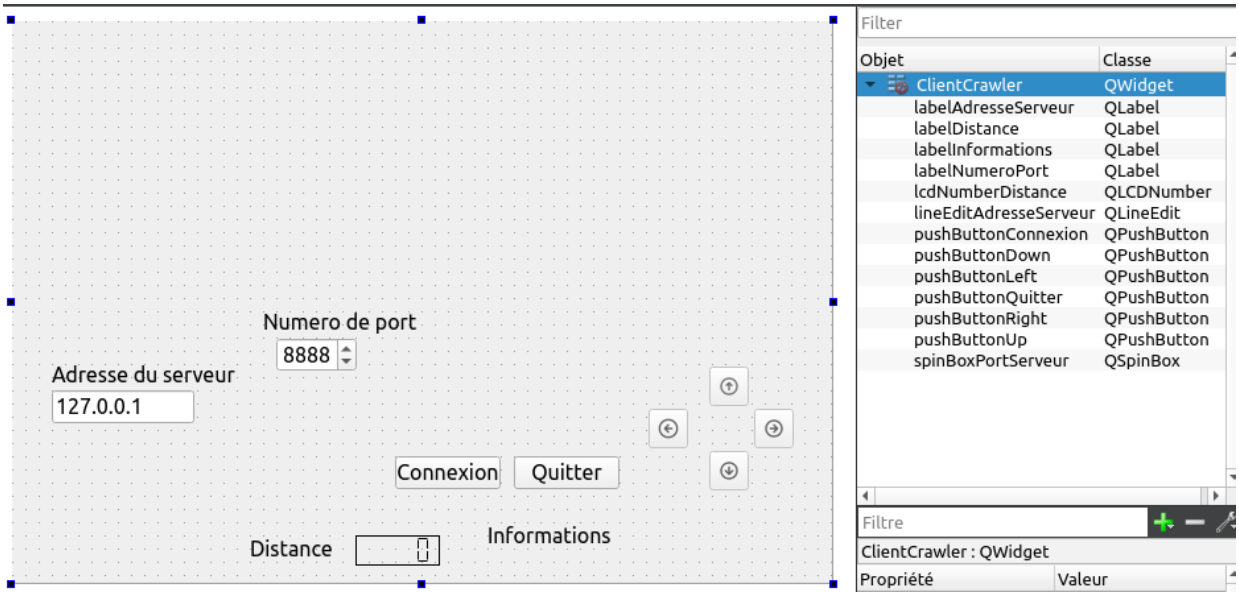
On suppose:

```
QList <typeobj>myList;
int index;
typeobj obj1 ;
typeobj obj2 ;

myLst.replace(index,obj2);    remplacer l'objet présent à la position "index" par obj2
myLst.removeAt(index);        enlever l'objet présent à la position "index" de la liste
myLst.indexOf(obj1);          avoir la position de l'objet "obj1" dans la liste lst
myLst.at(index);       accéder à l'objet présent à la position "index" de la liste
myLst.contains(obj2);  retourne vrai si la liste "lst" contient l'objet "obj2"
```

Extrait de code pour la construction de la grille du client

```
for(int ligne=0; ligne<TAILLE; ligne++)
 {
     for (int colonne=0; colonne<TAILLE; colonne++)
     {
         QToolButton *b=new QToolButton();
         grille->addWidget(b,ligne,colonne,1,1);
     }
 }
 // Placement sur la grille des objets présents sur ui
 // les labels
 grille->addWidget(ui->labelAdresseServeur,TAILLE,0,1,5);
 grille->addWidget(ui->labelNumeroPort,TAILLE,6,1,5);
 grille->addWidget(ui->labelDistance,TAILLE+3,0,1,5);
 grille->addWidget(ui->labelInformations,TAILLE+3,12,1,5);
 // les QLineEdit adresse et port
 grille->addWidget(ui->lineEditAdresseServeur,TAILLE+1,0,1,5);
 grille->addWidget(ui->spinBoxPortServeur,TAILLE+1,6,1,5);
 // les QPushButton connexion et quitter
 grille->addWidget(ui->pushButtonConnexion,TAILLE+2,0,1,5);
 grille->addWidget(ui->pushButtonQuitter,TAILLE+2,6,1,5);
 // les QPushButton fleches
 grille->addWidget(ui->pushButtonUp,TAILLE,15,1,1);
 grille->addWidget(ui->pushButtonLeft,TAILLE+1,14,1,1);
 grille->addWidget(ui->pushButtonRight,TAILLE+1,16,1,1);
 grille->addWidget(ui->pushButtonDown,TAILLE+2,15,1,1);
 //distance et informations
 grille->addWidget(ui->lcdNumberDistance,TAILLE+3,6,1,5);
 this->setLayout(grille);
```


Numero de port
8888
Adresse du serveur
127.0.0.1
Connexion
Quitter
Distance
Informations
Filter
Objet
Classe
ClientCrawler
QWidget
labelAdresseServeur
QLabel
labelDistance
QLabel
labelInformations
QLabel
labelNumeroPort
QLabel
lcdNumberDistance
QLCDNumber
lineEditAdresseServeur
QLineEdit
pushButtonConnexion
QPushButton
pushButtonDown
QPushButton
pushButtonLeft
QPushButton
pushButtonQuitter
QPushButton
pushButtonRight
QPushButton
pushButtonUp
QPushButton
spinBoxPortServeur
QSpinBox
Filtre
ClientCrawler : QWidget
Propriété
Valeur