Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások fejlesztése I

6. előadás

Összetett alkalmazások megvalósítása

Giachetta Roberto

A jegyzet az ELTE Informatikai Karának
2014. évi Jegyzetpályázatának támogatásával készült

Összetett alkalmazások megvalósítása

Ablakok

- A grafikus felületű alkalmazásokban a vezérlőket ablakokra helyezzük
  - ablaknak minősül bármely vezérlő, amely egy **`QWidget`**, vagy bármely leszármazottjának példánya, és nincs szülője
  - adottak speciális ablaktípusok is, pl.:
    - *üzenőablak* (**`QMessageBox`**), elsősorban üzenetek közlésére, vagy kérdések feltételére
    - *dialógusablak* (**`QDialog`**), amelynek eredménye van, elfogadható (**`accept`**), vagy elutasítható (**`reject`**)
    - *főablak* (**`QMainWindow`**), amely számos kiegészítést biztosít összetett ablakok megvalósítására

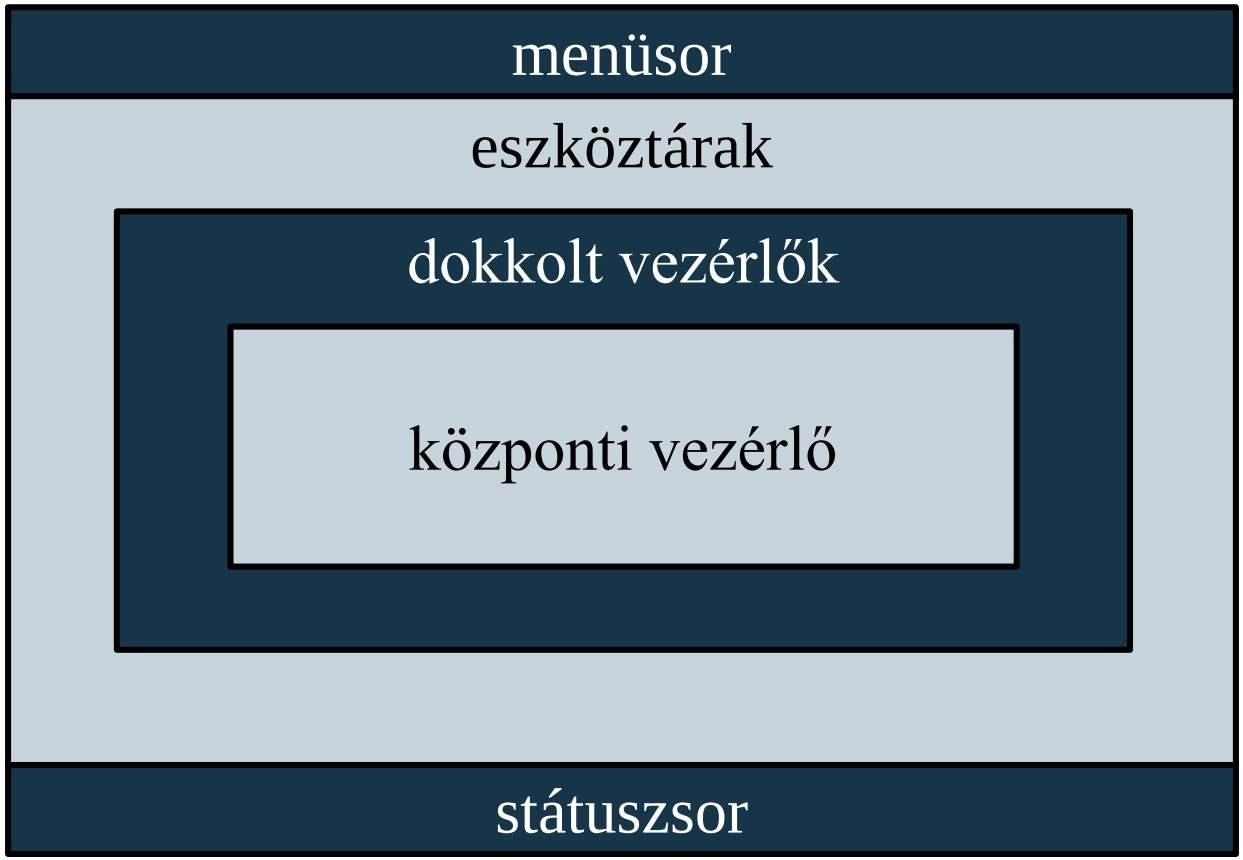
Összetett alkalmazások megvalósítása

Főablakok

- A *főablak* (**`QMainWindow`**) egy olyan speciális ablaktípus, amely megkönnyíti összetett, speciális vezérlőket tartalmazó ablakok létrehozását, úgymint
  - *menüsor* (*Menu Bar*): menüpontok gyűjteménye az ablak tetején
  - *státuszsor* (*Status Bar*): állapotkijelző sor az ablak alján
  - *eszköztár* (*Toolbar*): ikongyűjteményeket tartalmazó funkciógombok, amely az ablak bármely szélére elhelyezhetőek
- Az ablakon belül további vezérlőket helyezhetünk el, amelyeket dokkolhatunk az ablak széléhez, vagy középre

