

Exemplo 6: Jogo de nave

Tutorial baseado nos exemplos do C++ Qt Game Tutorial

<https://www.youtube.com/watch?v=8ntEQpg7gck&t=189s>

Para criar esse projeto vamos utilizar a classe [QGraphicsScene](#)

Essa classe serve como *container* para armazenar os componentes gráficos **QGraphicsItems**. Junto com a classe [QGraphicsView](#) podemos visualizar componentes gráficos variados, como linhas, retângulos, texto ou itens personalizados.

[QGraphicsScene](#) permite determinar a localização na tela dos objetos e a visibilidade deles.

Esse componente não tem nenhuma aparência própria, ele somente administra objetos gráficos.

Essa classe pode ser considerada como o “Mundo” do jogo, onde vão ser alocados o próprio jogador e outros objetos.

Para criar os elementos gráficos dentro do jogo vamos usar a classe [QGraphicsItem](#). Pois somente essa classe e seus herdeiros podem ser alocados dentro do [QGraphicsScene](#).

A elaboração do projeto vai ser feita em passos, que acrescentam cada vez mais funcionalidade ao jogo.

1. Criação de Scene (pode ser traduzido como cenário)

Nesse primeiro passo vamos representar o jogador com um retângulo.

Vamos criar um novo projeto:

File > New File or Project > Application > Qt Widgets Application > Choose

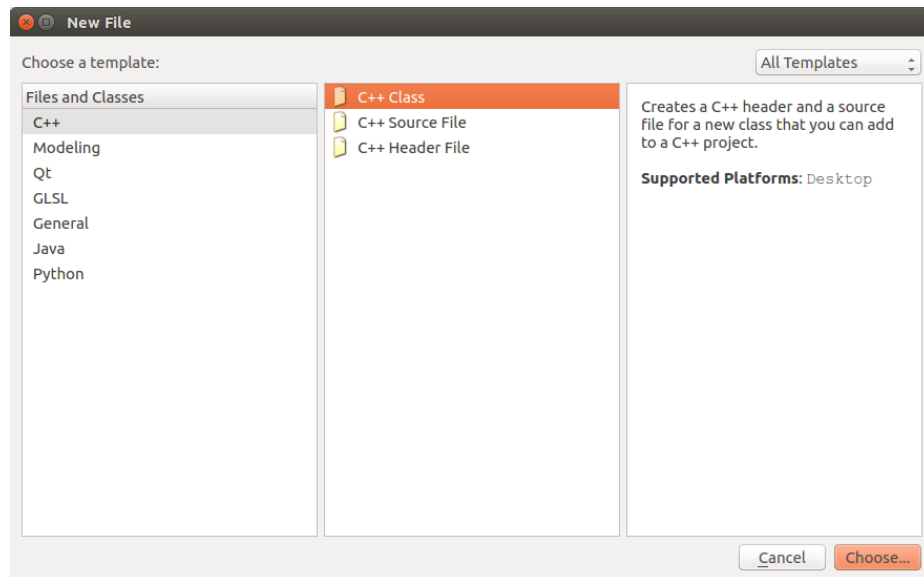
Nome do projeto: **MyGame**

As classes serão redefinidas manualmente, por isso podemos deixar o resto como padrão.

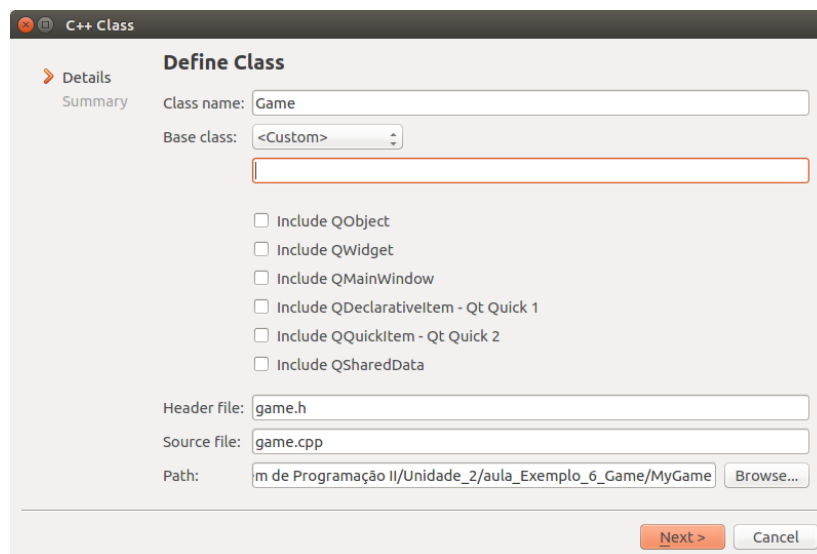
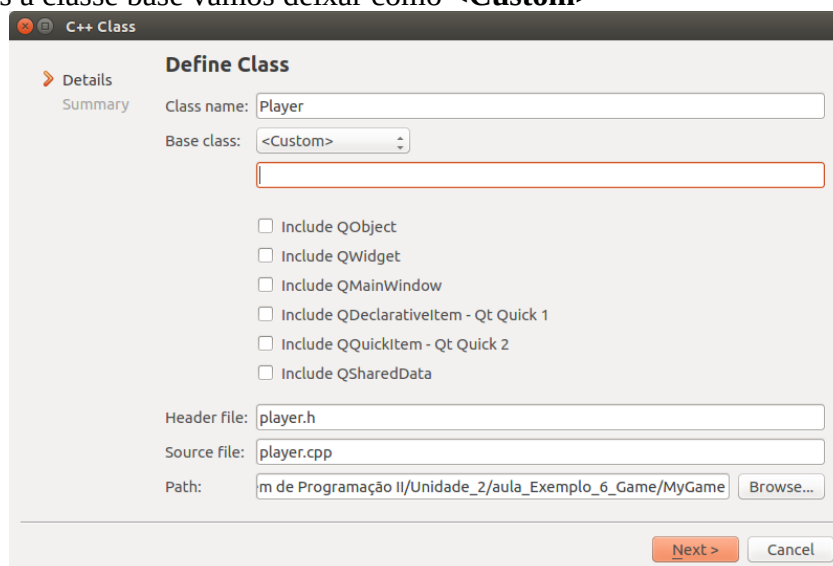
a) Agora temos que apagar todos os arquivos relacionados ao **MainWindow**, deixando somente os arquivos

MyGame.pro
main.cpp

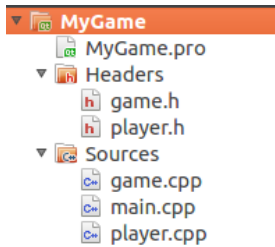
b) Agora vamos adicionar duas classes chamando o menu **AddNew** em cima do nome do projeto, e depois escolhendo **C++ → C++ Class**



A primeira classe vai se chamar **Player** e a segunda vai se chamar **Game**.
Em ambos os casos a classe base vamos deixar como **<Custom>**



Depois disso o nosso projeto vai ter a seguinte estrutura:



c) Temos que remover tudo que se refere a Classe **Mainwindow** do arquivo **main.cpp** deixando somente

main.cpp

```
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    return a.exec();
}
```

No primeiro momento vamos representar o jogador como um retângulo **QGraphicsRectItem**, que é a classe derivada da [QGraphicsItem](#).

