

Eseményvezérelt alkalmazások fejlesztése I

5. előadás

Grafikus felületű alkalmazások architektúrája

Giachetta Roberto

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

Grafikus felületű alkalmazások architektúrája

Architekturális tervezés

- *Szoftver architektúrának* nevezzük a szoftver fejlesztése során meghozott *elsődleges tervezési döntések* halmazát
 - azon döntések, amelyek megváltoztatása később jelentős újratervezést igényelné a szoftvernek
 - kihat a rendszer felépítésére, viselkedésére, kommunikációjára, nem funkcionális jellemzőire és megvalósítására
- A szoftver architektúra elsődleges feladata *a rendszer magas szintű felépítésének és működésének meghatározása*, a komponensek és relációk kiépítése
 - meghatározza a szolgáltatott és elvárt interfészek halmazát, a kommunikációs csatornákat és csatlakozási pontokat

Grafikus felületű alkalmazások architektúrája

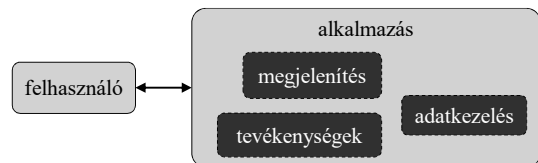
Architekturális tervezés

- A szoftver architektúráját különböző szempontok szerint közelíthetjük meg, pl.:
 - a szoftver által nyújtott szolgáltatások (funkciók) szerint
 - a felhasználó és a futtató platform közötti tevékenységi szint szerint
 - az adatátadás, kommunikáció módja szerint
- Az architektúra létrehozása során mintákra hagyatkozunk, a szoftver teljes architektúráját definiáló mintákat nevezzük *architekturális mintáknak (architectural pattern)*, az architektúra alkalmazásának módját, az egyes komponensek összekapcsolását segítik elő a *tervminták (design pattern)*

Grafikus felületű alkalmazások architektúrája

A monolitikus architektúra

- Minden szoftver rendelkezik architektúrával
- A legegyszerűbb felépítést az *monolitikus architektúra (monolithic architecture)* adja
 - nem különböztetjük meg az egyes feladatköröket (pl. megjelenítés, adatkezelés), hanem egységesen kezeljük őket



Grafikus felületű alkalmazások architektúrája

Példa

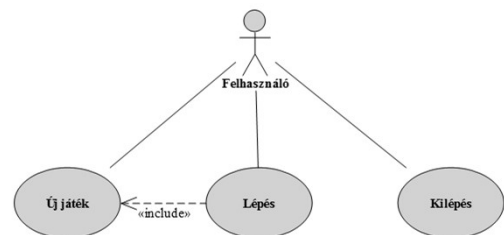
Feladat: Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

- a programban lehetőséget adunk új játék kezdésére, valamint lépésre (felváltva)
- a programban ‚X’ és ‚0’ jelekkel ábrázoljuk a két játékost
- a program automatikusan jelez, ha vége a játéknak (előugró üzenetben), majd új játékot kezd
- lehetőséget adunk, hogy a felhasználó bármikor új játékot indítson
- az alkalmazás felületét gombok segítségével valósítjuk meg (9 játékgomb, valamint új játék kezdése)

Grafikus felületű alkalmazások architektúrája

Példa

Tervezés (felhasználói esetek):



Grafikus felületű alkalmazások architektúrája

Példa

Tervezés (architektúra):

- az alkalmazást egy osztályban (`TicTacToeWidget`) valósítjuk meg, amely tartalmazza a grafikus felületet és a játék viselkedését
- a felületet a konstruktor és a `generateTable` segédművelet állítja elő, elrendezések segítségével
- a felületen elhelyezzük az új játék gombját (`newGameButton`), valamint a játéktábla gombjait (`gameTableButtons`), továbbá egy karakterrel eltároljuk az aktuális játékos jelét (`currentPlayerSymbol`)
- a játékot az eseménykezelők vezérlik