Összetett alkalmazások megvalósítása

Főablakok



Összetett alkalmazások megvalósítása

Akciók

- A különböző vezérlők sokszor ugyanazon funkciókat biztosítják más formában (ikon, szöveg, …)
- A funkciókat egységesen *akció*ként (**`QAction`**) kezelhetjük, amely
  - rendelkezik felirattal (**`text`**), ikonnal (**`icon`**), gyorsbillentyűvel (**`shortcut`**), segédüzenettel (**`statusTip`**)
  - lehetőséget ad kijelölésre (**`checked`**), valamint billentyűs gyorsnavigálásra (az **`&`** karakterrel)
  - kiváltható billentyűzettel vagy egérrel, a kiváltást esemény (**`triggered`**)
  - felhelyezhető tetszőleges menüre, illetve eszköztárra

Összetett alkalmazások megvalósítása

Akciók

- pl.:

```
QAction newAct = new QAction(QIcon("new.png"),
                             tr("Ú&j"), this);
   // ikon és név megadása, a j billentyűre
   // gyorsnavigál a menüben
newAct->setShortcuts(QKeySequence::New);
   // a keretrendszer által kirendelt "új"
   // billentyűkombináció
newAct->setStatusTip(tr("Új fájl létrehozása"));
connect(newAct, SIGNAL(triggered()),
        this, SLOT(newFile()));
   // eseménykezelő társítás
fileMenu->addAction(newAct); // felhelyezés
fileToolBar->addAction(newAct);
```

Összetett alkalmazások megvalósítása

Menü

- A menüt (**`QMenu`**) a főablak **`menuBar`** tulajdonságán keresztül kezelhetjük, a menühöz felvehetünk almenüket, akciókat és elválasztókat (**`separator`**)
  - a menük tetszőlegesen egymásba ágyazhatóak
  - pl.:

```
QMenu fileMenu = this->menuBar()
   ->addMenu(tr("&Fájl"));
   // új almenü létrehozása
fileMenu->addAction(newAct);
   // menüpont felvétele
fileMenu->addSpearator(); // elválasztó
fileMenu->addMenu(tr("&Legutóbbi fájlok"));
   // beágyazott almenü
```

Összetett alkalmazások megvalósítása

Eszköztár

- Eszköztárakból (**`QToolBar`**) tetszőlegesen sokat vehetünk fel, amelyek alapértelmezetten az ablak tetején jelennek meg
  - ikonok sorozatát adják, esetleges elválasztókkal szeparálva
  - az eszköztárak alapértelmezés szerint utólag áthelyezhetőek bármely szélére az ablaknak, illetve lehet lebegő (**`floating`**) állapotban is
  - pl.:

```
QToolBar fileBar = this->addToolBar(tr("Fájl"));
   // új eszköztár felvétele
fileBar->addAction(newAct);
   // új akció felvétele
fileBar->addSpearator(); // elválasztó
```

Összetett alkalmazások megvalósítása

Státuszsor és tartalom

- A státuszsor (**`QStatusBar`**) alapvetően státuszüzenetek kiírására szolgál, ugyanakkor bármilyen vezérlő ráhelyezhető
  - üzenetet kiírni a **`showMessage(<`*`üzenet`*`>)`** utasítással tudunk, törölni a **`clearMessage()`** utasítással
  - pl.: **`this->statusBar()->showMessage(tr("Kész"));`**
- Az ablak területére célszerű egy külön vezérlőben elhelyezni a tartalmat, ez a központi vezérlő (**`centralWidget`**)
- Amennyiben több tartalmat helyeznénk az ablakra, lehetőségünk van azokat dokkolni a **`QDockWidget`** osztály segítségével, amelyet az **`addDockWidget(<`*`vezérlő`*`>)`** művelettel helyezhetünk az ablakra

Összetett alkalmazások megvalósítása

Egyedi tulajdonságok

- A tulajdonságok célja egyszerűsített hozzáférés adott mezőértékekhez, lekérdező és beállító műveletek segítségével
  - lehetőségünk van saját tulajdonságok létrehozására a **Q_PROPERTY** makró segítségével, megadjuk típusát és nevét, valamint az író (**WRITE**) és olvasó (**READ**) műveleteket
- Pl.:

```
class MyObject : public QObject {
   Q_OBJECT
   Q_PROPERTY(int prop READ prop WRITE setProp)
   …
   int prop(); // olvasó művelet
   void setProp(int value); // író művelet
   …
```

