Qt Tutorial #2

2017.11.22

Goal

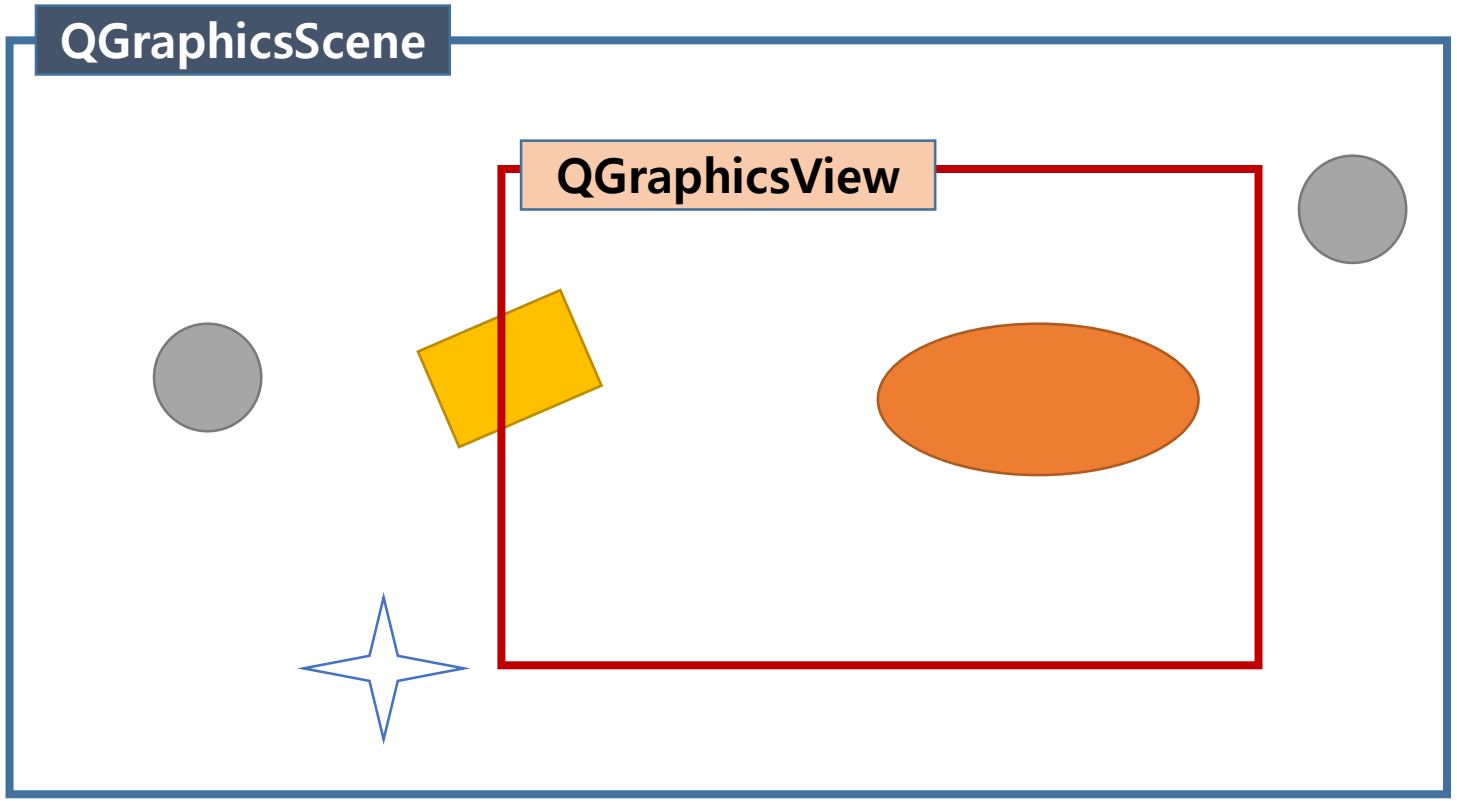
- 간단한 슈팅 게임 구현
- 플레이어: 조작에 따라 좌우로 이동, 총알 발사
- 총알: 자동으로 위로 이동, 적과 충돌하면 적과 함께 소멸
- 적: 일정시간 간격으로 랜덤한 위치에 생성, 자동으로 아래로 이동, 총알에 맞으면 소멸

- QGraphicsScene, QGraphicsItem, QGraphicsView, QTimer, QKeyEvent, QList

QGraphicsScene, QGraphicsView

- QGraphicsScene: 그래픽 요소들이 배치된 하나의 공간
- QGraphicsView: Scene 에서 실제로 유저의 화면에 그려지는 부분

  (유저가 확인할 수 있는 View)