Agora vamos adicionar o código:

main.cpp

```
#include <QApplication>
#include "game.h"

Game * game; // variavel global, que pode ser acessada
//por todos os arquivos do projeto

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    // criar novo jogo
    game = new Game();
    game->show();

    return a.exec();
}
```

game.h

```
#ifndef GAME_H
#define GAME_H

#include <QGraphicsView>
#include <QWidget>
#include <QGraphicsScene>

#include "player.h"

class Game: public QGraphicsView
{
```

```

public:
    Game(QWidget * parent=0);

    QGraphicsScene * scene;
    Player * player;
};

#endif // GAME_H

```

game.cpp

```

#include <QGraphicsTextItem>
#include "game.h"

Game::Game(QWidget *parent)
{
    // criar scene
    scene = new QGraphicsScene();

    // fixar o tamanho em 800x600, que é infinito por definição
    scene->setSceneRect(0,0,800,600);

    // visualizar o objeto scene (cenario)
    setScene(scene);

    //desabilitar as barras de rolagem
    setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    // fixar o tamanho
    setFixedSize(800,600);

    // criar o jogador
    player = new Player();
    // definir o tamanho do jogador 100 x 100
    player->setRect(0,0,100,100);
    // definir a posição padrão do jogador para ser em baixo da tela
    // por definição a visualização é centralizada para pegar todos os objetos
    player->setPos(400,500);
    // colocar o foco no jogador
    player->setFlag(QGraphicsItem::ItemIsFocusable);
    player->setFocus();
    // adicionar o jogador no cenário
    scene->addItem(player);

    show();
}

```

player.h

```

#ifndef PLAYER_H
#define PLAYER_H

#include <QGraphicsRectItem>
#include <QObject>
#include <QGraphicsItem>

```

```
class Player: public QObject, public QGraphicsRectItem
{
    Q_OBJECT
public:
    Player(QGraphicsItem * parent=0);

};

#endif // PLAYER_H
```

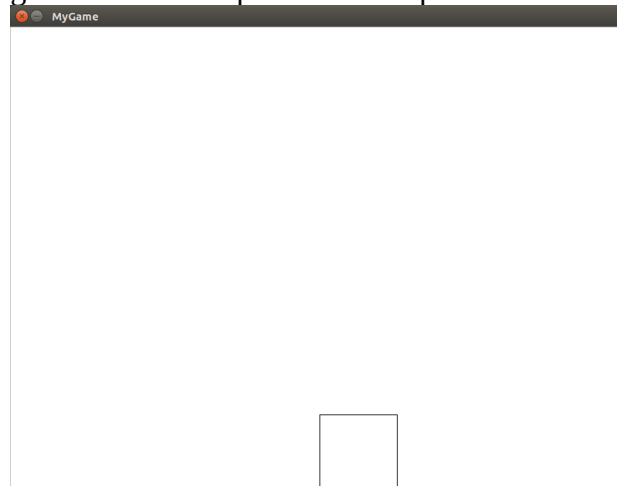
player.cpp

```
#include "player.h"
#include <QGraphicsScene>
#include <QObject>
#include <QGraphicsItem>

Player::Player(QGraphicsItem *parent): QGraphicsRectItem(parent)
{

}
```

Agora nosso protótipo de jogo deve ter essa aparência: um quadrado em baixo da tela branca



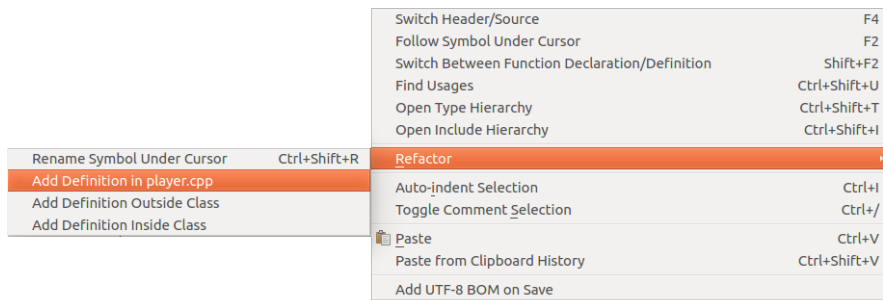
2. Movimentação do jogador

Vamos implementar a movimentação do jogador pela tela em resposta ao teclado, ou seja vamos implementar o processamento de eventos.

Observação: QT fornece algumas opções bastante úteis quando se chama menu em cima de um método em arquivo **.h**

Switch Header/Source	F4
Follow Symbol Under Cursor	F2
Switch Between Function Declaration/Definition	Shift+F2
Find Usages	Ctrl+Shift+U
Open Type Hierarchy	Ctrl+Shift+T
Open Include Hierarchy	Ctrl+Shift+I
Refactor	→
Auto-indent Selection	Ctrl+I
Toggle Comment Selection	Ctrl+/
Paste	Ctrl+V
Paste from Clipboard History	Ctrl+Shift+V
Add UTF-8 BOM on Save	

Esse menu permite pular automaticamente para parte da implementação ou ainda criar o cabeçalho para implementação de forma automática usando a opção **Refactor**.