Összetett alkalmazások megvalósítása

Alkalmazásszintű tulajdonságok

- A Qt alkalmazásokat minden esetben egy alkalmazás (**`QApplication`**) objektum vezérli, amely számos értéket tárol, úgymint:
  - alkalmazás információk (**`applicationName`**, **`orgnizationName`**, **`applicationVersion`**)
  - környezeti információk (**`applicationDirPath`**, **`arguments`**, **`keyboardModifiers`**, **`clipboard`**)
  - grafikus környezeti adatok (**`allWindows`**, **`windowIcon`**, **`palette`**, **`styleSheet`**, **`font`**)
- Az alkalmazás értékeihez bárhonnan, statikus műveletekkel hozzáférhetünk

Összetett alkalmazások megvalósítása

Alkalmazásszintű tulajdonságok

- Pl.:

```
QApplication::setOrganizationName("MySoft");
QApplication::setApplicationName("MyApp");
   // beállítunk némi információt
…
QString executableName =
   QApplication::arguments()[0];
   // lekérjük a programnevet
if (QApplication::arguments().size() > 1)
{
   // ha még van ezen felül argumentum
   QString arg1 = QApplication::arguments()[1];
}
```

Összetett alkalmazások megvalósítása

Beállítások kezelése

- Nagyobb alkalmazások rendszerint rendelkeznek külön alkalmazásszintű *beállítások*kal, amelyek célszerű elmentenünk, és újabb futtatáskor betöltenünk
- A beállítások eltárolhatóak egyedileg, de használhatjuk a beépített **`QSettings`** osztályt, amely egyszerűsíti a beállítások kezelését
  - a beállítások eltárolásának módja platformonként változik (Linux esetén konfigurációs fájlok, Windows esetén regisztrációs adatbázis), ezt az osztály elfedi, így a programozónak a tárolás módjával nem kell törődnie
  - a beállítások egy adott alkalmazásra és felhasználóra vonatkoznak

Összetett alkalmazások megvalósítása

Beállítások kezelése

- a beállításokba kulcs/érték párokat vehetünk fel a **`setValue(<kulcs>,<érték>)`** utasítással, ahol a kulcs szöveges, az érték tetszőleges **`QVariant`** lehet, lekérdezni a **`value(<kulcs>)`** utasítással tudunk
- a **`contains(<kulcs>)`** függvény ellenőrzi a kulcs létezését

- A **`QVariant`** egy általános típus, amely a primitív típusokat tudja „becsomagolni”, így az ottani tartalom rendelkezik több konverziós művelettel, pl.:

```
Qvariant vi(123); // létrehozás egészből
int i = vi.toInt(); // visszaalakítás egészre
Qvariant vc = QColor(15, 20, 200); // színből
QColor c = vc.value<QColor>(); // vissza színbe
```

Összetett alkalmazások megvalósítása

Beállítások kezelése

- Pl.:

```
QString value;
…
QSettings settings("MySoft", "MyApp");
    // beállítások létrehozása (ha korábban
    // beállítottuk az alkalmazás információkat,
    // használhatunk alapértelmezett konstruktort)
settings.setValue("myValue", value);
    // érték beállítása
…
settings.value("myValue").toString();
    // visszakérjük az értéket és szöveggé
    // alakítjuk
```

Összetett alkalmazások megvalósítása

Erőforrások

- A főablakon használt akciókat célszerű ellátni ikonokkal, amelyeket az alkalmazáshoz kell, hogy csatoljunk
- Az alkalmazáshoz használt ikonokat és egyéb nem kód tartalmat lehetőségünk van *erőforrás*ként (*resource*) csatolni az alkalmazáshoz
  - az erőforrások tartalma belefordul a futtatandó állományba, így nem kell külön másolni őket
  - az erőforrásokat a projekthez tartozó **`.qrc`** fájlban nevezhetjük meg
  - az erőforrásként megadott fájlokat a **`:<elérési útvonal>`** hivatkozással hívhatjuk be, pl.:

```
QIcon(":/images/new.png"); // erőforrás elérése
```

Összetett alkalmazások megvalósítása

Példa

Feladat: Készítsünk egy *Memory* kártyajátékot, amelyben két játékos küzd egymás ellen. A játékmezőn kártyapárok találhatóak, és a játékosok feladata ezek megtalálása.

- a játék különböző kártyacsomagokkal játszható, amelyek könyvtárakból tölthetőek be, minden ilyen könyvtárban található egy **`name.txt`**, ami a csomag nevét tartalmazza, és tetszőleges számú kép (ezek a kártyák), valamint egy hátlap (**`back`** fájlnévvel)
- lehetőségünk van egy beállító ablakban megadni a kiválasztott kártyacsomagot, valamint a játéktábla méretét (csak páros méretű, de legalább 4 kártyából álló lehet), valamint a játékosok neveit

