# Rapport - Projet langages avancés : Jeu de tanks

Younes Ben Yamna - Malek Zemni

8 janvier 2017

# 1 Introduction

Ce jeu de tanks est basé sur le principe de la programmation orientée objet. Il est écrit dans le langage C++ et utilise la bibliothèque Qt pour la gestion des fonctionnalités graphiques ainsi que la gestion des tests unitaires.

## 1.1 Installation du jeu :

Se positionner sur le répertoire du projet `Tanks` puis :

**Compilation du jeu :** taper la commande `make`.

**Lancement du jeu :** taper la commande `make run`.

## 1.2 Installation des tests :

Se positionner sur le répertoire de tests `Tanks/tests` puis :

**Compilation des tests :** taper la commande `make`.

**Lancement des tests :** taper la commande `make run`.

# 2 Explication du jeu

## 2.1 Déroulement du jeu

Lors du lancement du jeu, un menu principal s'affiche. Il est alors possible de choisir un jeu à 2, 3 ou 4 joueurs (humain contre humain) ou un jeu à 1 joueur (humain contre IA). Dans un jeu d'humain contre humain, chaque joueur doit attendre son tour pour pouvoir jour. Le joueur n°1 est positionné à gauche, le joueur n°2 à droite, le joueur n°3 en bas et le joueur n°4 en haut de l'écran.
Dans un jeu d'humain contre une IA, l'humain est positionné à gauche de l'écran et l'IA à droite.

Le menu du jeu (à droite de la fenêtre) affiche le numéro du joueur qui joue actuellement ainsi que des informations sur ce dernier. Ces informations sont :

— La capacité de déplacement du tank
— La résistance du tank (le points de vie)
— Les informations de chacun des 3 obus : la force, le rayon et le nombre d'obus restants

Le menu du jeu permet aussi de tirer un obus et de revenir au menu principal (quitter).

**Tour de jeu :** à chaque tour de jeu, le joueur (tank) peut se déplacer (tant qu'il lui reste des points de capacité), positionner son canon verticalement et horizontalement et enfin tirer (s'il lui reste des obus). Une fois que le joueur tire un obus (bouton du menu), le jeu passe au tour du joueur suivant.

**Angle de tir :** la direction de tir de l'obus est la même que la direction du canon (position horizontale du canon). La portée du tir quant à elle dépend de le position verticale du canon : celle-ci suit une direction parabolique, c'est à dire que la portée de tir est maximale pour un angle du canon compris entre 30° et 40°.

**Destruction d'un élément (tank ou obstacle) :** un élément est touché par un obus si l'impact de ce dernier (rayon de l'obus) touche l'élément. Il lui enlève des points de sa résistance. L'élément est détruit si sa résistance est inférieure ou égale à 0.

**Fin du jeu :** le jeu se termine s'il ne reste qu'un seul tank dans le terrain (tous les autres sont détruits).

## 2.2 Commandes du jeu

— **Déplacement du tank :** flèches du clavier
— **Positionnement du canon :** touches Q et D pour l'angle horizontal et Z et S pour l'angle vertical
— **Tir :** bouton du menu du jeu

# 3 Explication du code

## 3.1 Le fichier principal

Le fichier principal qui contient la fonction `main()` se contente de créer un objet `Menu` et de l'afficher. C'est cet objet `Menu` qui va ensuite lancer le jeu en appelant le slot `lancerJeu()` en fonction du nombre de joueurs choisi.

## 3.2 La classe `Menu`

La classe menu affiche plusieurs boutons qui sont gérés par la classe `ButtonWidget`. Cette classe permet de passer le texte du bouton en paramètre.

```
//Utilisation de la classe ButtonWidget pour permettre de passer le texte du bouton en
    parametre au slot
QObject::connect(boutons, SIGNAL(clicked(QString)), this, SLOT(lancerJeu(QString)));
```

Menu.cpp

Le slot `lancerJeu()` pourra ensuite créer l'objet `Jeu` en fonction du texte (passé en paramètre au slot) du bouton cliqué.

```
void Menu::lancerJeu(QString nbJoueurs)
{
    J = new Jeu(nbJoueurs[0].digitValue(),this);
    J->show();
}
```

Menu.cpp

## 3.3 La classe `Jeu`

La classe `Jeu` est la classe fondamentale qui va afficher les éléments du jeu puis gérer les événements.