Vamos adicionar o código:

player.h

```
#ifndef PLAYER_H
#define PLAYER_H

#include <QGraphicsRectItem>
#include <QObject>
#include <QGraphicsItem>

class Player: public QObject, public QGraphicsRectItem
{
    Q_OBJECT
public:
    Player(QGraphicsItem * parent=0);
    void keyPressEvent(QKeyEvent * event);
};

#endif // PLAYER_H
```

Primeiramente vamos testar a resposta do teclado usando <QDebug>

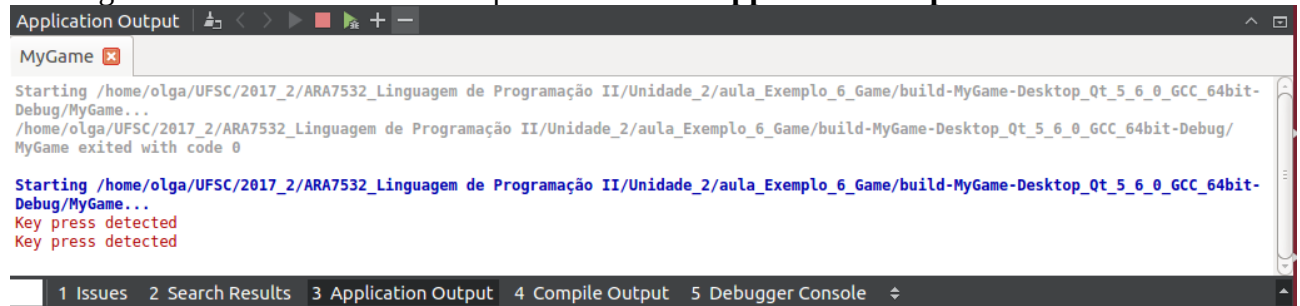
player.cpp

```
#include "player.h"
#include <QGraphicsScene>
#include <QObject>
#include <QGraphicsItem>
#include <QDebug>

Player::Player(QGraphicsItem *parent): QGraphicsRectItem(parent)
{
}

void Player::keyPressEvent(QKeyEvent *event)
{
    //mensagem informando que ocorreu uso do teclado
    qDebug()<<"Key press detected";
}
```

A mensagem sobre uso do teclado vai aparecer na aba **3 Application Output** em baixo da tela



Agora vamos implementar a resposta as teclas de movimentação do teclado com a restrição de movimentos no sentido direita e esquerda impedindo com que o jogador saia para fora da tela.

A movimentação para cima e para baixo fica livre.

Em um cenário (Scene) o canto esquerdo superior é considerado como (0,0) no eixo de coordenados, sendo que os valores de x e de y aumentam na direção para direita e para baixo respectivamente.

player.cpp

```
#include "player.h"
#include <QGraphicsScene>
#include <QObject>
#include <QGraphicsItem>
#include <QDebug>
#include <QKeyEvent>

Player::Player(QGraphicsItem *parent): QGraphicsRectItem(parent)
{
}

void Player::keyPressEvent(QKeyEvent *event)
{
    // mensagem informando que ocorreu uso do teclado
    //qDebug()<<"Key press detected";
    if(event->key() == Qt::Key_Left)
    {
        // limite para não sair da tela
        if(pos().x() > 0)
            setPos(x()-10,y());
    }
    else if(event->key() == Qt::Key_Right)
    {
        // limite para não sair da tela
        if(pos().x() + 100 < 800 )
            setPos(x()+10,y());
    }
    else if(event->key() == Qt::Key_Up)
    {
        setPos(x(),y()-10);
    }
    else if(event->key() == Qt::Key_Down)
    {
        setPos(x(),y()+10);
    }
}
```

3. Adicionar tiro e inimigos no projeto

Como primeiro passo vamos representar tanto tiro como inimigos por mesmo tipo de objeto que o jogador, ou seja o retângulo **QGraphicsRectItem**.

Vamos adicionar no nosso projeto uma classe definida por usuário chamada **Bullet** para representar o tiro.

Para implementar a funcionalidade de tiro temos que usar o mecanismo **Signal and Slots** que permite gerar a resposta para certos eventos.

bullet.h

```
#ifndef BULLET_H
#define BULLET_H

#include <QGraphicsRectItem>
#include <QObject>

class Bullet: public QObject, public QGraphicsRectItem
{
    Q_OBJECT
public:
    Bullet(QGraphicsItem * parent=0);

public slots:
    void move();
};

#endif // BULLET_H
```

bullet.cpp

```
#include <QGraphicsRectItem>
#include <QTimer>
#include <QDebug>

#include "bullet.h"
#include "game.h"

// declaração da variavel do tipo extern
// indica que essa variavel já foi declarada em algum outro arquivo do projeto
extern Game * game;

Bullet::Bullet(QGraphicsItem *parent): QObject(), QGraphicsRectItem(parent)
{
    // desenhar o tiro
    setRect(0,0,10,50);

    // conectar o timer com o movimento
    QTimer * timer = new QTimer();
    connect(timer,SIGNAL(timeout()),this,SLOT(move()));

    // inicializar o timer
    timer->start(50);
}

void Bullet::move()
{
    // movimentação do tiro
    setPos(x(),y()-10);

    if(pos().y() < -10)
    {

```



```

        scene()->removeItem(this);
        delete this;
        qDebug()<<"Bullet removed";
    }
}

```

O tiro vai ser criado dentro do arquivo **player.cpp** em resposta a ação do usuário

player.cpp

```

#include <QGraphicsScene>
#include <QObject>
#include <QGraphicsItem>
#include <QDebug>
#include <QKeyEvent>

#include "player.h"
#include "bullet.h"

Player::Player(QGraphicsItem *parent): QGraphicsRectItem(parent)
{

}

void Player::keyPressEvent(QKeyEvent *event)
{
    // mensagem informando que ocorreu uso do teclado
    //qDebug()<<"Key press detected";
    if(event->key() == Qt::Key_Left)
    {
        // limite para não sair da tela
        if(pos().x() > 0)
            setPos(x()-10,y());
    }
    else if(event->key() == Qt::Key_Right)
    {
        // limite para não sair da tela
        if(pos().x() + 100 < 800 )
            setPos(x()+10,y());
    }
    else if(event->key() == Qt::Key_Up)
    {
        setPos(x(),y()-10);
    }
    else if(event->key() == Qt::Key_Down)
    {
        setPos(x(),y()+10);
    }

    // criar tiro (bullet)
    else if(event->key() == Qt::Key_Space)
    {
        // teste via Debug
        // qDebug()<<"Bullet created";

        Bullet * bullet = new Bullet();
        bullet->setPos(x(),y());
        scene()->addItem(bullet);
    }
}

```

Observação: Caso apareçam mensagens de erro na compilação depois de adicionar novos arquivos no projeto a dica é ir no menu em cima e escolher **Build** → **Clean All** e depois executar **Build** → **Run qmake**

Vamos adicionar no nosso projeto uma classe definida por usuário chamada a classe **Enemy** para representar o inimigo.