Összetett alkalmazások megvalósítása

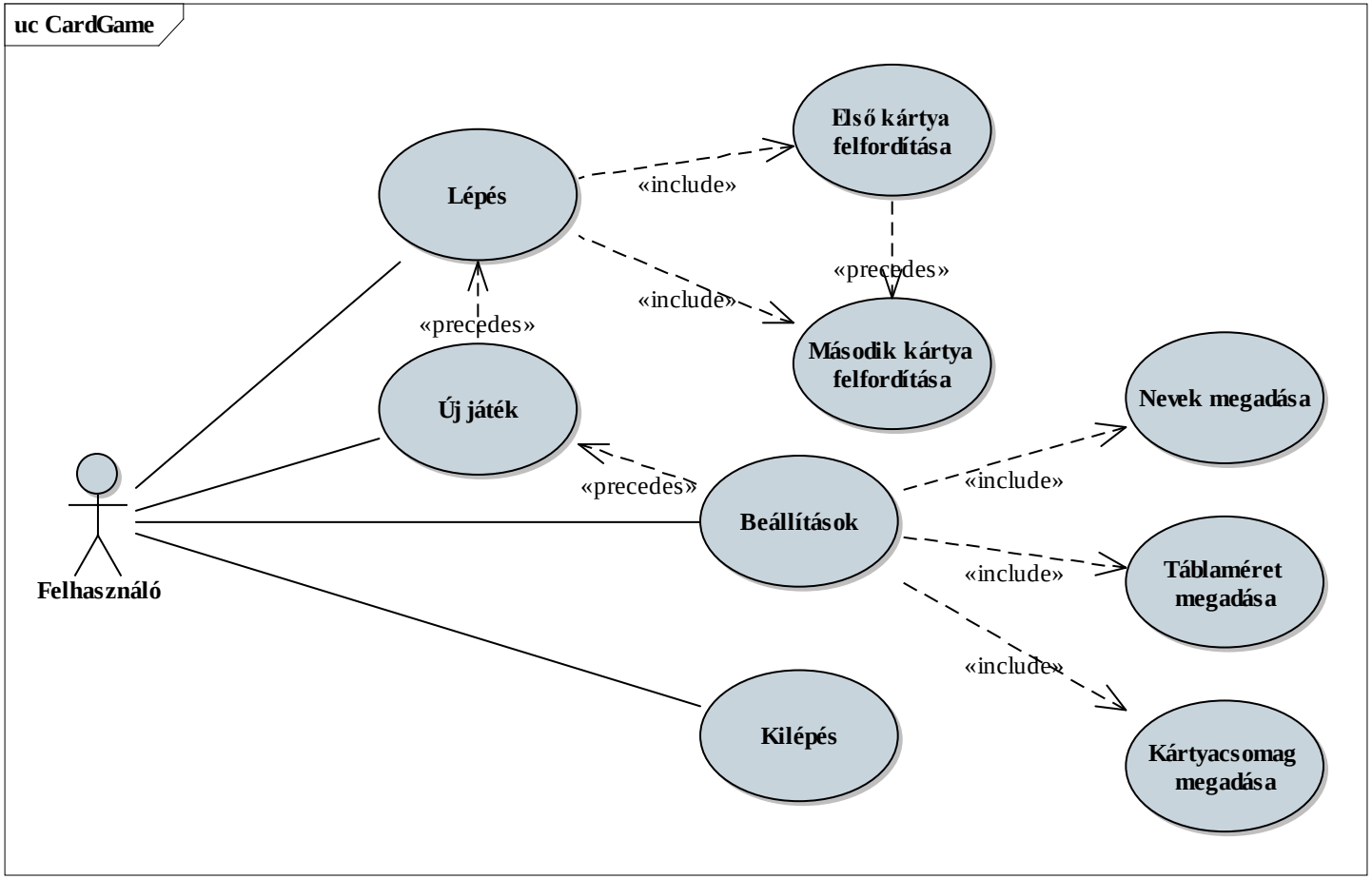
Példa

- kezdetben minden kártya le van fordítva, a játékosok felváltva lépnek, minden lépésben felfordíthatnak két kártyát
- amennyiben a kártyák egyeznek, úgy felfordítva maradnak és a játékos ismét léphet, különben 1 másodperc múlva visszafordulnak, és a másik játékos következik
- a játékot az nyeri, aki több kártyapárt talált meg
- megnyert játékok számát göngyölítve jelenítjük meg, amíg új játékosokat nem állítunk be
- a felületen folyamatosan megjelenítjük a játékosok adatait (sikeres, sikertelen lépések száma, megnyert játszmák száma)

Összetett alkalmazások megvalósítása

Példa

Felhasználói esetek:



Összetett alkalmazások megvalósítása

Példa

Tervezés (architektúra):

- a játékot kétrétegű architektúrában valósítjuk meg
- a modell tartalmazza:
  - magát a játékot, amit egy kezelőosztály felügyel (**`GameManager`**), valamint hozzá segédosztályként a játékost (**`Player`**)
  - a kártyacsomagokat (**`CardPack`**)
- a nézet tartalmazza:
  - a játék főablakát (**`MainWindow`**), amely tartalmaz egy menüt és egy státuszsort
  - a beállítások segédablakát (**`ConfigurationDialog`**)

