Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások fejlesztése I

5. előadás

Alkalmazások architektúrája

Giachetta Roberto

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

Alkalmazások architektúrája

Architekturális tervezés

- Az alkalmazások tervezése során fontos szerep jut az architekturális tervezésnek, azaz a felépítés meghatározásának
  - a feladat elemzése, a *felhasználói követelmények/esetek* megfogalmazása elvezet a biztosítandó *funkció*khoz
  - a funkciókat objektumorientált környezetben *osztályok*kal biztosítjuk
  - az osztályok között *függőség*eket definiálhatunk
    - az azonos/hasonló feladatot ellátó osztályok között szoros kapcsolat, nagy fokú együttműködés, hivatkozás
    - a különböző feladatot ellátó osztályok között laza kapcsolat, kevés együttműködés, hivatkozás

Alkalmazások architektúrája

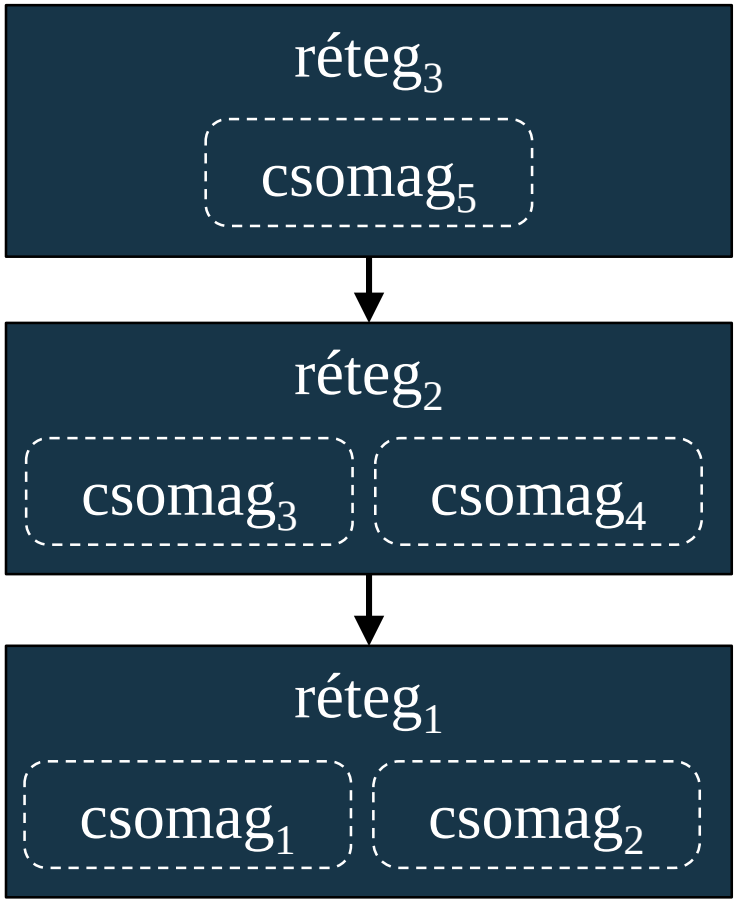
Csomagok

- A felépítést *csomagok*ba (*package*) szervezhetjük, ahol a csomag olyan része az alkalmazásnak, amely
  - egy adott feladatcsoporthoz tartozó funkciókat tartalmazza, de függhet más csomagoktól (felhasználhatja őket)
  - belül erős, kívülre laza kohéziót támaszt
  - alapján felállítható egy *körmentes irányított gráf* (DAG), amely reprezentálja a komponensek összefüggését