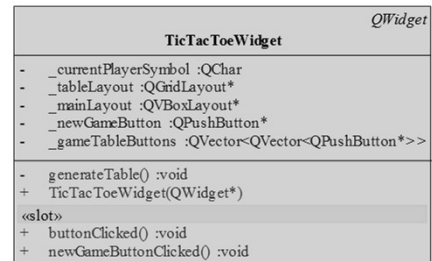
ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:7

Grafikus felületű alkalmazások architektúrája

Példa

Tervezés (architektúra):



ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:8

Grafikus felületű alkalmazások architektúrája

Példa

Megvalósítás (`tictactoewidget.cpp`):

```
void TicTacToeWidget::buttonClicked()
{
    // lekérjük az esemény küldőjét
    QPushButton* senderButton =
        qobject_cast<QPushButton*>(sender());
    int location = _tableLayout
        ->indexOf(senderButton);

    int x = location / 3;
    int y = location % 3;
    // a gomb rácson belüli pozíciója megadja a
    // koordinátákat
```

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:9

Grafikus felületű alkalmazások architektúrája

Példa

Megvalósítás (`tictactoewidget.cpp`):

```
_gameTableButtons[x][y]
->setText(_currentPlayerSymbol);
// megjelenítés a gombon

_gameTableButtons[x][y]->setEnabled(false);

if (_currentPlayerSymbol == 'X')
    // váltjuk a játékost
    _currentPlayerSymbol = '0';
else
    _currentPlayerSymbol = 'X';
...
}
```

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:10

Grafikus felületű alkalmazások architektúrája

A monolitikus architektúra korlátai

- Összetettebb alkalmazásoknál a monolitikus felépítés korlátozza a program
 - *áttekinthetőségét* (nehezen azonosítható, hol tároljuk a számításához szükséges adatokat)
 - *tesztelését* (nem ellenőrizhetjük külön-külön az egyes programfunkciókat)
 - *módosíthatóságát, bővíthetőségét* (a felület kinézetét csak úgy módosíthatjuk, ha a működést is átírjuk)
 - *újrafelhasználhatóságát* (a funkciók nem emelhetők ki és vihetők át másik alkalmazásba)

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:11

Grafikus felületű alkalmazások architektúrája

A monolitikus architektúra finomítása

- Ezért célszerű *a program felépítését felbontani*
 - *a funkciók mentén*: válasszuk szét a különböző tevékenységeket, emeljünk ki a tényleges tevékenységeket külön alprogramba
 - pl.: a játékbeli lépést helyezhetjük külön alprogramba, így függetlenedik az eseménykezelőtől
 - *az adatok mentén*: ne közvetlenül a felületen tárolt információkkal dolgozzuk, hanem külön adatokkal, amelyek függetlenek a megjelenítéstől
 - pl. a játéktábla értékeit ábrázoljuk egész számokkal, ahelyett, hogy a grafikus elemek feliratát használnánk

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:12

Grafikus felületű alkalmazások architektúrája

Példa

Feladat: Módosítsuk a Tic-Tac-Toe programot úgy, hogy áttekinthetőbb és tagoltabb legyen.

- a játékosok reprezentálhatóak számokkal (1: X, 5: O, 0: még nincs érték)
- a játékban tárolt értékeket egy külön mátrixban (`_gameTable`) tároljuk, a az aktuális játékost is számként ábrázoljuk (`_currentPlayer`)
- elmentjük a lépések számát (`_stepNumber`), így nem kell állandóan ellenőrizni a mezőket
- új metódusokat veszünk fel a játék kezelésére (`newGame`, `stepGame`, `isGameWon`)

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:13

Grafikus felületű alkalmazások architektúrája

Példa

Tervezés (architektúra):