Összetett alkalmazások megvalósítása

Példa

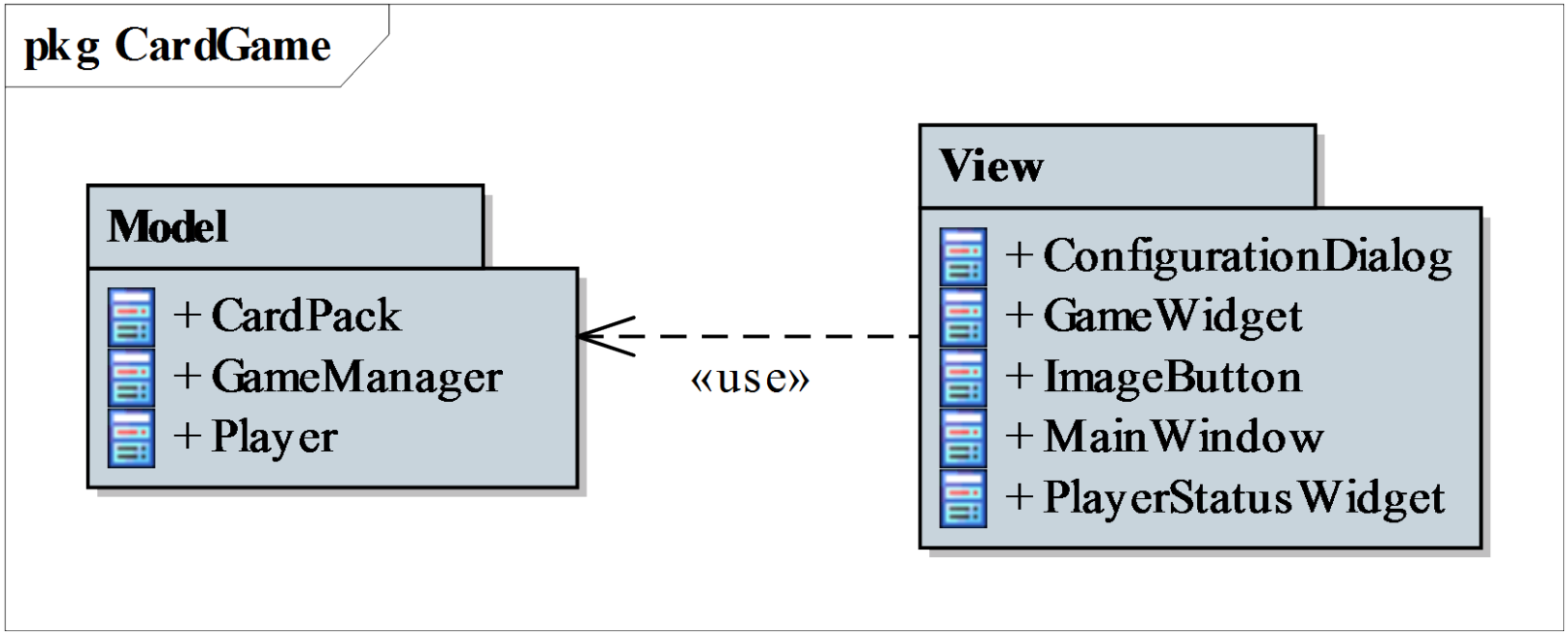
- a játékfelületet megjelenítő vezérlőt (**`GameWidget`**), amely tartalmazza a játékmezővel kapcsolatos tevékenységeket
- ehhez segédosztályként a felhasználói információkat kiíró vezérlőt (**`PlayerStatusWidget`**, ezt előléptetett vezérlővel állítjuk be a felülettervezőben), valamint a képet megjeleníteni tudó egyedi gombot (**`ImageButton`**)

- a nézet a modell publikus műveleteit hívja, és eseményeket is kaphat tőle
- egy csomag kártyát erőforrásként csatolunk az alkalmazáshoz (**`packs.qrc`**), hogy mindig legyen legalább egy csomag kártya