enemy.h

```
#ifndef ENEMY_H
#define ENEMY_H

#include <QGraphicsRectItem>
#include <QObject>

class Enemy: public QObject, public QGraphicsRectItem
{
    Q_OBJECT
public:
    Enemy();

public slots:
    void move();
};

#endif // ENEMY_H
```

enemy.cpp

```
#include <QTimer>
#include <QGraphicsScene>
#include <stdlib.h>
#include <QDebug>

#include "enemy.h"
#include "game.h"

extern Game * game;

Enemy::Enemy()
{
    // posição aleatoria
    int random_number = rand() % 700;
    setPos(random_number, 0);
    // desenhar o inimigo
    setRect(0, 0, 100, 100);

    //movimentação
    QTimer * timer = new QTimer();
    connect(timer, SIGNAL(timeout()), this, SLOT(move()));

    timer->start(50);
}

void Enemy::move()
{
}
```

```

        setPos(x(),y()+5);

        if(pos().y() > 600)
        {
            scene()->removeItem(this);
            delete this;
            qDebug()<<"Enemy removed";
        }
    }
}

```

Também precisamos atualizar os arquivos:

game.cpp

```

#include <QGraphicsTextItem>
#include <QTimer>

#include "game.h"
#include "enemy.h"

Game::Game(QWidget *parent)
{
    // criar scene
    scene = new QGraphicsScene();

    // fixar o tamanho em 800x600, que é infinito por definição
    scene->setSceneRect(0,0,800,600);

    // visualizar o objeto scene (cenario)
    setScene(scene);

    //desabilitar as barras de rolagem
    setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    // fixar o tamanho
    setFixedSize(800,600);

    // criar o jogador
    player = new Player();
    // definir o tamanho do jogador 100 x 100
    player->setRect(0,0,100,100);
    // definir a posição padrão do jogador para ser em baixo da tela
    // por definição a visualização é centralizada para pegar todos os objetos
    player->setPos(400,500);
    // colocar o foco no jogador
    player->setFlag(QGraphicsItem::ItemIsFocusable);
    player->setFocus();
    // adicionar o jogador no cenário
    scene->addItem(player);

    // criar inimigos
    QTimer * timer = new QTimer();
    QObject::connect(timer,SIGNAL(timeout()),player,SLOT(spawn()));
    timer->start(2000);

    show();
}

```

player.h

```

#ifndef PLAYER_H
#define PLAYER_H

//#include <QGraphicsRectItem>
#include <QGraphicsPixmapItem>

#include <QObject>
#include <QGraphicsItem>

class Player: public QObject, public QGraphicsPixmapItem
{
    Q_OBJECT
public:
    Player(QGraphicsItem * parent=0);
    void keyPressEvent(QKeyEvent * event);

public slots:
    void spawn();
};

#endif // PLAYER_H

```

player.cpp

```

#include <QGraphicsScene>
#include <QObject>
#include <QGraphicsItem>
#include <QDebug>
#include <QKeyEvent>

#include "player.h"
#include "bullet.h"
#include "enemy.h"

Player::Player(QGraphicsItem *parent): QGraphicsRectItem(parent)
{

}

void Player::keyPressEvent(QKeyEvent *event)
{
    // mensagem informando que ocorreu uso do teclado
    //qDebug()<<"Key press detected";
    if(event->key() == Qt::Key_Left)
    {
        // limite para não sair da tela
        if(pos().x() > 0)
            setPos(x()-10,y());
    }
    else if(event->key() == Qt::Key_Right)
    {
        // limite para não sair da tela
        if(pos().x() + 100 < 800 )
            setPos(x()+10,y());
    }
    else if(event->key() == Qt::Key_Up)
    {
        setPos(x(),y()-10);
    }
    else if(event->key() == Qt::Key_Down)
    {

```

```

        setPos(x(),y()+10);
    }

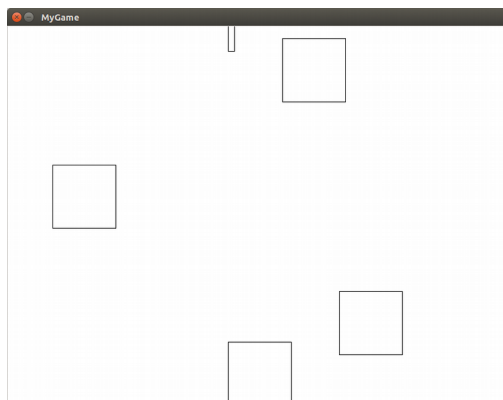
    // criar tiro (bullet)
    else if(event->key() == Qt::Key_Space)
    {
        // teste via Debug
        // qDebug()<<"Bullet created";

        Bullet * bullet = new Bullet();
        bullet->setPos(x(),y());
        scene()->addItem(bullet);
    }
}

void Player::spawn()
{
    // cria inimigo
    Enemy * enemy = new Enemy();
    scene()->addItem(enemy);
}

```

Agora nosso aplicativo deve ter essa aparência:



4. Adicionar detecção de colisões e pontuação no projeto

Uma das formas de adicionar a pontuação é criar mais duas classes **Score** e **Health** para representar a pontuação e a vida do jogador.

score.h

```

#ifndef SCORE_H
#define SCORE_H
#include <QGraphicsTextItem>

class Score: public QGraphicsTextItem
{
public:
    Score(QGraphicsItem *parent = 0);
    void increase();
    int getScore();

private:
    int score;
}

```

```
};  
  
#endif // SCORE_H
```

score.cpp

```
#include "score.h"  
#include <QFont>  
  
Score::Score(QGraphicsItem *parent):QGraphicsTextItem(parent)  
{  
    score = 0;  
    setPlainText(QString("Score: ") + QString::number(score));  
    setDefaultTextColor(Qt::blue);  
    setFont(QFont("times",16));  
}  
  
void Score::increase()  
{  
    score++;  
    setPlainText(QString("Score: ") + QString::number(score));  
    setDefaultTextColor(Qt::blue);  
    setFont(QFont("times",16));  
}  
  
int Score::getScore()  
{  
    return score;  
}
```

health.h

```
#ifndef HEALTH_H  
#define HEALTH_H  
#include <QGraphicsTextItem>  
  
class Health: public QGraphicsTextItem  
{  
public:  
    Health(QGraphicsItem *parent = 0);  
    void decrease();  
    int getHealth();  
  
private:  
    int health;  
};  
  
#endif // HEALTH_H
```

health.cpp

```
#include <QFont>  
#include "health.h"  
  
Health::Health(QGraphicsItem *parent):QGraphicsTextItem(parent)  
{
```

```

    health = 10;
    setPlainText(QString("Health: ") + QString::number(health));
    setDefaultTextColor(Qt::green);
    setFont(QFont("times",16));
}

void Health::decrease()
{
    health--;
    setPlainText(QString("Health: ") + QString::number(health));
    setDefaultTextColor(Qt::green);
    setFont(QFont("times",16));
}

int Health::getHealth()
{
    return health;
}