TicTacToeWidget	QWidget
- _tableLayout :QGridLayout*	
- _mainLayout :QVBoxLayout*	
- _newGameButton :QPushButton*	
- _gameTableButtons :QVector<QPushButton*>	
- _stepNumber :int	
- _currentPlayer :int	
- _gameTable :int**	
- generateTable() :void	
- isGameWon() :void	
- newGame() :void	
- stepGame(int, int) :void	
+ TicTacToeWidget(QWidget*)	
«slot»	
+ buttonClicked() :void	
+ newGameButtonClicked() :void	

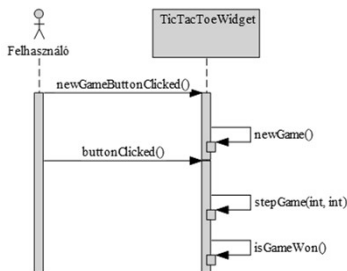
ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:14

Grafikus felületű alkalmazások architektúrája

Példa

Tervezés (viselkedés):



ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:15

Grafikus felületű alkalmazások architektúrája

Példa

Megvalósítás (tictactoewidget.cpp):

```
void TicTacToeWidget::buttonClicked()
{
    QPushButton* senderButton =
        qobject_cast<QPushButton*>(sender());

    int location = _tableLayout
        ->indexOf(senderButton);

    stepGame(location / 3, location % 3);
}
```

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:16

Grafikus felületű alkalmazások architektúrája

Példa

Megvalósítás (tictactoewidget.cpp):

```
void TicTacToeWidget::stepGame(int x, int y) {
    _gameTable[x][y] = _currentPlayer;
    // pozíció rögzítése

    if (_currentPlayer == 1)
        _gameTableButtons[x][y]->setText("X");
    else
        _gameTableButtons[x][y]->setText("O");
    _gameTableButtons[x][y]->setEnabled(false);

    _stepNumber++;
    _currentPlayer = _currentPlayer % 2 + 1;
    ...
}
```

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:17

Grafikus felületű alkalmazások architektúrája

A modell/nézet architektúra

- A programszerkezet felépítése akkor ideális, ha teljesen külön programegységbe tudjuk leválasztani a felhasználói felülettel kapcsolatos részeket a ténylegesen a feladat megoldását szolgáló funkcionálisról
- Ezt a felbontást követve jutunk el a *modell/nézet (MV, model-view)* architektúrához, amelyben
 - a *modell* tartalmazza a feladat végrehajtását szolgáló programegységeket, az állapotkezelést, valamint az adatkezelést, ezt nevezzük *alkalmazáslogikának*, vagy *üzleti logikának*
 - a *nézet* tartalmazza a grafikus felhasználói felület megvalósítását, a felület elemeit és az eseménykezelőket

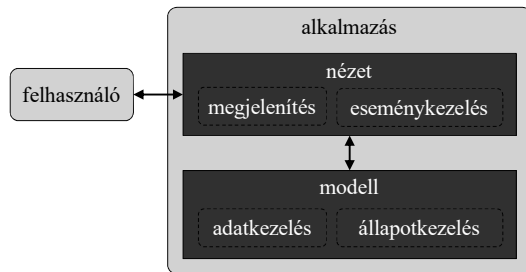
ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:18

Grafikus felületű alkalmazások architektúrája

A modell/nézet architektúra

- a felhasználó a nézettel kommunikál, a modell és a nézet egymással



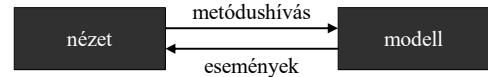
ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:19

Grafikus felületű alkalmazások architektúrája

A modell/nézet architektúra

- A modell és a nézet kapcsolatát úgy kell megvalósítani, hogy
 - a nézet ismerheti a modell felületét (interfészét), és hívhatja annak (publikus) műveleteit
 - a modellnek semmilyen tudomása sem lehet a nézetről, ezért nem hívhatja annak műveleteit, de eseményeken keresztül kommunikálhat vele



- A megvalósításban a nézet hivatkozhat a modellre (tartalmazhatja annak példányát)

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:20

Grafikus felületű alkalmazások architektúrája

Példa

Feladat: Módosítsuk a Tic-Tac-Toe programot úgy, hogy kétrétegű architektúrában valósuljon meg.