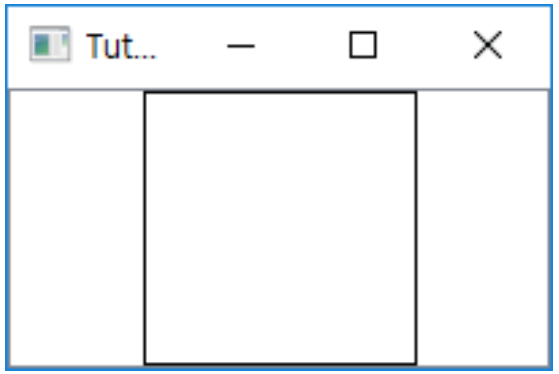
QGraphicsItem

- QGraphicsScene에 들어갈 수 있는 객체 클래스
- Geometry, Collision Detection, Painting 등을 포함
- http://doc.qt.io/qt-5/qgraphicsitem.html
- QGraphicsRectItem, QGraphicsEllipseItem 등의 하위 클래스 존재
- 이번 시간에는 QGraphicsRectItem을 사용

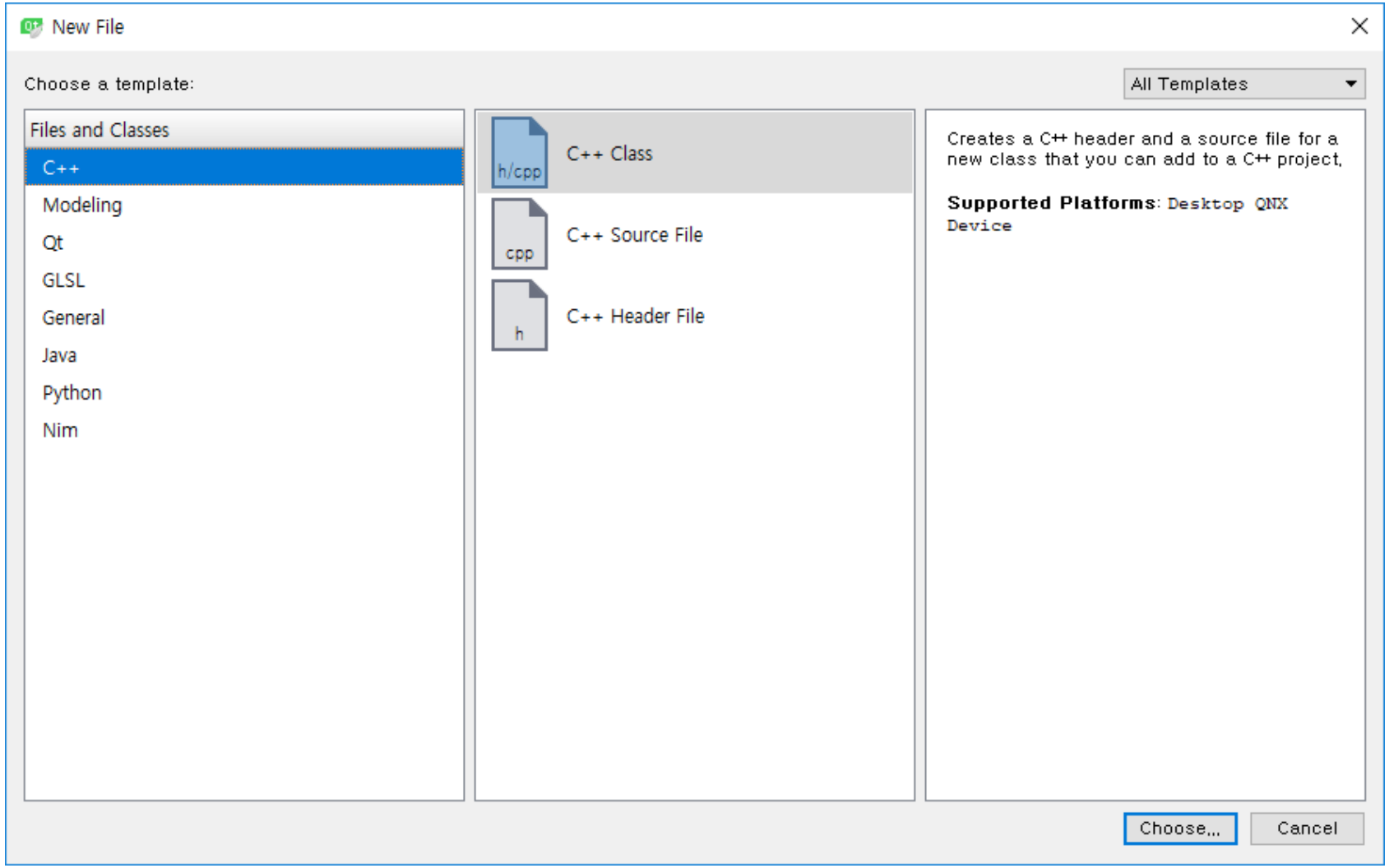
QGraphicsItem으로 사각형 그려보기

main.cpp

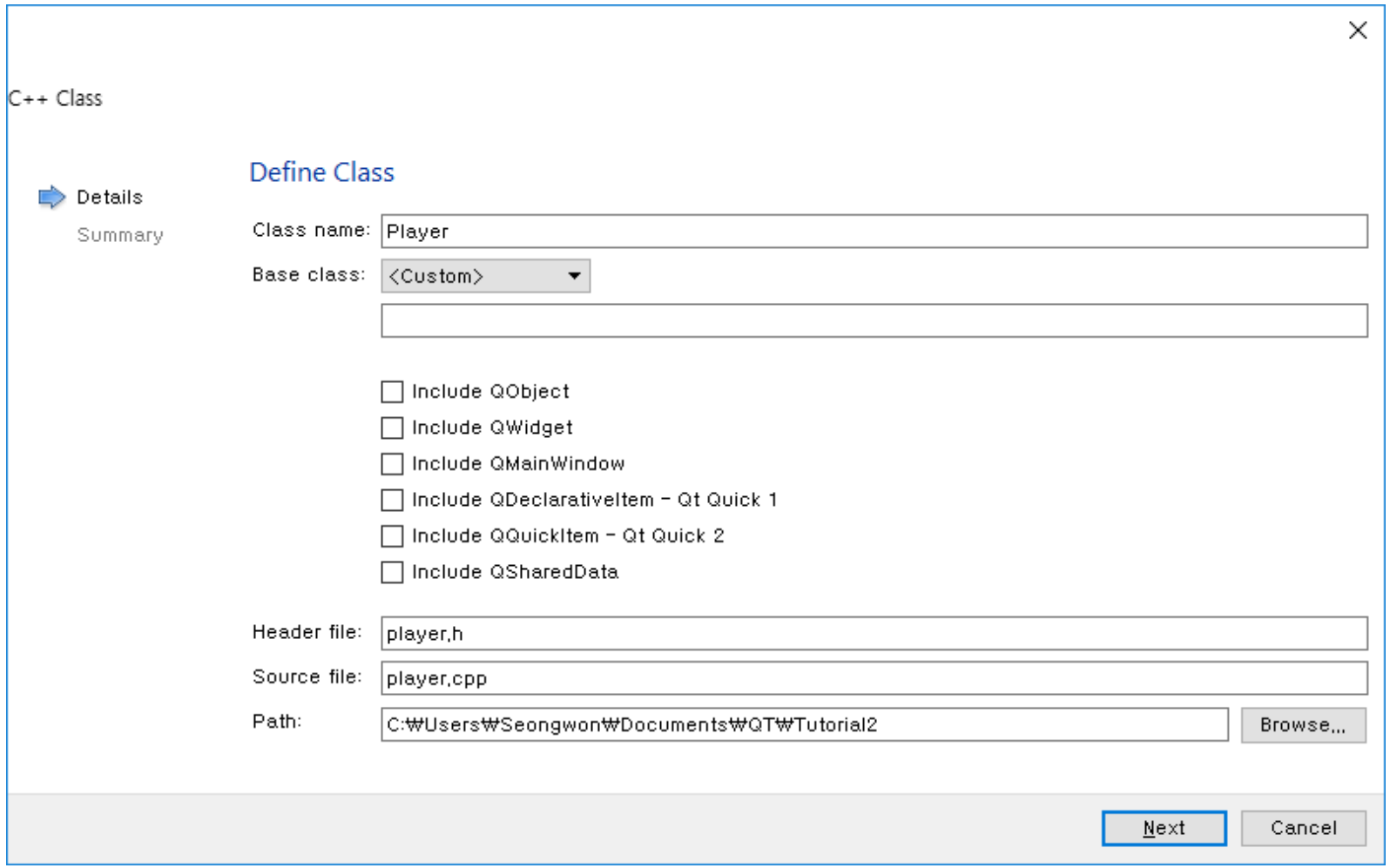
```
#include <QApplication>

#include <QGraphicsScene>
#include <QGraphicsView>
#include <QGraphicsRectItem>

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

    QGraphicsScene* scene = new QGraphicsScene();