### 3.3.1 Affichage des éléments

Le constructeur de la classe `Jeu` va créer un terrain (classe `Terrain`) et y rajouter les éléments du jeu (tanks et obstacles). La classe `Jeu` est un `QGraphicsView` qui va afficher le terrain (qui est un `QGraphicsScene` ). C'est donc le `Terrain` qui contient les éléments (tanks et obstacle) et c'est le `Jeu` qui les affiche.

### 3.3.2 Gestion des événements

La classe `Jeu` contient un slot `tourDeJeu()` qui va être appelé à chaque nouveau tour. Le slot ou la méthode `tourDeJeu()` commence par afficher le menu correspondant au joueur courant et les boutons des obus. On voit ici le principe de la **programmation événementielle** : on n'attend pas que le tank finisse son positionnement pour qu'il puisse tirer, il peut se déplacer et positionner son canon jusqu'à ce qu'un événement se produise : un clic sur un bouton de tir d'obus. Exemple pour l'obus 1 :

```
tirObus1 = new ButtonWidget(QStringList("O1 - TIRER"),QColor(Qt::black));
if (tanks[aQuiLeTour]->getNbObus1()>0){
    QObject::connect(tirObus1, SIGNAL(clicked(QString)), tanks[aQuiLeTour], SLOT(tirer(
    QString)));
    QObject::connect(tirObus1, SIGNAL(clicked(QString)), this, SLOT(waitTir()));
} else
    tirObus1->setDisabled(true);
```

Jeu.cpp

Vers la fin, le slot `tourDeJeu()` va mettre à jour la variable globale `aQuiLeTour` tout en prenant compte les joueurs morts. Si tous les joueurs sont morts, le jeu sera terminé et un message de fin apparaîtra.

```
//Passage au prochain joueur (s'il n'est pas mort)
int prochain = (aQuiLeTour+1)%nombreJoueurs;
```

```
while (estMort(prochain+1))      //le num de joueur est augmenté de 1 (tour 0 pour le
    joueur 1)
    prochain = (prochain+1)%nombreJoueurs;

if (prochain == aQuiLeTour)
    emit jeuFini();
else
    aQuiLeTour = prochain;
```

Jeu.cpp

### 3.3.3 Tour de jeu de l'IA

Il existe aussi dans la classe `Jeu` une méthode `tourDeJeuIA()` similaire à `tourDeJeu()` qui simule le comportement d'un joueur (intélligence artificielle). Ici il n'y a pas d'affichage de menu, mais simplement un cas particulier de la méthode `keyPressedEvent()` de la classe `Tank` (qui gère les événements liés au tank et son canon).

## 3.4 La classe `Tank`

La classe `Tank` permet de construire un tank et gérer les événements liés à son déplacement et le positionnement de son canon et aussi au tir.

### 3.4.1 Construction

Un tank est une image `QGraphicsPixmapItem` qui est aussi un élément d'un `QGraphicsScene`, c'est à dire le terrain. Il en est de même pour le canon (et les obstacles aussi). Le constructeur de cette classe va donc créer le tank et le canon (qui héritera du tank pour avoir un déplacement homologue).