Alkalmazások architektúrája

Rétegek

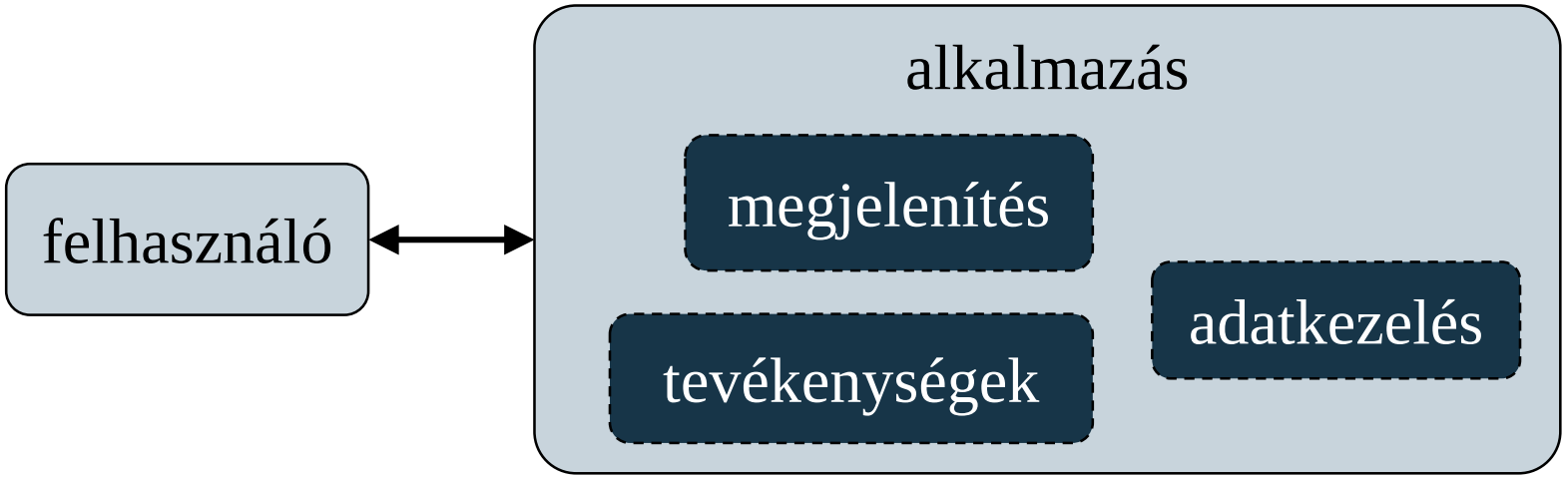
- A csomagokat *rétegek*be (*layer*) szervezhetjük, ahol a réteg az alkalmazás egy tevékenységi szintjének felel meg
  - a rétegek egymásra épülnek, egy réteg csak az alatta levő réteget használhatja, a felette, illetve több szinten alatta lévőkről nincs információja
  - a felhasználó a legfelső réteggel kommunikál
  - a réteg egy, vagy több csomagból áll



Alkalmazások architektúrája

Az egyrétegű architektúra

- A legegyszerűbb felépítést az *egyrétegű architektúra* adja
  - nincsenek különböző rétegekbe szétválasztva a funkciók
  - a felületet megjelenítő kód vegyül az adatkezeléssel, a tevékenységek végrehajtásával, stb.
  - egyszerűbb alkalmazásokra célszerű



Alkalmazások architektúrája

Példa

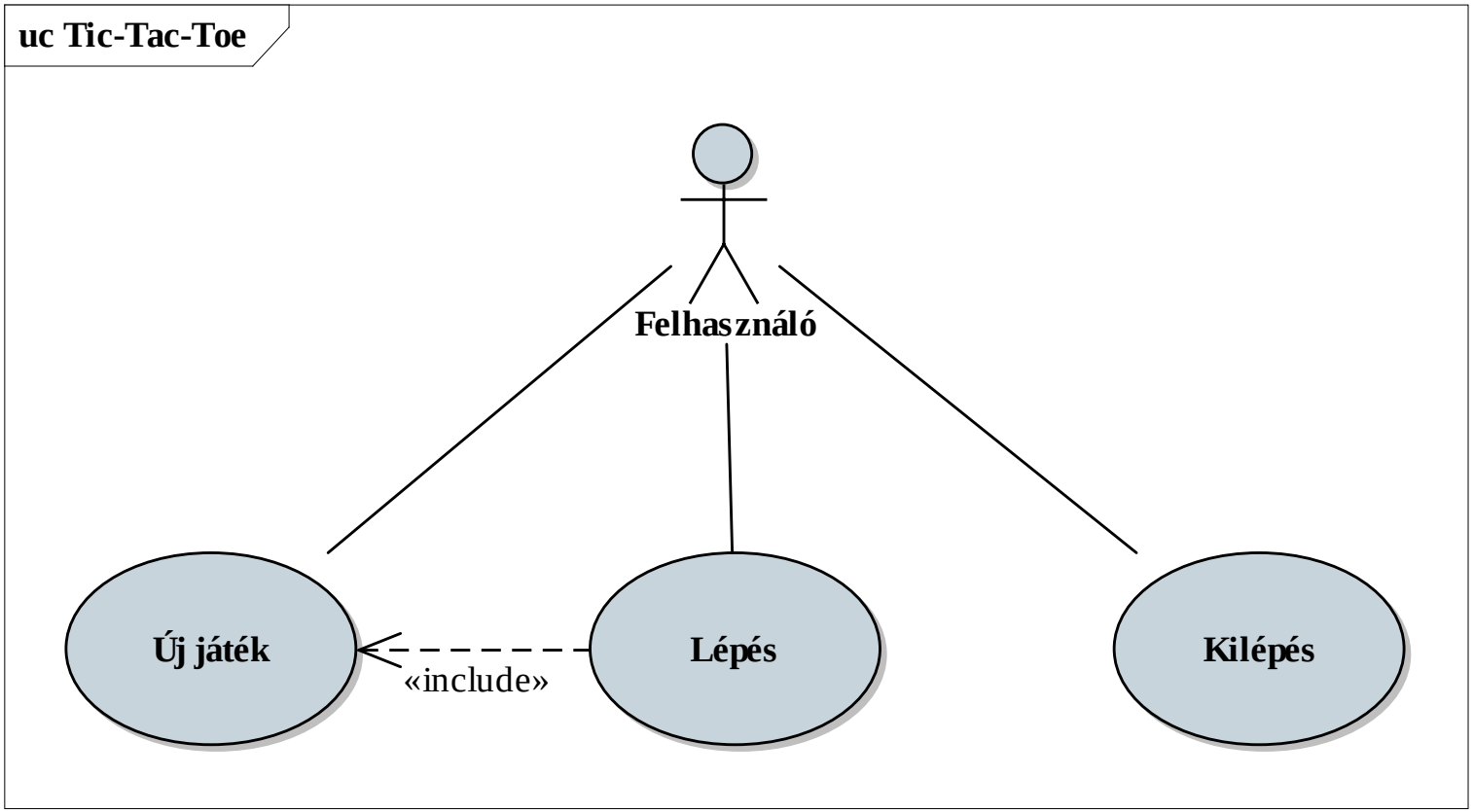
Feladat: Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