    QGraphicsRectItem* item = new QGraphicsRectItem();
    item->setRect(0,0,100,100);
    scene->addItem(item);

    QGraphicsView* view = new QGraphicsView(scene);
    view->show();
    return a.exec();
}
```



QGraphicsItem을 상속하는 캐릭터 클래스 생성



QGraphicsItem으로 캐릭터 클래스 생성



QGraphicsItem으로 캐릭터 클래스 생성

Player.h

```
#ifndef PLAYER_H
#define PLAYER_H

#include <QGraphicsRectItem>

class Player : public QGraphicsRectItem
{
public:
    Player();
};

#endif // PLAYER_H
```

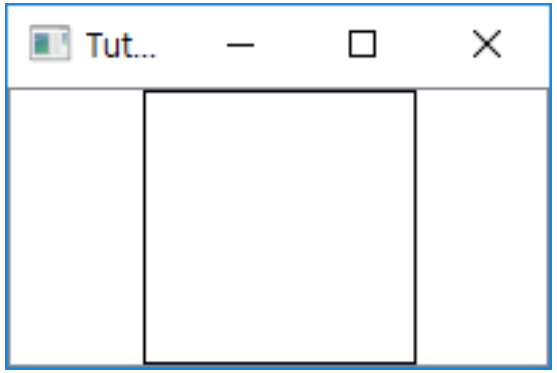
Player.cpp

```
#include "player.h"

Player::Player()
{

}
```