- a játékvért felelős programrészeket kiemeljük egy új osztályba, amely a modellt valósítja meg (**TicTacToeModel**), itt csak egész számokkal dolgozunk, függetlenül a felülettől, a játékműveletek publikusak lesznek
- a modellt megfelelő ellenőrzésekkel kell ellátni (mivel leválasztottuk a tevékenységeit)
- a nézet (**TicTacToeWidget**) aggregálja a modellt, és biztosítja a grafikus megjelenítést, valamint a játék műveleteinek hívását, az eredmények megjelenítését

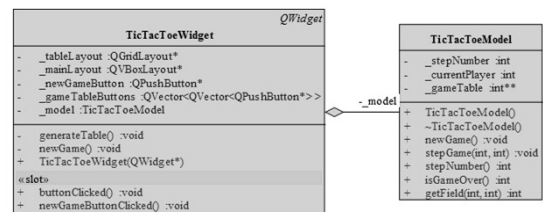
ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:21

Grafikus felületű alkalmazások architektúrája

Példa

Tervezés (architektúra):



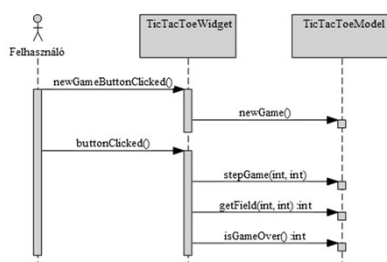
ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:22

Grafikus felületű alkalmazások architektúrája

Példa

Tervezés (viselkedés):



ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:23

Grafikus felületű alkalmazások architektúrája

Példa

Megvalósítás (tictactoeomodel.cpp):

```
void TicTacToeModel::stepGame(int x, int y)
{
    // ellenőrzzük lépésszámot, tartományt, mezőt
    if (_stepNumber >= 9) return;
    if (x < 0 || x > 2 || y < 0 || y > 2) return;
    if (_gameTable[x][y] != 0) return;

    _gameTable[x][y] = _currentPlayer;
    // pozíció rögzítése
    _stepNumber++;
    _currentPlayer = _currentPlayer % 2 + 1;
}
```

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:24

Grafikus felületű alkalmazások architektúrája

Példa

Megvalósítás (tictactoewidget.cpp):

```
void TicTacToeWidget::buttonClicked() {
    ...
    _model.stepGame(x, y); // játék léptetése

    if (_model.getField(x, y) == 1)
        _gameTableButtons[x][y]->setText("X");
    else
        _gameTableButtons[x][y]->setText("0");
    _gameTableButtons[x][y]->setEnabled(false);

    int won = _model.isGameOver();
    // játék végének ellenőrzése
    ...
}
```

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:25

Grafikus felületű alkalmazások architektúrája

Egyedi események

- A saját osztályainkban lehetőségünk van események és eseménykezelők létrehozására, az események kiváltására
 - az események olyan visszatérési érték nélküli metódusok, amelyeket csak deklarálnunk kell
 - az eseményeket egy `QObject` leszármazott (és `QObject` makróval ellátott) osztály `signals` részében helyezzük el
 - az események mindig publikusak
 - eseményeket az `<eseménynév>` (`<paraméterek>`) utasítással válthatunk ki az osztályon belül
 - az eseményt ugyanabban, vagy más osztályban is lekezelhetjük

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:26

Grafikus felületű alkalmazások architektúrája

Egyedi események

• Pl.:

```
class SignalDemoClass : public QObject
{
    Q_OBJECT // makróval megjelölt
public:
    void someMethod();
    signals: // saját események
    void demoSignal(); // csak deklaráljuk
};

void SignalDemoClass::someMethod()
{
    demoSignal(); // kiváltjuk az eseményt
}
```

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:27

Grafikus felületű alkalmazások architektúrája

Példa

Feladat: Módosítsuk a Tic-Tac-Toe programot úgy, hogy a modell is kommunikáljon a nézettel, események segítségével.