### 3.4.2 Gestion des événements du clavier

La classe `Jeu` contient une méthode `keyPressedEvent()` qui va gérer les événement liés aux touches du clavier (touches pour le déplacement du tank et le positionnement du canon). Si une touche de déplacement est appuyée, des tests seront effectués pour vérifier si le déplacement souhaité est possible puis déplace le tank (et met à jour les propriétés du tank et les cases du terrain), sinon rien ne se passe. Exemple pour la touche flèche gauche :

```
if (event->key() == Qt::Key_Left && capaciteDeplacement>0){
    if (direction == HAUT){
        setRotation(rotation()-90);
        canon->setAngleHAbsolu(canon->getAngleHAbsolu()+90);
    }
    else if (direction == BAS){
        setRotation(rotation()+90);
        canon->setAngleHAbsolu(canon->getAngleHAbsolu()-90);
    }
    else if (direction == DROITE){
        setRotation(rotation()+180);
        canon->setAngleHAbsolu(canon->getAngleHAbsolu()-180);
    }
    else if
```

```
    //Si la case d'arrivée est accessible
    (getPosition().getX()-uniteDeplacement >= 0
    && (
        terrain->getCases((getPosition().getX()-uniteDeplacement)/tailleCase,getPosition()
    .getY()/tailleCase) == VIDE
        || (numJoueur == 1 &&
        terrain->getCases((getPosition().getX()-uniteDeplacement)/tailleCase,getPosition()
    .getY()/tailleCase) == TANK1
        ) || (numJoueur == 2 &&
        terrain->getCases((getPosition().getX()-uniteDeplacement)/tailleCase,getPosition()
    .getY()/tailleCase) == TANK2
        ) || (numJoueur == 3 &&
        terrain->getCases((getPosition().getX()-uniteDeplacement)/tailleCase,getPosition()
    .getY()/tailleCase) == TANK3
        ) || (numJoueur == 4 &&
        terrain->getCases((getPosition().getX()-uniteDeplacement)/tailleCase,getPosition()
    .getY()/tailleCase) == TANK4
        )
    ) && (
        terrain->getCases((getPosition().getX()-uniteDeplacement)/tailleCase,(getPosition
    ().getY()+2*getRayon())/tailleCase) == VIDE
        || (numJoueur == 1 &&
        terrain->getCases((getPosition().getX()-uniteDeplacement)/tailleCase,(getPosition
    ().getY()+2*getRayon())/tailleCase) == TANK1
        ) || (numJoueur == 2 &&
        terrain->getCases((getPosition().getX()-uniteDeplacement)/tailleCase,(getPosition
    ().getY()+2*getRayon())/tailleCase) == TANK2
        ) || (numJoueur == 3 &&
        terrain->getCases((getPosition().getX()-uniteDeplacement)/tailleCase,(getPosition
    ().getY()+2*getRayon())/tailleCase) == TANK3
        ) || (numJoueur == 4 &&
        terrain->getCases((getPosition().getX()-uniteDeplacement)/tailleCase,(getPosition
    ().getY()+2*getRayon())/tailleCase) == TANK4
        )
    ) )
    {
        setPos(x()-uniteDeplacement,y());
        capaciteDeplacement--;
        //Si le tank traverse une crevasse, sa capacité de déplacement est encore diminuée
        if (traverseCrevasse())
            capaciteDeplacement--;
        emit capaciteChanged(capaciteDeplacement);
    }
    direction = GAUCHE;
}
```

Tank.cpp

Le déplacement du canon est quant à lui sans conditions. On se contente juste de mettre à jour les angles de ce dernier pour qu'ils puissent guider la trajectoire du tir (l'obus).

### 3.4.3 La méthode `tirer()`

Cette méthode (ou slot) est appelée lors de l'appui sur un bouton de tir du menu de jeu. Elle crée un objet `Obus` et l'ajoute au terrain.

```
void Tank::tirer(QString typeObus){
    int type = typeObus[1].digitValue();
    switch (type) {
        case 1: nbObus1--; break;
        case 2: nbObus2--; break;
        case 3: nbObus3--; break;
    }
    Obus* ob = new Obus(type,canon->getAngleHAbsolu(), canon->getAngleVAbsolu());
    ob->setPos(x()+getRayon()-uniteDeplacement,y()+getRayon()-uniteDeplacement);
    terrain->addItem(ob);
}
```

Tank.cpp

## 3.5 La classe `Obus`

La classe `Obus` permet de construire un obus (à la position et à la direction passés en paramètre) et gérer les événements liés à son mouvement et son impact.