캐릭터를 Scene에 추가

main.cpp

```
#include <QApplication>
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QGraphicsRectItem>

#include "player.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QGraphicsScene* scene = new QGraphicsScene();

    Player* player = new Player();
    player->setRect(0, 0, 100, 100);
    scene->addItem(player);

    QGraphicsView* view = new QGraphicsView(scene);
    view->show();
    return a.exec();
}
```



키보드 입력 받기 (QKeyEvent)

Player.h

```
#ifndef PLAYER_H
#define PLAYER_H

#include <QGraphicsRectItem>

class Player : public QGraphicsRectItem
{
public:
    Player();

    void keyPressEvent(QKeyEvent* event);
};

#endif // PLAYER_H
```

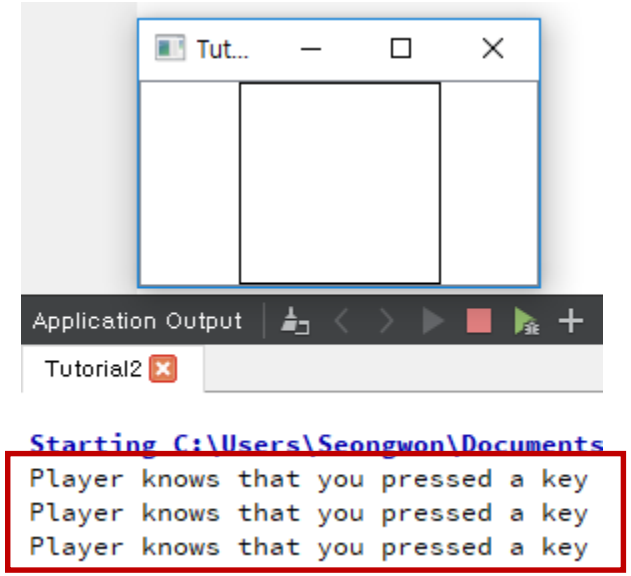
Player.cpp

```
#include "player.h"

#include <QDebug>
#include <QKeyEvent>

Player::Player()
{

}

void Player::keyPressEvent(QKeyEvent *event)
{
    qDebug() << "Player knows that you pressed a key";
}
```

keyPressEvent는 QGraphicsItem의 virtual 함수. 키 입력에 대한 이벤트 처리.

Focusable

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

    QGraphicsScene* scene = new QGraphicsScene();

    Player* player = new Player();
    player->setRect(0, 0, 100, 100);

    player->setFlag(QGraphicsItem::ItemIsFocusable);
    player->setFocus();

    scene->addItem(player);

    QGraphicsView* view = new QGraphicsView(scene);
    view->show();

