Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások fejlesztése I

6. előadás

Grafikus felületű alkalmazások tesztelése

Giachetta Roberto

http://people.inf.elte.hu/groberto

Grafikus felületű alkalmazások tesztelése

Tesztelés

- A tesztelés célja a szoftverhibák felfedezése és szoftverrel szemben támasztott minőségi elvárások ellenőrzése
  - a tesztelés során különböző *teszteset*eket (*test case*) különböztetünk meg, amelyek az egyes funkciókat, illetve elvárásokat tudják ellenőrizni
    - megadjuk, adott bemenő adatokra mi a várt eredmény (*expected result*), amelyet a teszt lefutása után összehasonlítunk a kapott eredménnyel (*actual result*)
- A tesztelés nem a teljes program elkészülte után, egyben történik, hanem általában 3 szakaszból áll: *fejlesztői teszt* (*development testing*), *kiadásteszt* (*release testing*), *felhasználói teszt* (*acceptance testing*)

Grafikus felületű alkalmazások tesztelése

Tesztelés

- A fejlesztői tesztnek további négy szakasza van:
  - *egységteszt* (*unit test*): a programegységeket (osztályok, metódusok) külön-külön, egymástól függetlenül teszteljük
  - *integrációs teszt* (*integration test*): a programegységek együttműködésének tesztje, a rendszer egy komponensének vizsgálata
  - *rendszerteszt* (*system test*): az egész rendszer együttes tesztje, a rendszert alkotó komponensek közötti kommunikáció vizsgálata
- A tesztelés egy része automatizálható, bizonyos részét azonban mindenképpen manuálisan kell végrehajtanunk

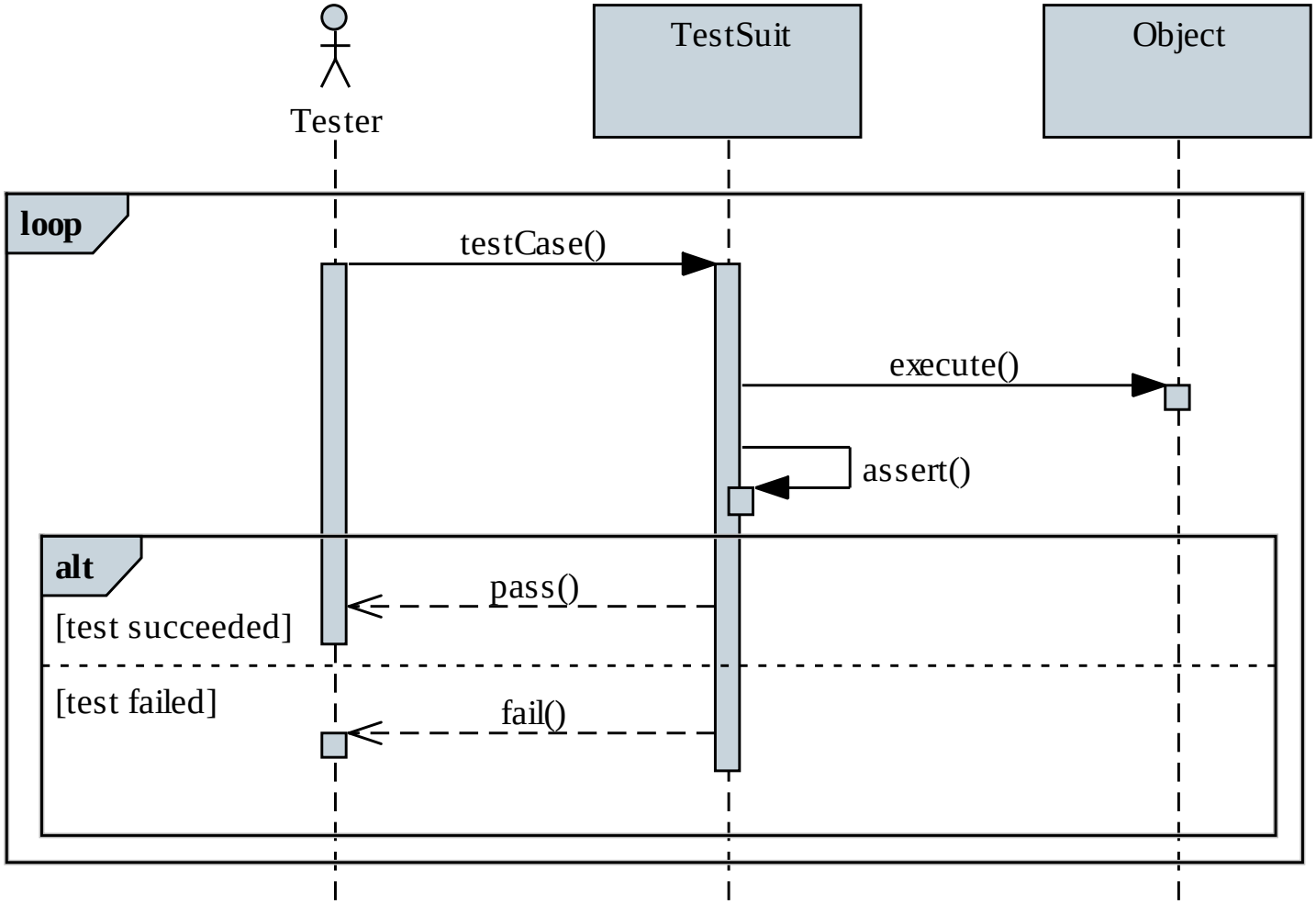
Grafikus felületű alkalmazások tesztelése