Összetett alkalmazások megvalósítása

Példa

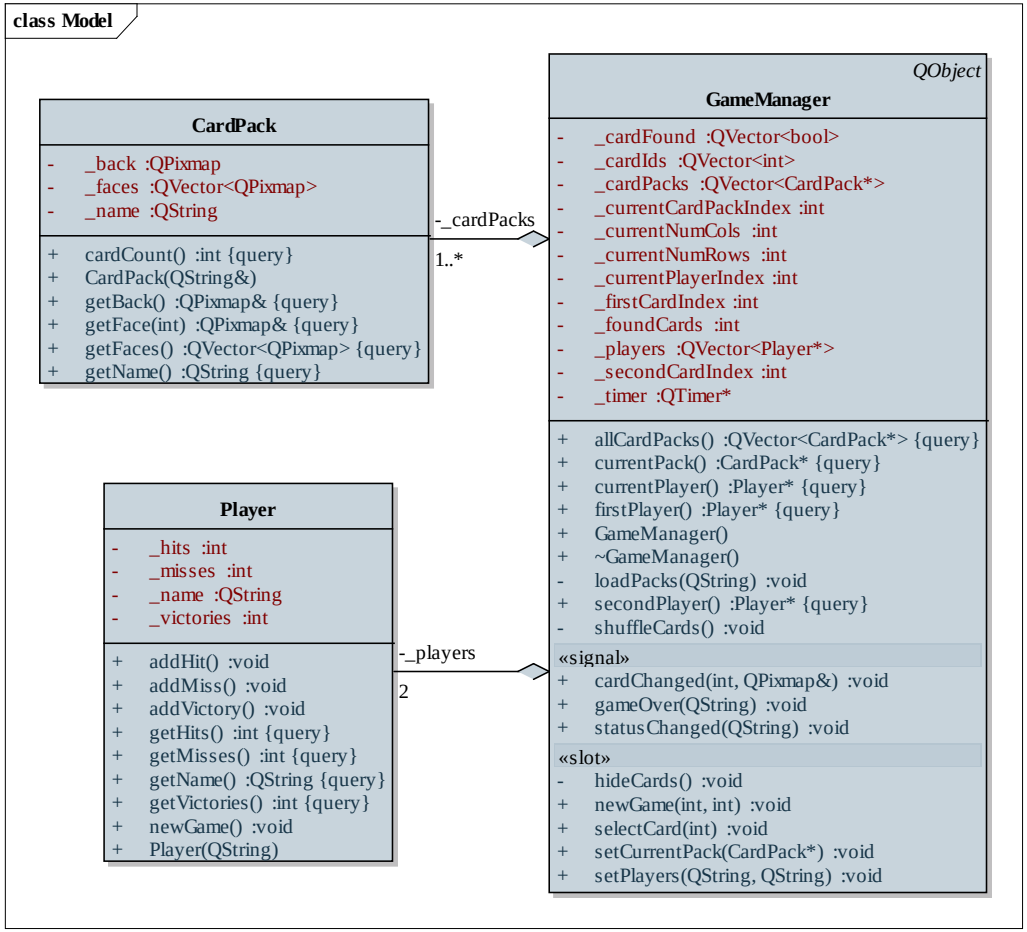
Tervezés (architektúra):



Összetett alkalmazások megvalósítása

Példa

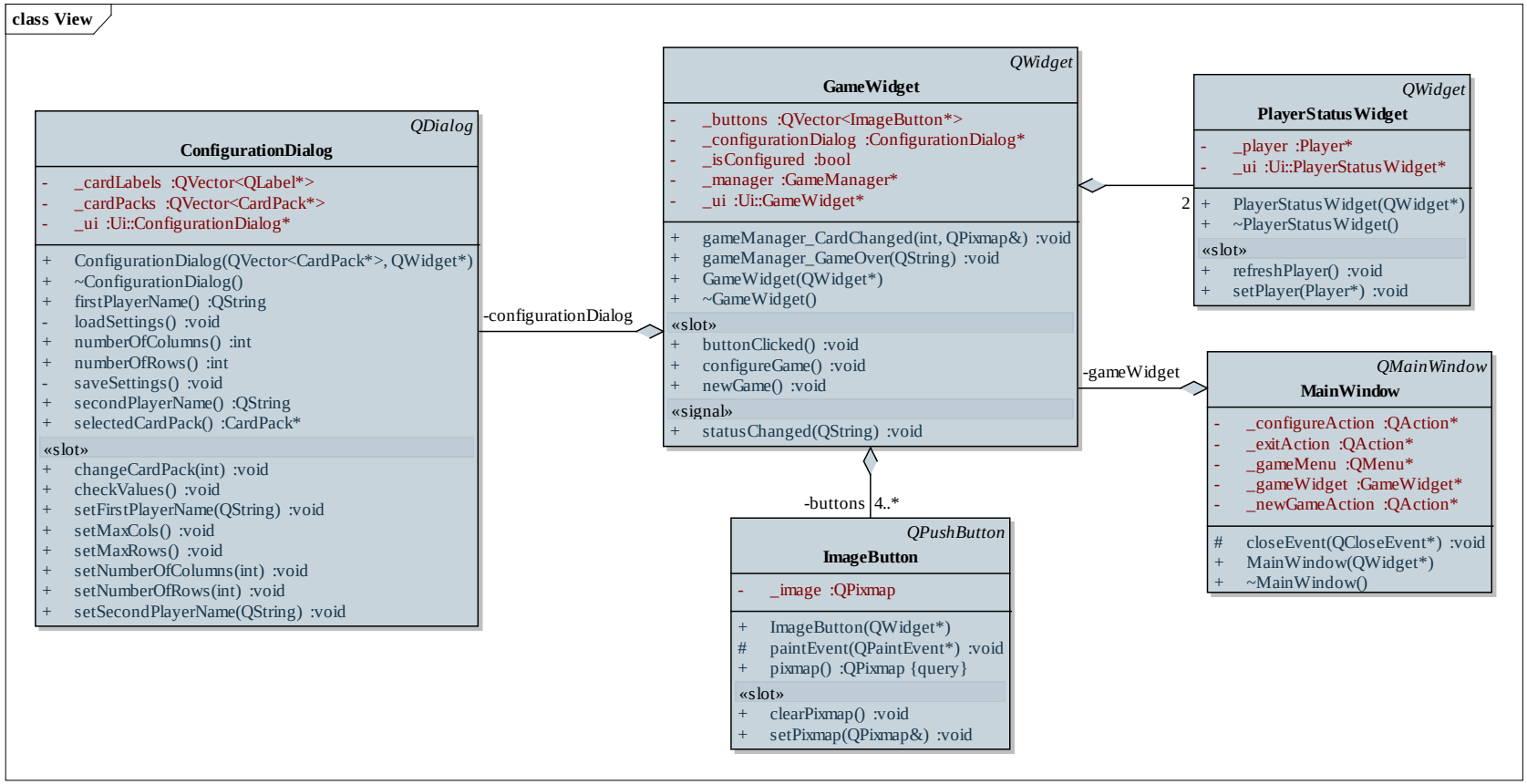
Tervezés (modell):