    return a.exec();
}
```



Focus 설정을 하지 않으면 키를 입력해도 아무 일이 일어나지 않음
Focus 설정을 하면 키보드를 누를 때마다 메시지 출력

특정한 키가 눌렸는지 확인을 하려면?

- QKeyEvent의 key()를 통해 현재 눌린 key에 대한 정보를 알 수 있음.

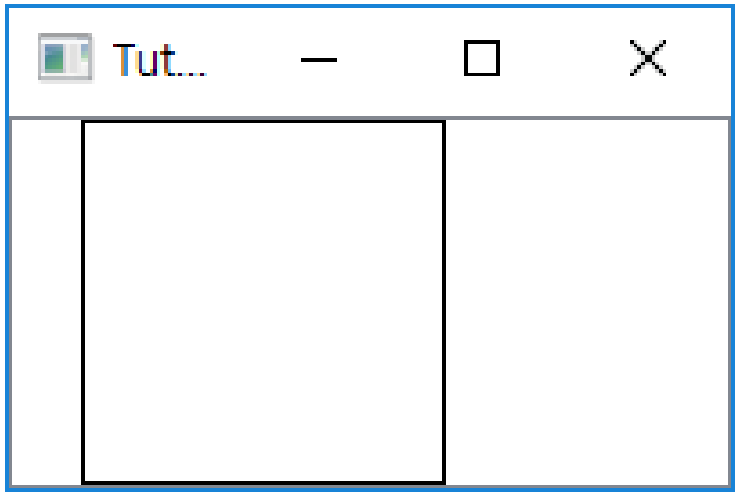
```
void Player::keyPressEvent(QKeyEvent *event)
{
    if(event->key() == Qt::Key_Left){
        setPos(x()-10, y());
    }
    if(event->key() == Qt::Key_Right){
        setPos(x()+10, y());
    }
}
```

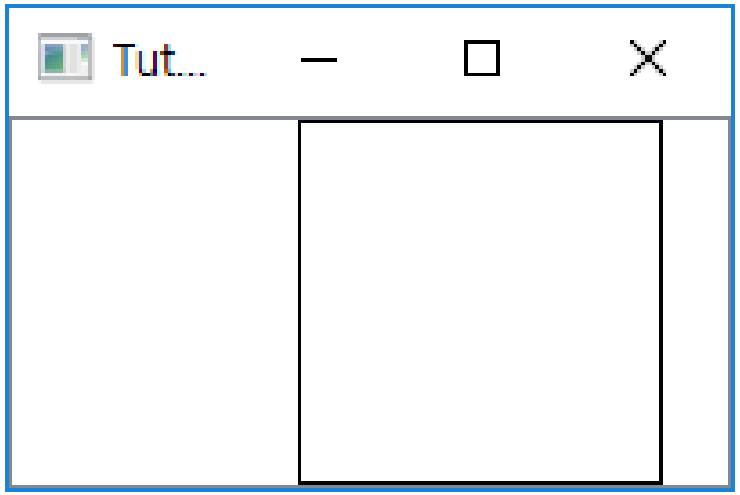
- 위의 예제에서는 입력된 키가 왼쪽 방향키, 오른쪽 방향키일 때 특정한 동작 수행

- setPos는 Item의 좌표를 설정하는 함수

- 왼쪽 방향키를 누르면 x 좌표가 10 감소해 왼쪽으로 이동
- 오른쪽 방향키를 누르면 x 좌표가 10 증가해 오른쪽으로 이동

실행 예제





총알

- 다음으로 스페이스바를 누르면 총알 객체를 생성하도록 구현해봅시다.

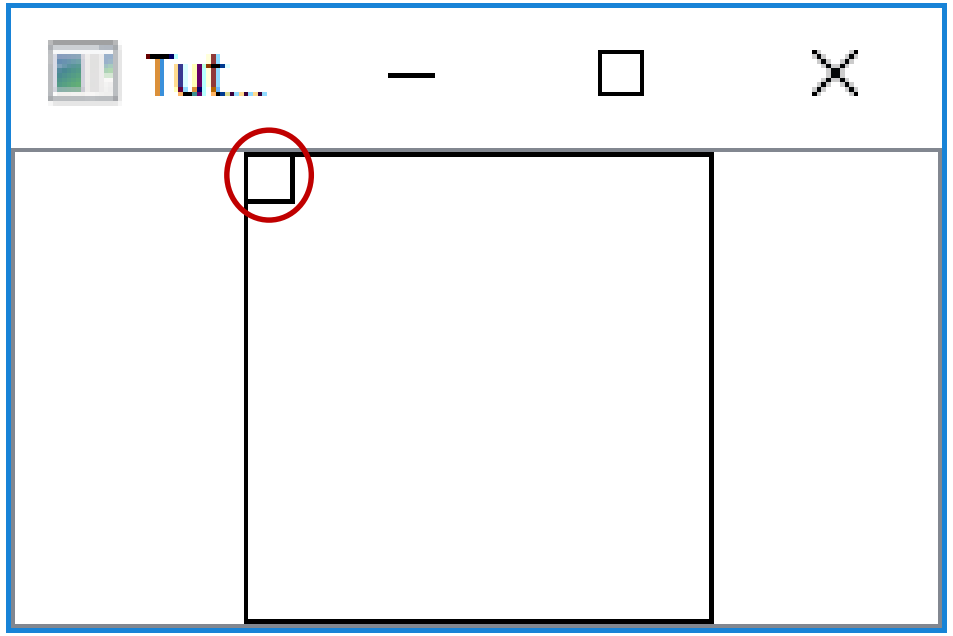
- 먼저 총알에 대한 클래스를 만들어주세요.
(총알도 QGraphicsRectItem을 상속하는 것으로 합니다.)

- 그리고 생성자에서 setRect를 사용해 생성과 동시에 그려지게 해줍시다

```
Bullet::Bullet()
{
    setRect(0, 0, 10, 10);
}
```

스페이스 바를 누를 때 총알 생성

```
void Player::keyPressEvent(QKeyEvent *event)
{
    if(event->key() == Qt::Key_Left){
        setPos(x()-10, y());
    }
    if(event->key() == Qt::Key_Right){
        setPos(x()+10, y());
    }
    if(event->key() == Qt::Key_Space){
        Bullet* bullet = new Bullet;
        scene()->addItem(bullet);
    }
}
```



총알 움직임 (QTimer)

Bullet.h

```
#include <QGraphicsRectItem>
#include <QObject>

class Bullet : public QObject,
               public QGraphicsRectItem
{
    Q_OBJECT

public:
    Bullet();

public slots:
    void move();
};
```

QObject를 상속하고 Q_OBJECT를 적어야 정의된 Slot을 인식

Bullet.cpp

```
Bullet::Bullet()
{
    setRect(0, 0, 10, 10);