Egységtesztek

- Az egységtesztek automatizálását, és az eredmények kiértékelését hatékonyabbá tehetjük tesztelési keretrendszerek (*unit testing frameworks*) használatával
  - általában a tényleges főprogramoktól függetlenül építhetünk teszteseteket, amelyeket futtathatunk, és megkapjuk a futás pontos eredményét
  - a tesztestekben egy, vagy több ellenőrzés (*assert*) kap helyet, amelyek jelezhetnek hibákat
  - amennyiben egy hibajelzést sem kaptunk egy tesztesetből, akkor az eset sikeres (*pass*), egyébként sikertelen (*fail*)
  - alapvető eszköze a *tesztvezérelt fejlesztés*nek (*Test Driven Development*, *TDD*)

Grafikus felületű alkalmazások tesztelése

Egységtesztek



Grafikus felületű alkalmazások tesztelése

Tesztelés Qt keretrendszerben

- A Qt keretrendszer tartalmaz egy beágyazott tesztelő modul (*QTestLib*), amely lehetőségeket ad egységtesztek és teljesítménytesztek könnyű megfogalmazására, és végrehajtására
  - a tesztekhez szükséges funkciókat a **`QtTest`** fájlban találjuk
  - a tesztkörnyezetet **`QObject`** leszármazott osztályokban valósítjuk meg (amelyeket ellátunk **`Q_OBJECT`** makróval)
  - a tesztesetek eseménykezelők lesznek, amelyekben ellenőrzéseket végzünk
  - a projektben megjelöljük a modul használatát (**`QT += testlib`**)

Grafikus felületű alkalmazások tesztelése

Tesztelés Qt keretrendszerben

- Az ellenőrzéseket makrók segítségével valósítjuk meg, pl.:
  - logikai kifejezés ellenőrzése: **`QVERIFY(<kifejezés>)`**
  - összehasonlítás:
    `QCOMPARE(<aktuális érték>, <várt érték>)`
  - hiba: **`QFAIL(<üzenet>)`**
  - figyelmeztetés: **`QWARN(<üzenet>)`**
- A teszt futtatását a **`QTEST_MAIN(<osztálynév>)`** (vagy **`QTEST_MAIN(<osztálynév>)`**) makróval végezhetjük, amely automatikusan legenerál egy főprogramot, és végrehajtja a teszteseteket, így a tesztek egyszerű konzolos alkalmazásként futtathatóak