Összetett alkalmazások megvalósítása

Példa

Tervezés (nézet):



Összetett alkalmazások megvalósítása

Példa

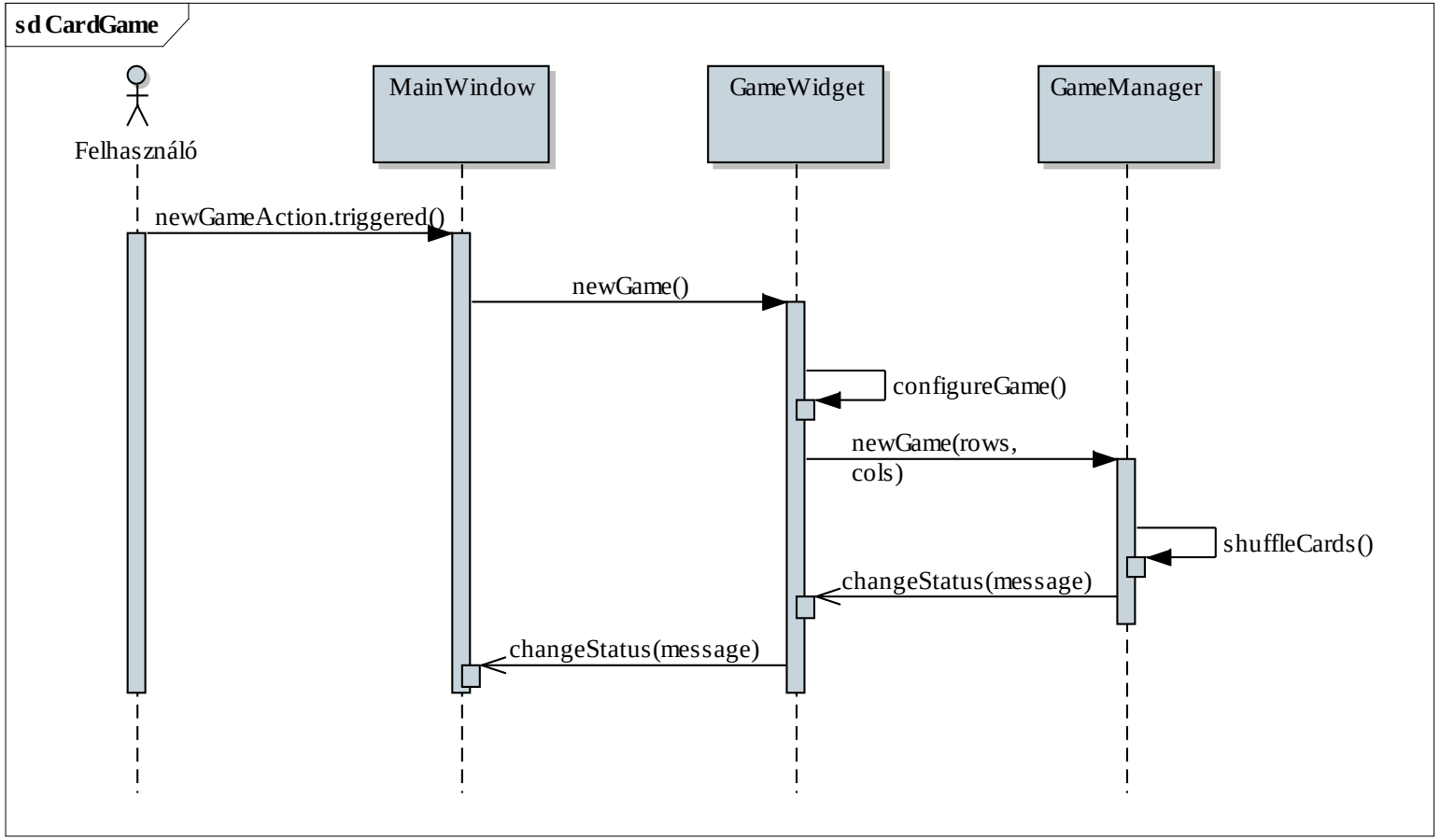
Tervezés (dinamikus):