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

    timer->start(50);
}

void Bullet::move()
{
    setPos(x(), y()-10);
}
```

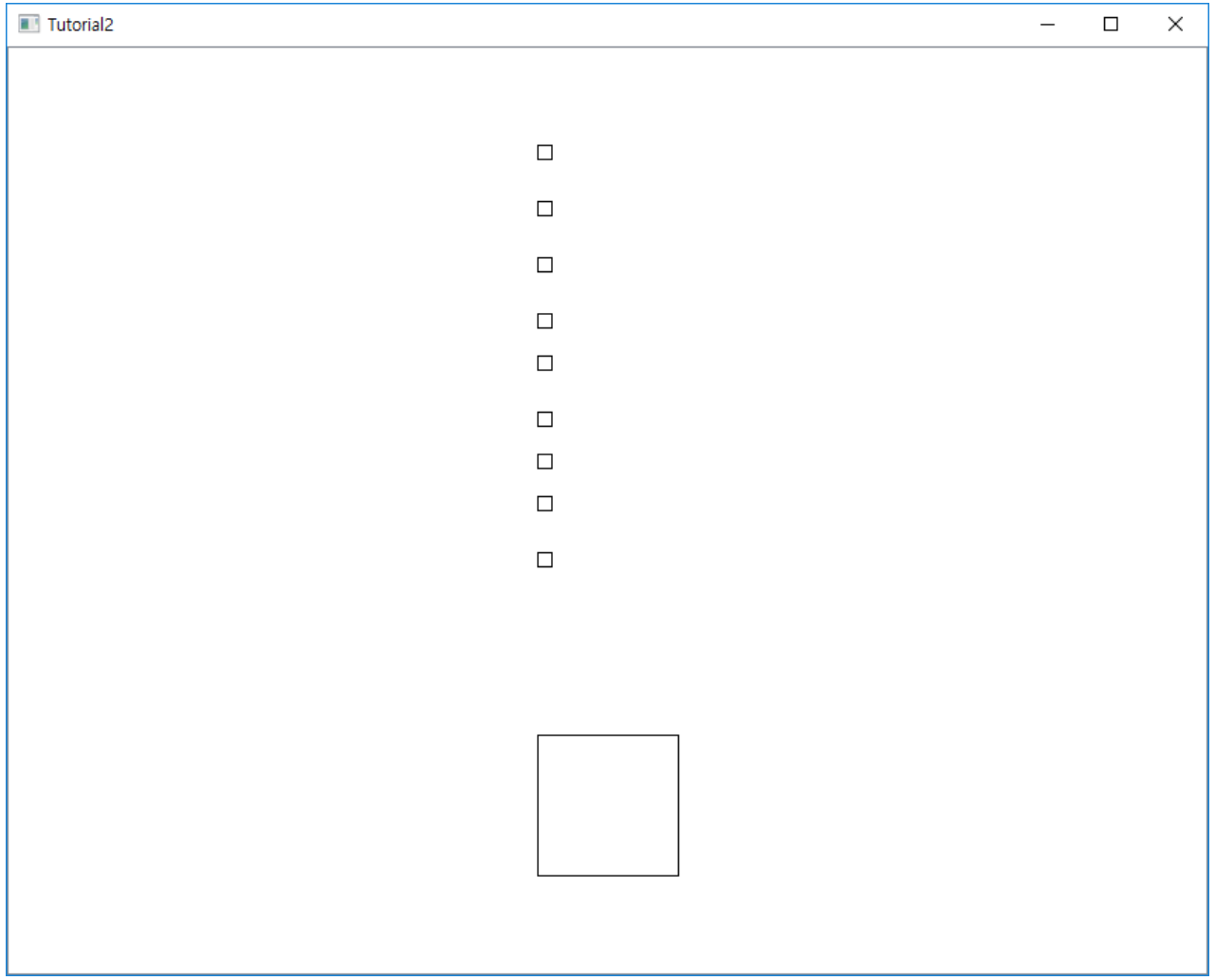
QGraphics에서는 y 감소가 위쪽 방향

혹시 컴파일 과정 중 vtable 에러가 생기면?

```
SOURCES += ₩
        main.cpp ₩
        mainwindow.cpp ₩
    bullet.cpp₩
    player.cpp