```

Precisamos atualizar o código em outros arquivos

bullet.h

```

#ifndef BULLET_H
#define BULLET_H

#include <QGraphicsRectItem>
#include <QObject>

class Bullet: public QObject,public QGraphicsRectItem
{
    Q_OBJECT
public:
    Bullet(QGraphicsItem * parent=0);

public slots:
    void move();
};

#endif // BULLET_H

```

bullet.cpp

```

#include <QGraphicsRectItem>
#include <QTimer>
#include <QDebug>
#include <QList>

#include "bullet.h"
#include "game.h"
#include "enemy.h"

// declaração da variavel do tipo extern
// indica que essa variavel já foi declarada em algum outro arquivo do projeto
extern Game * game;

Bullet::Bullet(QGraphicsItem *parent): QObject(), QGraphicsRectItem(parent)

```

```

{
    // desenhar o tiro
    setRect(0,0,10,50);

    // conectar o timer com o movimento
    QTimer * timer = new QTimer();
    connect(timer,SIGNAL(timeout()),this,SLOT(move()));

    // inicializar o timer
    timer->start(50);
}

void Bullet::move()
{
    // verificação de colisão:
    // se tiro atingiu o inimigo
    QList<QGraphicsItem *> colliding_item = collidingItems();

    for(int i = 0, n = colliding_item.size(); i < n; i++)
    {
        if(typeid(*(colliding_item[i]))== typeid(Enemy))
        {
            game->score->increase();
            scene()->removeItem(colliding_item[i]);
            scene()->removeItem(this);

            delete colliding_item[i];
            delete this;
            return;
        }
    }

    // movimentação do tiro
    setPos(x(),y()-10);

    if(pos().y() < -10)
    {
        scene()->removeItem(this);
        delete this;
        qDebug()<<"Bullet removed";
    }
}

```

game.h

```

#ifndef GAME_H
#define GAME_H

#include <QGraphicsView>
#include <QWidget>
#include <QGraphicsScene>

#include "player.h"
#include "score.h"
#include "health.h"

class Game: public QGraphicsView
{
public:
    Game(QWidget * parent=0);

```



```

    QGraphicsScene * scene;
    Player * player;
    Score * score;
    Health *health;
};

#endif // GAME_H

```

game.cpp

```

#include <QGraphicsTextItem>
#include <QTimer>

#include "game.h"
#include "enemy.h"

Game::Game(QWidget *parent)
{
    // criar scene
    scene = new QGraphicsScene();

    // fixar o tamanho em 800x600, que é infinito por definição
    scene->setSceneRect(0,0,800,600);

    // visualizar o objeto scene (cenario)
    setScene(scene);

    //desabilitar as barras de rolagem
    setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    // fixar o tamanho
    setFixedSize(800,600);

    // criar o jogador
    player = new Player();
    // definir o tamanho do jogador 100 x 100
    player->setRect(0,0,100,100);
    // definir a posição padrão do jogador para ser em baixo da tela
    // por definição a visualização é centralizada para pegar todos os objetos
    player->setPos(400,500);
    // colocar o foco no jogador
    player->setFlag(QGraphicsItem::ItemIsFocusable);
    player->setFocus();
    // adicionar o jogador no cenário
    scene->addItem(player);

    // criar inimigos
    QTimer * timer = new QTimer();
    QObject::connect(timer, SIGNAL(timeout()), player, SLOT(spawn()));
    timer->start(2000);

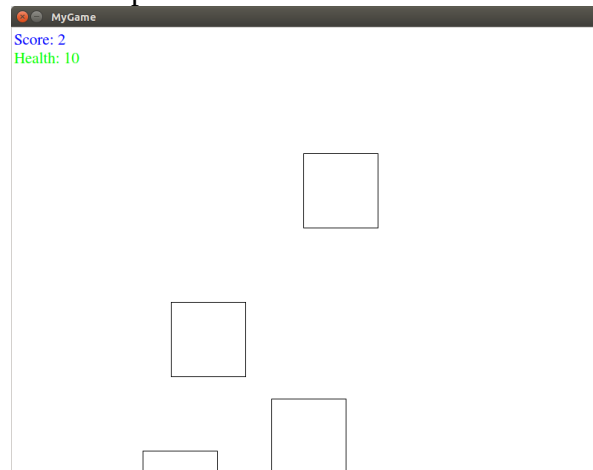
    // adicionar contagem de pontos e vida do jogador
    score = new Score();
    scene->addItem(score);
    health = new Health();
    health->setPos(health->x(), health->y()+25);
    scene->addItem(health);

    show();
}

```

```
}
```

Agora nosso aplicativo deve ter essa aparência:



Observação: O decremento de “vida” do jogador e pode ser feito no momento que os inimigos saem da tela e a memória é liberada usando

```
game->health->decrease();
```

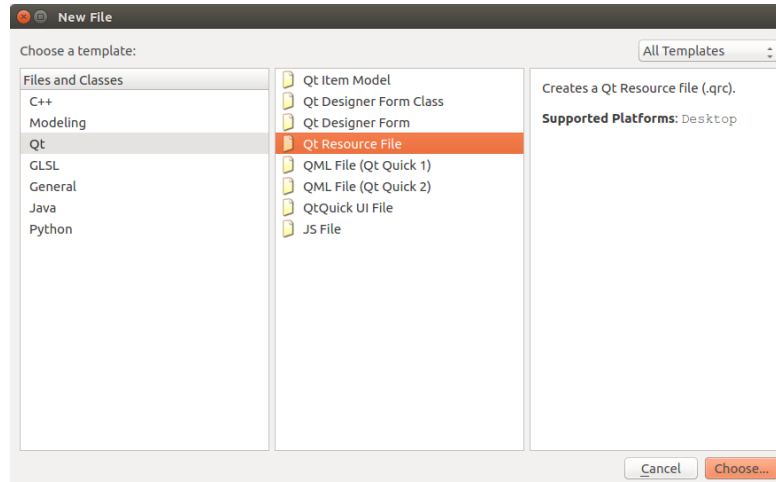
5. Modificação do projeto para incluir as imagens

É aconselhável usar recursos abertos, ou seja livres de royalties no desenvolvimento dos projetos.