Grafikus felületű alkalmazások tesztelése

Tesztelés Qt keretrendszerben

- Pl.:

```
class MyClass {
    // tesztelendő osztály
private:
    int _value;

public:
    // a publikus műveleteket teszteljük
    MyClass (int v) { _value = v; }
    void add(int v) { _value += v; }
    int getValue() const { return _value; }
}
```

Grafikus felületű alkalmazások tesztelése

Tesztelés Qt keretrendszerben

```
class MyClassTest : QObject {
   // tesztkörnyezet
   Q_OBJECT
private slots:
   // tesztesetek, mint eseménykezelők

   void testGetValue()
   {
      MyClass mc(10);
      // végrehajtunk egy ellenőrzést:
      QVERIFY(mc.getValue() == 10);
      // másként:
      // QCOMPARE(mc.getValue(), 10);
   }
```

Grafikus felületű alkalmazások tesztelése

Tesztelés Qt keretrendszerben

```
    void testAdd()
    {
        MyClass mc(10);
        mc.add(5);
        QCOMPARE(mc.getValue(), 15);

        mc.add(15);
        QCOMPARE(mc.getValue(), 30);
        // tetszőleges sok ellenőrzést
        // végezhetünk
    }
    …
}
```

Grafikus felületű alkalmazások tesztelése

Tesztelés Qt keretrendszerben

- A Qt Creator biztosít egy teszt projekt típust (*Qt Unit Test*)
  - létrehozza a megadott tesztkörnyezetet, valamint a főprogram generátort (egy forrásfájlban)
- A tesztünk futtatása részletes eredményt ad, tesztesetenként láthatjuk az eredményt, az esetleges hibajelenséget, valamint a hiba helyét, pl.:

```
PASS    : MyClassTest::testGetValue()
PASS    : MyClassTest::testAddValue()
FAIL!   : MyClassTest::…()
   Compared values are not the same
   Loc : [../MyTest/myclasstest.cpp(106)]!
Totals: 2 passed, 1 failed, 0 skipped
```

Grafikus felületű alkalmazások tesztelése

Tesztelés Qt keretrendszerben

- Lehetőségünk van a tesztkörnyezet konfigurálására
  - a tesztkörnyezetben mezőként bármilyen adatot eltárolhatunk, ezeket speciális eseménykezelőkkel állíthatjuk
  - az első teszteset előbb lefut a tesztkörnyezet inicializálás (`initTestCase`)
  - az utolsó teszteset után lefut a tesztkörnyezet megsemmisítés (`cleanupTestCase`)
  - minden teszt előtt lefut a teszteset inicializálás (`init`)
  - minden teszt után lefut a teszteset megsemmisítés (`cleanup`)

Grafikus felületű alkalmazások tesztelése

Tesztelés Qt keretrendszerben

- Pl.:

```
class MyClassTest : QObject {
   Q_OBJECT
private: // a tesztkörnyezet értékei
   MyClass* _mc;
private slots:
   void initTestCase() { // inicializálás
      _mc = new MyClass(10);
   }
   void cleanupTestCase() { // megsemmisítés
      delete _mc;
   }
   // _mc az összes tesztesetben elérhető lesz
   …
```