HEADERS += ₩
        mainwindow.h ₩
    bullet.h ₩
    player.h

FORMS += ₩
        mainwindow.ui
```

왼쪽과 같이 bullet과 player의 순서를 바꿔보세요

실행 예제



- 화면을 드래그해서 늘린 상태
- 스페이스바를 누르면 총알이 생성되고 위로 이동
- 총알이 계속 움직이면 화면이 밀려나고 스크롤바 생성

View 설정

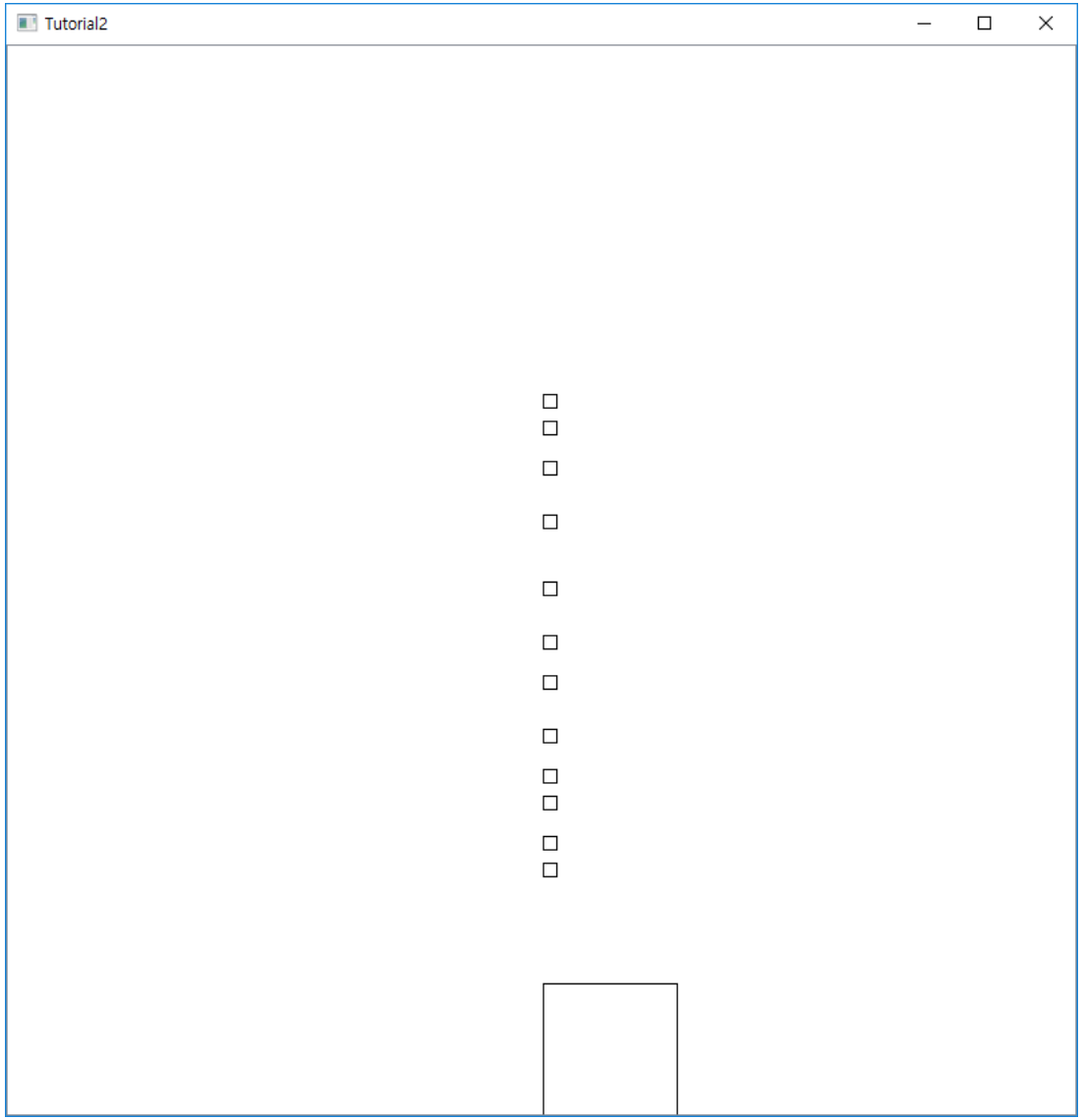
```
int main(int argc, char *argv[])
{
    …
    …
    QGraphicsView* view = new QGraphicsView(scene);

    view->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    view->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

    view->show();

    view->setFixedSize(800, 800);
    scene->setSceneRect(0, 0, 800, 800);

    player->setPos(view->width()/2, view->height() – player->rect().height());

    return a.exec();
}
```

스크롤바 Off

Scene, View 사이즈 설정

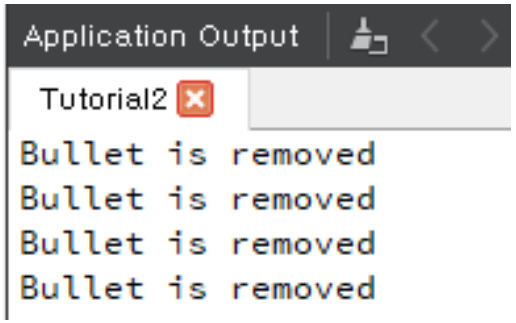
플레이어 초기 위치 설정

실행 예제



총알 소멸

```
#include "bullet.h"
#include <QTimer>
#include <QGraphicsScene>
#include <QDebug>

void Bullet::move()
{
    setPos(x(), y()-10);