Para esse projeto vamos usar imagens disponíveis em <https://opengameart.org/content/space-shooter-art>

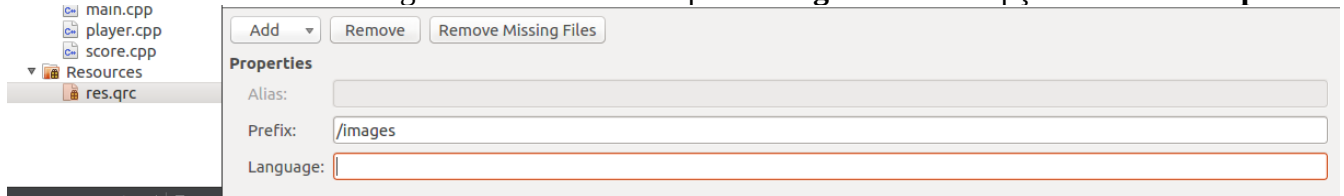
Para trabalhar com as imagens temos que mudar a classe dos objetos de **QGraphicsRectItem** para **QGraphicsPixmapItem**, com objetivo de atribuir as imagens definidas pelo usuário para cada objeto.

Para incluir arquivos de imagens ou sons é conveniente usar arquivo de recursos do QT, que podemos chamar digamos de **res**



Ainda podemos criar umas pastas para organizar melhor os arquivos que pretendemos usar.

Para trabalhar com imagens vamos criar uma pasta **images** usando a opção **Add → Add prefix**



Agora temos que copiar as imagens que pretendemos usar para representar o jogador, o tiro, o inimigo e o fundo.

Para atribuir uma imagem para cada objeto de interesse temos que mudar a classe básica dos objetos e definir qual imagem deve ser utilizada.

```
bullet.h

#ifndef BULLET_H
#define BULLET_H

// #include <QGraphicsRectItem>
#include <QGraphicsPixmapItem>
#include <QObject>

class Bullet: public QObject, public QGraphicsPixmapItem
{
    Q_OBJECT
public:
    Bullet(QGraphicsItem * parent=0);

public slots:
    void move();
};

#endif // BULLET_H
```

bullet.cpp

```
//#include <QGraphicsRectItem>
#include <QGraphicsScene>
#include <QTimer>
#include <QDebug>
#include <QList>

#include "bullet.h"
#include "game.h"
#include "enemy.h"

// declaração da variavel do tipo extern
// indica que essa variavel já foi declarada em algum outro arquivo do projeto
extern Game * game;

Bullet::Bullet(QGraphicsItem *parent): QObject(), QGraphicsPixmapItem(parent)
{
    // desenhar o tiro
    //setRect(0,0,10,50);
    setPixmap(QPixmap(":/images/laserRed.png"));

    // conectar o timer com o movimento
    QTimer * timer = new QTimer();
    connect(timer,SIGNAL(timeout()),this,SLOT(move()));

    // inicializar o timer
    timer->start(50);
}

void Bullet::move()
{
    // verificação de colisão:
    // se tiro atingiu o inimigo
    QList<QGraphicsItem *> colliding_item = collidingItems();

    for(int i = 0, n = colliding_item.size(); i < n; i++)
    {
        if(typeid(*(colliding_item[i]))== typeid(Enemy))
        {
            game->score->increase();
            scene()->removeItem(colliding_item[i]);
            scene()->removeItem(this);

            delete colliding_item[i];
            delete this;
            return;
        }
    }

    // movimentação do tiro
    setPos(x(),y()-10);

    if(pos().y() < -10)
    {
        scene()->removeItem(this);
        delete this;
        qDebug()<<"Bullet removed";
    }
}
```

```
}
```

enemy.h

```
#ifndef ENEMY_H
#define ENEMY_H

//#include <QGraphicsRectItem>
#include <QGraphicsPixmapItem>
#include <QObject>

class Enemy: public QObject, public QGraphicsPixmapItem
{
    Q_OBJECT
public:
    Enemy();

public slots:
    void move();
};

#endif // ENEMY_H
```

enemy.cpp

```
#include <QTimer>
#include <QGraphicsScene>
#include <stdlib.h>
#include <QDebug>

#include "enemy.h"
#include "game.h"

extern Game * game;

Enemy::Enemy()
{
    // posição aleatoria
    int random_number = rand() % 700;
    setPos(random_number,0);
    // desenhar o inimigo
    //setRect(0,0,100,100);
    setPixmap(QPixmap(":/images/enemyShip.png"));

    //movimentação
    QTimer * timer = new QTimer();
    connect(timer, SIGNAL(timeout()), this, SLOT(move()));

    timer->start(50);
}

void Enemy::move()
{
    setPos(x(),y()+5);
}
```

```

    if(pos().y() > 600)
    {
        game->health->decrease();

        scene()->removeItem(this);
        delete this;
        qDebug()<<"Enemy removed";
    }
}

```

player.h

```

#ifndef PLAYER_H
#define PLAYER_H

//#include <QGraphicsRectItem>
#include <QGraphicsPixmapItem>
#include <QObject>
#include <QGraphicsItem>

class Player: public QObject, public QGraphicsPixmapItem
{
    Q_OBJECT
public:
    Player(QGraphicsItem * parent=0);
    void keyPressEvent(QKeyEvent * event);

public slots:
    void spawn();
};

#endif // PLAYER_H

```

player.cpp

```

#include <QGraphicsScene>
#include <QObject>
#include <QGraphicsItem>
#include <QDebug>
#include <QKeyEvent>

#include "player.h"
#include "bullet.h"
#include "enemy.h"

Player::Player(QGraphicsItem *parent): QGraphicsPixmapItem(parent)
{
    setPixmap(QPixmap(":/images/player.png"));
}

void Player::keyPressEvent(QKeyEvent *event)
{
    // mensagem informando que ocorreu uso do teclado
    //qDebug()<<"Key press detected";
    if(event->key() == Qt::Key_Left)
    {
        // limite para não sair da tela
    }
}

```

```

        if(pos().x() > 0)
            setPos(x()-10,y());
    }
    else if(event->key() == Qt::Key_Right)
    {
        // limite para não sair da tela
        if(pos().x() + 100 < 800 )
            setPos(x()+10,y());
    }
    else if(event->key() == Qt::Key_Up)
    {
        setPos(x(),y()-10);
    }
    else if(event->key() == Qt::Key_Down)
    {
        setPos(x(),y()+10);
    }
}

// criar tiro (bullet)
else if(event->key() == Qt::Key_Space)
{
    // teste via Debug
    // qDebug()<<"Bullet created";

    Bullet * bullet = new Bullet();
    bullet->setPos(x(),y());
    scene()->addItem(bullet);
}
}

void Player::spawn()
{
    // cria inimigo
    Enemy * enemy = new Enemy();
    scene()->addItem(enemy);
}