- a modellben megvalósítunk 3 eseményt amelyeket a megfelelő pontokon kiváltunk:
 - mező megváltozása (`fieldChanged`)
 - játék vége valamely játékos győzelmével (`gameWon`)
 - játék vége döntetlennel (`gameOver`)
- a nézetben eseménykezelőket kötünk rájuk, így egyszerűsödik a játéklépés megvalósítása, mivel a további tevékenységek (mező átírása, játék vége) az események hatására futnak le

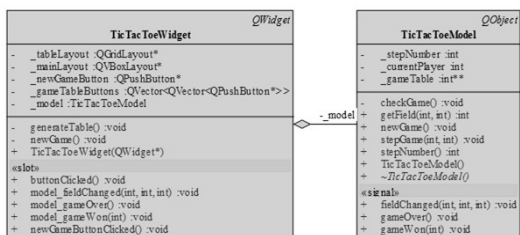
ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:28

Grafikus felületű alkalmazások architektúrája

Példa

Tervezés (architektúra):



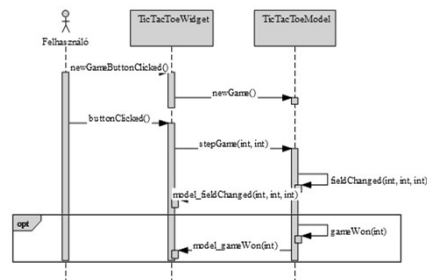
ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:29

Grafikus felületű alkalmazások architektúrája

Példa

Tervezés (viselkedés):



ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:30

Grafikus felületű alkalmazások architektúrája

Példa

Megvalósítás (tictactoeomodel.cpp):

```
void TicTacToeModel::checkGame()
{
    int won = 0;
    ...
    if (won > 0) // ha valaki győzött
    {
        gameWon(won); // esemény kiváltása
    }
    else if (_stepNumber == 9) // döntetlen játék
    {
        gameOver(); // esemény kiváltása
    }
}
```

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:31

Grafikus felületű alkalmazások architektúrája

Példa

Megvalósítás (tictactoewidget.cpp):

```
void TicTacToeWidget::buttonClicked()
{
    ...
    _model.stepGame(x, y); // játék léptetése
}
...
void TicTacToeWidget::model_gameOver()
{
    QMessageBox::information(this,
        trUtf8("Játék vége!"),
        trUtf8("A játék döntetlen lett!"));
    _model.newGame();
}
```

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:32

Grafikus felületű alkalmazások architektúrája

A modell megvalósítása

- Mivel modell független a nézettől, és újrahasznosítható, nem tudható előre, milyen módon, milyen körülmények között hívják meg műveleteit
 - a hívás paramétereit, a modell állapotát *ellenőrizni kell* a tevékenységek végrehajtása előtt
 - pl. lépés előtt megnézzük, hogy az végrehajtható-e
- A modell és a nézet közötti kommunikációban mindkét irányban törekedni kell az egyértelműsége és a lehető legkevesebb hibalehetősége
 - pl. korlátozott értékhasználatra használunk felsoroló típusokat (**enum**)

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:33

Grafikus felületű alkalmazások architektúrája

Példa

Feladat: Módosítsuk a Tic-Tac-Toe programot úgy, hogy a modellben a játékos és a szimbólumokat enumeráció segítségével valósítsuk meg.

- felvesszük a játékos (**Player**) felsoroló típust beágyazott típusként három lehetséges értékkel (**NoPlayer**, **PlayerX**, **PlayerO**)
- a játéktábla reprezentációját, valamint a metódusok, események paramétereit is ennek megfelelően alakítjuk át (eseményeknél a teljes elérési útvonalat meg kell adnunk, azaz a külső osztályt is)
- a nézet az eseményparaméter függvényében **switch** elágazással dönt a tevékenységről