- a kommunikáció a főablaknál indul (pl. **`newGameAction`**), amely tevékenységeit a játékkezelő vezérlőn át végzi (**`newGame`**), amely továbbítja a megfelelő formában a játékkezelőnek, amely végrehajtja a tevékenységet a logikai szinten (**`newGame(<szélesség>, <magasság>)`**)
- amennyiben a tevékenység változást eredményez, a logikai réteg esemény (pl. **`statusChanged(<üzenet>)`**, **`gameOver(<üzenet>)`**) segítségével jelzi a játékkezelő vezérlőnek a változtatást, amely esetlegesen továbbjelzi azt a főablaknak (újabb esemény kiváltásával)

Összetett alkalmazások megvalósítása

Példa

Tervezés (dinamikus):



Összetett alkalmazások megvalósítása

Példa

Megvalósítás (`mainwindow.cpp`*):*

```
…
MainWindow::MainWindow(){
    …
    connect(newGameAction, SIGNAL(triggered()),
       gameWidget, SLOT(newGame()));
    connect(configureAction, SIGNAL(triggered()),
       gameWidget, SLOT(configureGame()));
    …
    connect(gameWidget, SIGNAL(statusChanged(
       QString)), this->statusBar(),
       SLOT(showMessage(QString)));
       // állapotváltás a játékban
}
```

Összetett alkalmazások megvalósítása

Példa

*Megvalósítás (**gamewidget.cpp**):*

```
…
GameWidget::GameWidget(QWidget *parent) : … {
    …
    connect(manager,
        SIGNAL(statusChanged(QString)), this,
        SIGNAL(statusChanged(QString)));
        // a logikai réteg eseménye egy újabb
        // eseményt vált ki
    connect(manager,
        SIGNAL(statusChanged(QString)),
        ui->firstPlayerStatus,
        SLOT(refreshPlayer()));
    …
```

Összetett alkalmazások megvalósítása

Példa

Megvalósítás (`gamemanager.h`*):*

```
…
class GameManager : public Qobject {
   Q_OBJECT
   // tulajdonságok beállítása:
   Q_PROPERTY(CardPack* currentPack
      READ currentPack WRITE setCurrentPack)
   Q_PROPERTY(Player* currentPlayer
      READ currentPlayer)
public:
   explicit GameManager();
      // nincs másoló konstruktor és értékadás
   ~GameManager();
   …
```

Összetett alkalmazások megvalósítása

Példa

*Megvalósítás (**gamemanager.cpp**):*

```
…
void GameManager::newGame(int numRows,
                          int numCols){
   …
   statusChanged(trUtf8("Új játék elindítva, ") +
      players[currentPlayerIndex]->getName() +
      trUtf8(" következik."));
      // esemény kiváltása
   }
}
…
```

Összetett alkalmazások megvalósítása

Többablakos környezet

- A dokumentumkezelő alkalmazásokban sokszor hasznos, ha egyszerre több dokumentumot is megnyithatunk, szerkeszthetünk
- Az alkalmazásokat így két csoportba soroljuk:
  - egy dokumentumot kezelő (*Single Document Interface*, *SDI*)
  - több dokumentumot kezelő (*Multi Document Interface*, *MDI*), ahol egy ablakon belül párhuzamosan több dokumentumot is meg tudunk jeleníteni
    - minden dokumentum egy külön ablak a felületen belül, amelyekből mindig csak egy lehet aktív

Összetett alkalmazások megvalósítása

Többablakos környezet

- A Qt-ben is lehetőségünk van többablakos alkalmazás készítésére, erre szolgál a **`QMdiArea`** vezérlő, amely definiál egy többablakos területet (egy főablak központi vezérlőjeként)
  - az MDI terület biztosít általános (**`SubWindowView`**), valamint lap alapú (**`TabbedView`**) felépítést, ez szabályozható (**`viewMode`**)
  - az ablakok rendezhetőek mozaikosan (**`tileSubWindows()`**) és lépcsőzetesen (**`cascadeSubWindows()`**)
  - a vezérlők felhelyezhetőek az **`addSubWindow(<vezérlő>)`** utasítással, az így felvett ablakok később lekérdezhetőek (**`subWindowList`**), mindig van egy aktív ablak (**`currentSubWindow`**)