```

O arquivo game.cpp também deve ser atualizado:

```

game.cpp

#include <QGraphicsTextItem>
#include <QTimer>

#include "game.h"
#include "enemy.h"

Game::Game(QWidget *parent)
{
    // criar scene
    scene = new QGraphicsScene();

    // fixar o tamanho em 800x600, que é infinito por definição
    scene->setSceneRect(0,0,800,600);

    // visualizar o objeto scene (cenario)
    setScene(scene);

    //desabilitar as barras de rolagem
    setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
}

```

```

setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
// fixar o tamanho
setFixedSize(800,600);

// criar o jogador
player = new Player();
// definir o tamanho do jogador 100 x 100
//player->setRect(0,0,100,100);

// definir a posição padrão do jogador para ser em baixo da tela
// por definição a visualização é centralizada para pegar todos os objetos
player->setPos(400,500);
// colocar o foco no jogador
player->setFlag(QGraphicsItem::ItemIsFocusable);
player->setFocus();
// adicionar o jogador no cenário
scene->addItem(player);

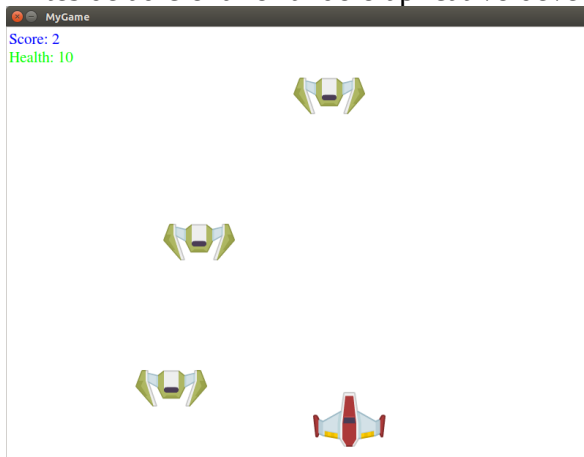
// criar inimigos
QTimer * timer = new QTimer();
QObject::connect(timer,SIGNAL(timeout()),player,SLOT(spawn()));
timer->start(2000);

// adicionar contagem de pontos e vida do jogador
score = new Score();
scene->addItem(score);
health = new Health();
health->setPos(health->x(), health->y()+25);
scene->addItem(health);

show();
}

```

Antes de adicionar o fundo o aplicativo deve ter essa aparência:



Finalmente, para adicionar o fundo temos que modificar de novo o arquivo game.cpp

```

game.cpp
#include <QGraphicsTextItem>
#include <QTimer>
#include <QImage>
#include <QBrush>

#include "game.h"

```



```

#include "enemy.h"

Game::Game(QWidget *parent)
{
    // criar scene
    scene = new QGraphicsScene();

    // fixar o tamanho em 800x600, que é infinito por definição
    scene->setSceneRect(0,0,800,600);
    scene->setBackgroundBrush(QBrush(QImage(":/images/starBackground.png")));

    // visualizar o objeto scene (cenario)
    setScene(scene);

    //desabilitar as barras de rolagem
    setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    // fixar o tamanho
    setFixedSize(800,600);

    // criar o jogador
    player = new Player();
    // definir o tamanho do jogador 100 x 100
    //player->setRect(0,0,100,100);
    player->setPos(400,500);
    // definir a posição padrão do jogador para ser em baixo da tela
    // por definição a visualização é centralizada para pegar todos os objetos
    player->setPos(400,500);
    // colocar o foco no jogador
    player->setFlag(QGraphicsItem::ItemIsFocusable);
    player->setFocus();
    // adicionar o jogador no cenário
    scene->addItem(player);

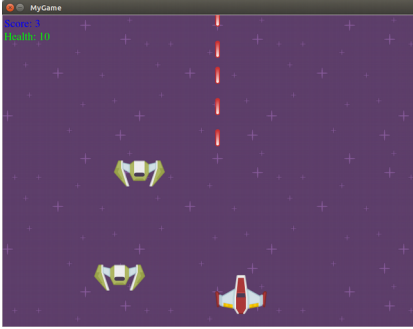
    // criar inimigos
    QTimer * timer = new QTimer();
    QObject::connect(timer,SIGNAL(timeout()),player,SLOT(spawn()));
    timer->start(2000);

    // adicionar contagem de pontos e vida do jogador
    score = new Score();
    scene->addItem(score);
    health = new Health();
    health->setPos(health->x(), health->y()+25);
    scene->addItem(health);

    show();
}

```

A versão final então fica assim:



6. Adicionar sons

Para adicionar o som de fundo e o barulho de tiro pode ser usado o componente **QMediaPlayer**.

A mesma observação feita para imagens vale também para recursos sonoros, ou seja é aconselhável utilizar os recursos de distribuição livre.

Observação: Caso apareçam problemas com esse componente no Ubuntu (componente não detectado) pode adicionar esse componente manualmente usando o seguinte comando no Terminal:

```
sudo apt-get install qtmultimedia5-dev
```

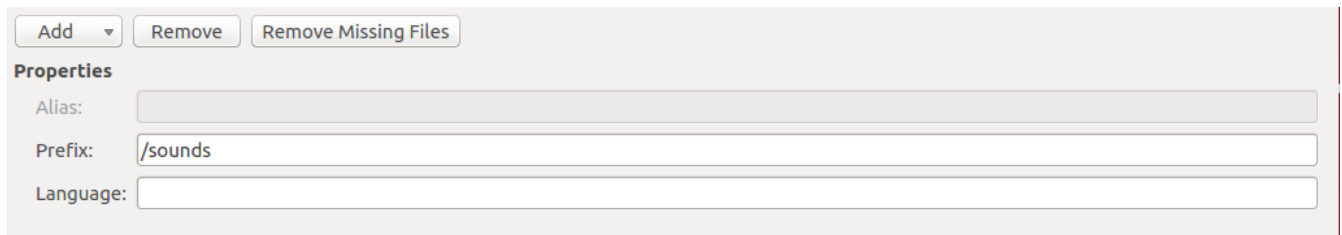
Para utilizar **QMediaPlayer** em nosso projeto temos que modificar o arquivo **MyGame.pro**

MyGame.pro

```
#-----  
#  
# Project created by QtCreator 2017-11-09T22:14:06  
#  
#-----  
  
QT      += core gui \  
        multimedia  
  
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets  
  
TARGET = MyGame  
TEMPLATE = app  
  
SOURCES += main.cpp \  
    player.cpp \  
    game.cpp \  
    bullet.cpp \  
    enemy.cpp \  
    score.cpp \  
    health.cpp  
  
HEADERS  += \  
    player.h \  
    game.h \  
    bullet.h \  
    enemy.h \  
    score.h \  
    health.h  
  
FORMS    +=  
  
RESOURCES += \  
    res.qrc
```

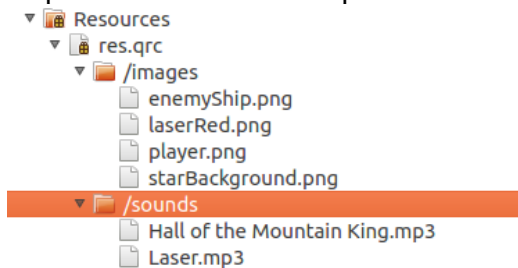
Agora temos que copiar os arquivos de som na pasta de projeto e adicionar eles no nosso arquivo de recursos.