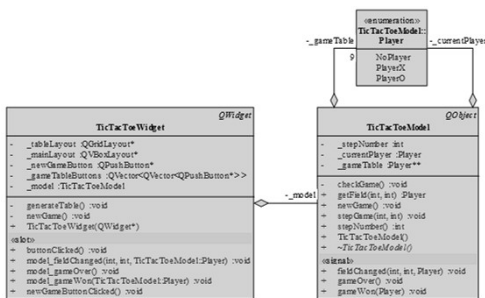
ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:34

Grafikus felületű alkalmazások architektúrája

Példa

Tervezés (architektúra):



ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:35

Grafikus felületű alkalmazások architektúrája

Példa

Megvalósítás (tictactoeomodel.cpp):

```
void TicTacToeModel::stepGame(int x, int y){
    ...
    _gameTable[x][y] = _currentPlayer;
    // pozíció rögzítése
    fieldChanged(x, y, _currentPlayer);
    // jelezzük egy eseménykiváltással, hogy
    // változott a mező
    _stepNumber++;
    _currentPlayer = (Player) (_currentPlayer % 2 +
        1);
    // egészként kezelhető az érték, de
    // konvertálnunk kell
    ...
}
```

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:36

Grafikus felületű alkalmazások architektúrája

Példa

Megvalósítás (tictactoewidget.cpp):

```
void TicTacToeWidget::model_fieldChanged(int x,
int y, TicTacToeModel::Player player)
{
    switch (player)
    {
    case TicTacToeModel::PlayerX:
        _gameTableButtons[x][y]->setText("X");
        _gameTableButtons[x][y]->setEnabled(false);
        break;
    ...
    }
}
```

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:37

Grafikus felületű alkalmazások architektúrája

Példa

Feladat: Készítsünk új nézetet a Tic-Tac-Toe játékhoz, amelyben nem karakterekkel jelenítjük meg a játékkállást, hanem alakzatokat rajzolunk.

- a képernyőről levesszük az összes vezérlőt, a megjelenítését rajzolás segítségével (QPainter) valósítjuk meg
- a játékosok ismét egér segítségével foglalhatják el a mezőket, új játékot pedig a **ctrl+N** billentyűkombinációval indíthatnak
- ehhez felüldefiniáljuk a billentyű- és egérlenyomás eseménykezelőket (keyPressEvent, mousePressEvent)
- a modellen semmit sem kell változtatnunk

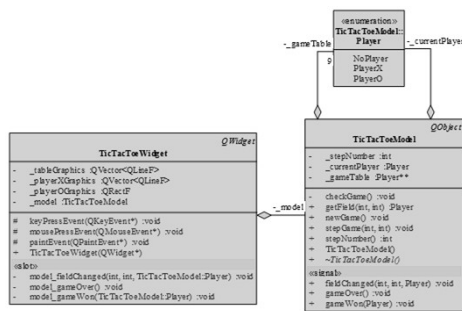
ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:38

Grafikus felületű alkalmazások architektúrája

Példa

Tervezés (architektúra):



ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:39

Grafikus felületű alkalmazások architektúrája

Példa

Megvalósítás (tictactoewidget.cpp):

```
void TicTacToeWidget::keyPressEvent(QKeyEvent
*event)
{
    if (event->key() == Qt::Key_N &&
        QApplication::keyboardModifiers() ==
        Qt::ControlModifier)
    {
        // lekezeljük a Ctrl+N kombinációt
        _model.newGame();
        update();
    }
}
```

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:40

Grafikus felületű alkalmazások architektúrája

Példa

Megvalósítás (tictactoewidget.cpp):

```
void TicTacToeWidget::mousePressEvent(QMouseEvent
*event)
{
    // az event->pos() megadja az egérpozíciót,
    // ami QPoint típusú, ebből kiszámolható,
    // melyik mezőn vagyunk:
    int x = event->pos().x() * 3 / width();
    int y = event->pos().y() * 3 / height();

    _model.stepGame(x, y); // játék léptetése
}
```

ELTE IK, Eseményvezérelt alkalmazások fejlesztése I

5:41