Grafikus felületű alkalmazások tesztelése

Példa

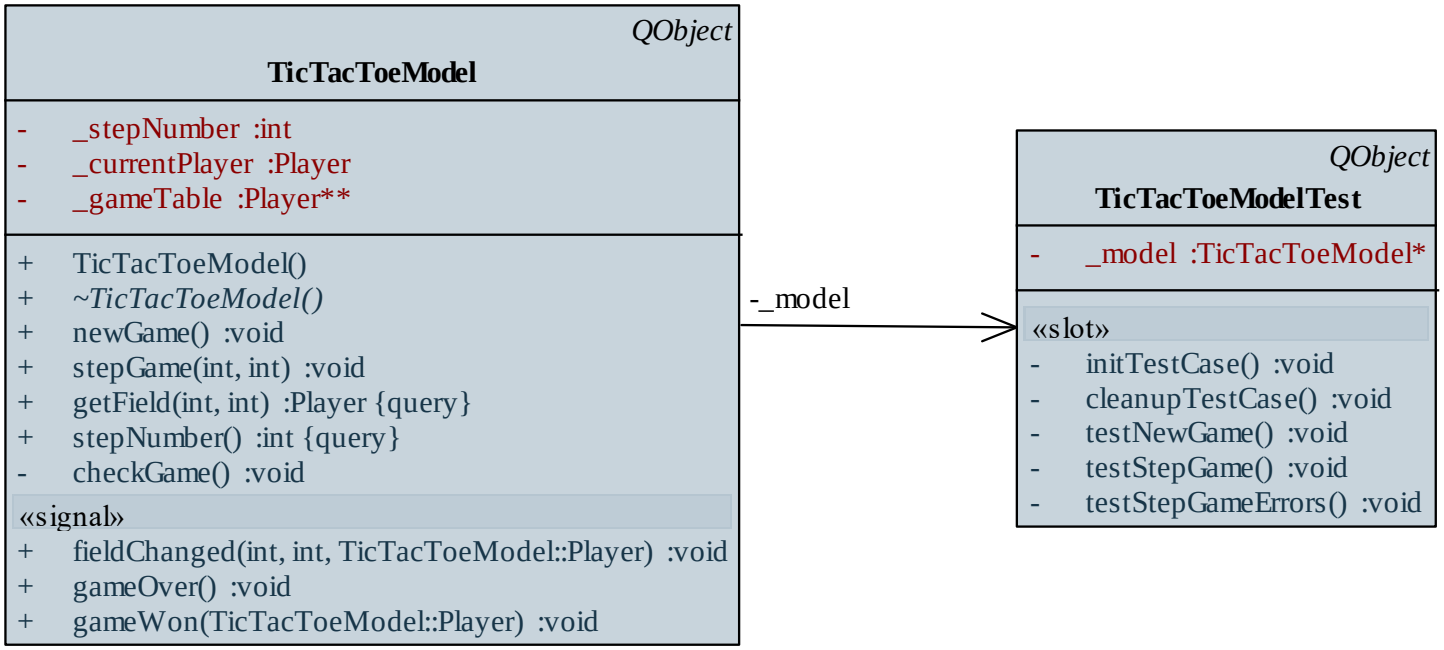
Feladat: Teszteljük le a Tic-Tac-Toe játék kétrétegű megvalósításának modelljét.

- létrehozunk egy tesztprojektet, amelybe bemásoljuk a **`TicTacToeModel`** osztályt
- létrehozunk egy tesztkörnyezetet (**`TicTacToeModelTest`**), amelyben teszteljük új játék kezdését (**`testNewGame`**), lépések végrehajtását (**`testStepGame`**)
- a tesztkörnyezet tárolja a modell egy példányát, amelyet inicializál (**`initTestCase`**), majd megsemmisít (**`cleanupTestCase`**)

Grafikus felületű alkalmazások tesztelése

Példa

Tervezés:



Grafikus felületű alkalmazások tesztelése

Példa

Megvalósítás (`tictactoemodeltest.cpp`*):*

```
void TicTacToeModelTest::testNewGame()
{
    _model->newGame();

        // ellenőrizzük, hogy kezdetben minden mező
        // üres és 0 a lépésszám
    QCOMPARE(_model->stepNumber(), 0);

    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            QCOMPARE(_model->getField(i, j),
                     TicTacToeModel::NoPlayer);
}
```