Para deixar os recursos de forma mais organizada podemos criar um novo **prefix** chamado **sounds** e adicionar os arquivos de som sobre esse **prefix**



The screenshot shows the Qt Resource Editor's Properties tab. At the top, there are buttons for 'Add', 'Remove', and 'Remove Missing Files'. Below, the 'Prefix' field is set to '/sounds'. The 'Alias' and 'Language' fields are empty.

Depois de adicionar os arquivos de som vamos ter os seguintes recursos no nosso projeto:



Para adicionar o som de fundo e barulho de tiro temos que adicionar o código nos seguintes arquivos:

```
game.cpp

#include <QGraphicsTextItem>
#include <QTimer>
#include <QImage>
#include <QBrush>
#include <QMediaPlayer> // sudo apt-get install qtmultimedia5-dev

#include "game.h"
#include "enemy.h"

Game::Game(QWidget *parent)
{
    // criar scene
    scene = new QGraphicsScene();

    // fixar o tamanho em 800x600, que é infinito por definição
    scene->setSceneRect(0,0,800,600);
    scene->setBackgroundBrush(QBrush(QImage(":/images/starBackground.png")));

    // visualizar o objeto scene (cenario)
    setScene(scene);

    //desabilitar as barras de rolagem
    setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    // fixar o tamanho
    setFixedSize(800,600);

    // criar o jogador
    player = new Player();
    // definir o tamanho do jogador 100 x 100
    //player->setRect(0,0,100,100);
```

```

player->setPos(400,500);
// definir a posição padrão do jogador para ser em baixo da tela
// por definição a visualização é centralizada para pegar todos os objetos
player->setPos(400,500);
// colocar o foco no jogador
player->setFlag(QGraphicsItem::ItemIsFocusable);
player->setFocus();
// adicionar o jogador no cenário
scene->addItem(player);

// criar inimigos
QTimer * timer = new QTimer();
QObject::connect(timer,SIGNAL(timeout()),player,SLOT(spawn()));
timer->start(2000);

// adicionar contagem de pontos e vida do jogador
score = new Score();
scene->addItem(score);
health = new Health();
health->setPos(health->x(), health->y()+25);
scene->addItem(health);

// adicionar som de fundo
QMediaPlayer * music = new QMediaPlayer(this);
music->setMedia(QUrl("qrc:/sounds/Hall of the Mountain King.mp3"));
music->play();

show();
}

```

Depois dessa modificação o som de fundo já deve aparecer no jogo.

Para adicionar o barulho de tiro temos que adicionar o código nos seguintes arquivos:

```

player.h

#ifndef PLAYER_H
#define PLAYER_H

//#include <QGraphicsRectItem>
#include <QGraphicsPixmapItem>
#include <QObject>
#include <QGraphicsItem>
#include <QMediaPlayer>

class Player: public QObject, public QGraphicsPixmapItem
{
    Q_OBJECT
public:
    Player(QGraphicsItem * parent=0);
    void keyPressEvent(QKeyEvent * event);

public slots:
    void spawn();

private:
    QMediaPlayer * bulletSound;
};

#endif // PLAYER_H

```

```

#include <QGraphicsScene>
#include <QObject>
#include <QGraphicsItem>
#include <QDebug>
#include <QKeyEvent>

#include "player.h"
#include "bullet.h"
#include "enemy.h"

Player::Player(QGraphicsItem *parent): QGraphicsPixmapItem(parent)
{
    setPixmap(QPixmap(":/images/player.png"));

    bulletsound = new QMediaPlayer();
    bulletsound->setMedia(QUrl("qrc:/sounds/Laser.mp3"));
}

void Player::keyPressEvent(QKeyEvent *event)
{
    // mensagem informando que ocorreu uso do teclado
    //qDebug()<<"Key press detected";
    if(event->key() == Qt::Key_Left)
    {
        // limite para não sair da tela
        if(pos().x() > 0)
            setPos(x()-10,y());
    }
    else if(event->key() == Qt::Key_Right)
    {
        // limite para não sair da tela
        if(pos().x() + 100 < 800 )
            setPos(x()+10,y());
    }
    else if(event->key() == Qt::Key_Up)
    {
        setPos(x(),y()-10);
    }
    else if(event->key() == Qt::Key_Down)
    {
        setPos(x(),y()+10);
    }

    // criar tiro (bullet)
    else if(event->key() == Qt::Key_Space)
    {
        // teste via Debug
        // qDebug()<<"Bullet created";

        Bullet * bullet = new Bullet();
        bullet->setPos(x(),y());
        scene()->addItem(bullet);

        // adicionar barulho de tiro
        // analisando o estado de reprodução de Player
        if(bulletsound->state() == QMediaPlayer::PlayingState)
        {
            // caso o som ainda esta tocando resetar o Player para posição

```

```

inicial
    bulletsound->setPosition(0);
}
else if(bulletsound->state() == QMediaPlayer::StoppedState)
{
    bulletsound->play();
}
}

void Player::spawn()
{
    // cria inimigo
    Enemy * enemy = new Enemy();
    scene()->addItem(enemy);
}

```

Observação: Para implementar a repetição do som de fundo pode ser utilizado o mecanismo de **Signal and Slots** para analisar o estado do **QMediaPlayer** ou então utilizar **QMediaPlaylist**:

```

QMediaPlaylist *playlist = new QMediaPlaylist();
playlist->addMedia(QUrl("qrc:/sounds/Hall of the Mountain King.mp3"));
playlist->setPlaybackMode(QMediaPlaylist::Loop);

QMediaPlayer * music = new QMediaPlayer(this);
music->setPlaylist(playlist);
//music->setMedia(QUrl("qrc:/sounds/Hall of the Mountain King.mp3"));
music->play();

```