    if(pos().y() < 0){
        scene()->removeItem(this);
        delete this;
        qDebug() << "Bullet is removed";
    }
}
```



scene(): 현재 view가 나타내고 있는 QGraphicsScene 포인터 리턴

응용: Enemy 생성 (Self!)

- Enemy class 생성

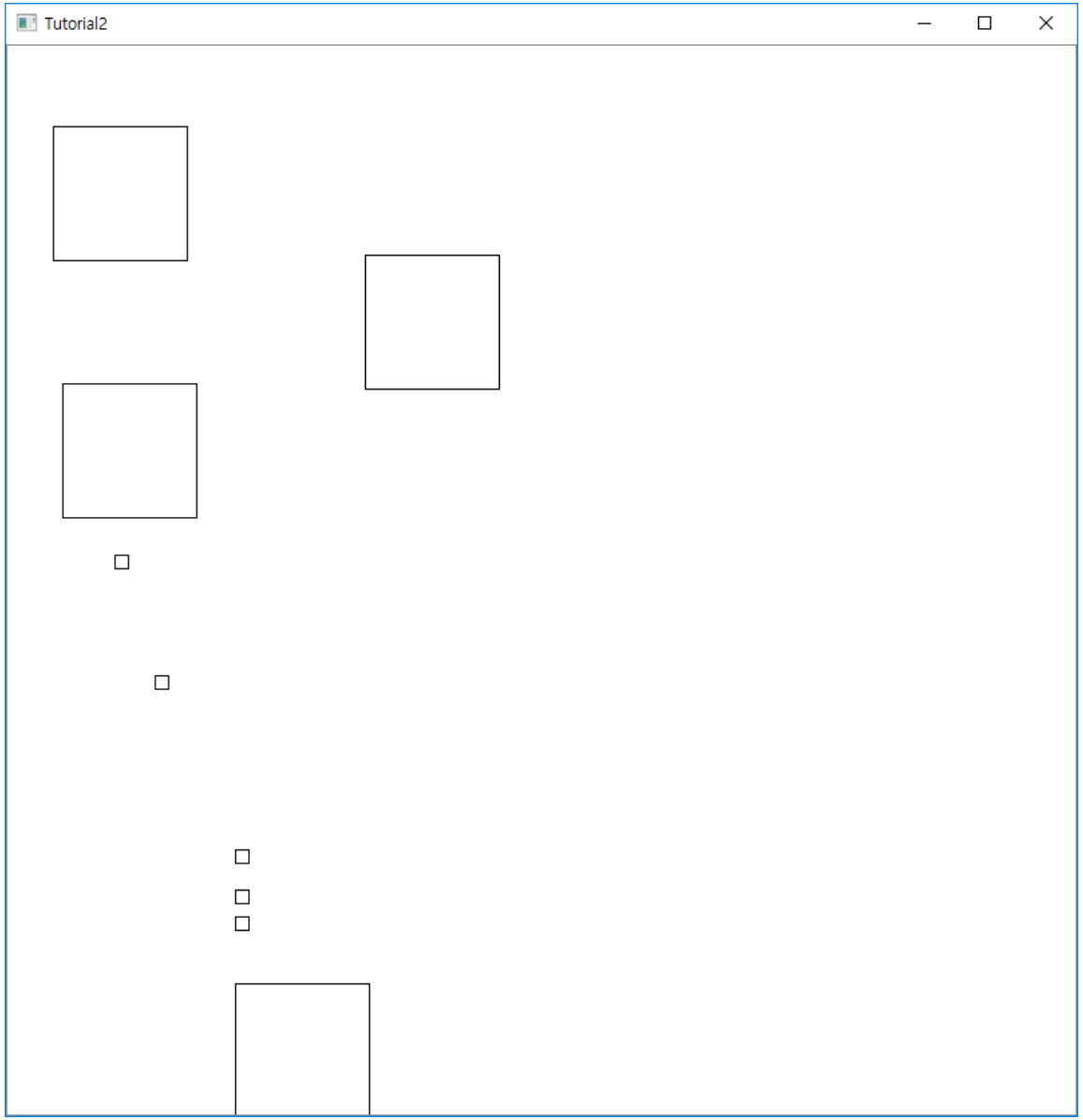
- Main에서 QTimer를 사용해 2초 간격으로 Enemy를 생성
(이번에는 Slot 함수를 Player에 정의하고 사용)

- Enemy의 초기 x 좌표를 랜덤하게 설정, 크기는 (100, 100)으로 설정
(x: 0~700 사이 랜덤, y: 0으로 고정)

- Enemy가 일정 시간 간격마다 아래 방향으로 이동하도록 구현
(50ms 마다 3씩 이동)

- Enemy가 화면 밖을 벗어날 때 소멸하도록 구현

실행 예제



충돌 체크

```
...

#include <QList>
#include "enemy.h"

void Bullet::move()
{
    QList<QGraphicsItem *> collided = collidingItems();
    for (int i=0; i<collided.size(); i++){
        if(typeid(*(collided[i])) == typeid(Enemy)){
            scene()->removeItem(collided[i]);
            scene()->removeItem(this);
            delete collided[i];
            delete this;
            return;
        }
    }
    ...
}
```

QList : linked list와 같은 배열 형태라고 생각하면 편하다

typeid는 어떤 객체인지 혹은 어떤 타입인지를 확인시켜준다.

왼쪽 코드는 충돌한 object가 enemy인 경우 제거하는 모습이다.

typeid(A): Run time에 A 객체의 타입 확인

참고 영상

https://www.youtube.com/watch?v=8ntEQpg7gck

End!