### 3.5.1 Construction

Lors de l'appel à la méthode `tirer()` de la classe `Tank`, un obus va être crée. Celui-ci va prendre une position initiale qui ne dépend que de la position horizontale du canon du tank. Il s'agit de l'attribut `angleH`. Cet attribut va ensuit guider la trajectoire de déplacement du tank grâce aux fonctions trigonométriques.

```
setPos(x()+uniteDeplacement*cos(angleH*M_PI/180.0),y()-uniteDeplacement*sin(angleH*M_PI
    /180.0));
compteurPosObus+= uniteDeplacement;
```

Obus.cpp

La classe `Obus` a aussi un attribut `angleV` qui permet de calculer la position de l'impact de l'obus grâce à une équation parabolique.

```
distanceImpact = -0.148*aV*aV + 10.0*aV + 300;
```

Obus.cpp

### 3.5.2 Mouvement

Le mouvement de l'obus va être géré dans la méthode `bouger()` qui sera appelée toutes les 10 millisecondes.

```
QTimer* timer = new QTimer(this);
connect(timer,SIGNAL(timeout()),this,SLOT(bouger()));
timer->start(10);
```

Obus.cpp

À chaque appel à la méthode `bouger()`, on va incrémenter le compteur de déplacement `compteurPosObus` jusqu'à ce qu'il atteigne une valeur correspondant à la distance de l'impact `distanceImpact`. Dans ce cas, on fait exploser l'obus et on regarde les éléments qui sont touchés par l'impact de l'obus (rayon de l'obus) puis

on effectue les actions adéquates : si l'impact touche un tank ou un obstacle, on va réduire leur résistance, si le tank ou l'obstacle touché n'a plus de résistance, on le détruit.

```
//Si la crevasse touche un objet, on traite l'impact
for (int i = 0, n = elementsTouches.size(); i < n; ++i)
{
    //Si c'est un tank
    if (typeid(*(elementsTouches[i])) == typeid(Tank))
    {
        cout << "Tank touche" << endl;
        (dynamic_cast<Tank*>(elementsTouches[i]))->decrementeResistance(force);
        if ( (dynamic_cast<Tank*>(elementsTouches[i]))->getResistance() <= 0 ){
            J->getTerrain()->removeItem(elementsTouches[i]);
            J->setJoueurMort((dynamic_cast<Tank*>(elementsTouches[i]))->getNumJoueur());
            switch ((dynamic_cast<Tank*>(elementsTouches[i]))->getNumJoueur()) {
                case 1: J->getTerrain()->vider(TANK1);  break;
                case 2: J->getTerrain()->vider(TANK2);  break;
                case 3: J->getTerrain()->vider(TANK3);  break;
                case 4: J->getTerrain()->vider(TANK4);  break;
            }
            delete elementsTouches[i];
        }
    }
    //Si c'est un obstacle
    if (typeid(*(elementsTouches[i])) == typeid(Obstacle))
    {
        cout << "Obstacle touche" << endl;
        (dynamic_cast<Obstacle*>(elementsTouches[i]))->decrementeResistance(force);
        if ( (dynamic_cast<Obstacle*>(elementsTouches[i]))->getResistance() <= 0 ){
            J->getTerrain()->updateCases(Point(elementsTouches[i]->pos().x()+tailleCase/2,
    elementsTouches[i]->pos().y()+tailleCase/2),VIDE);
            J->getTerrain()->removeItem(elementsTouches[i]);
            delete elementsTouches[i];
        }
    }
}
```

Obus.cpp

# 4 Conclusion

Ce jeu écrit en C++ a permis de mettre en œuvre plusieurs notions liées à la programmation orientée objet. La bibliothèque Qt sur laquelle il est basé a été non seulement une concrétisation des vertus de la programmation objet mais aussi un lien entre ce principe de programmation et la programmation événementielle.