- 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)

Alkalmazások architektúrája

Példa

Tervezés (felhasználói esetek):



Alkalmazások architektúrája

Példa

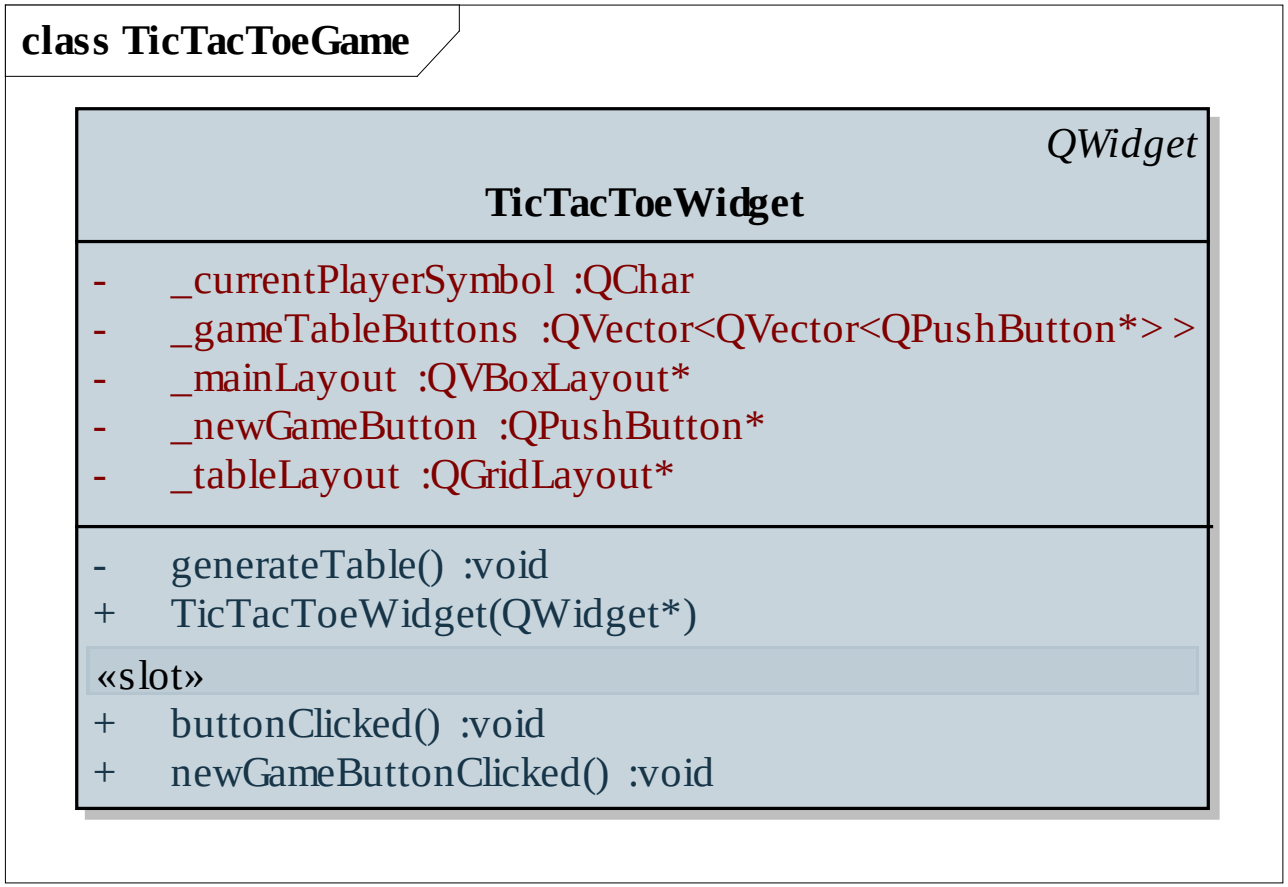
Tervezés (architektúra):

- 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

Alkalmazások architektúrája

Példa

Tervezés (architektúra):



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
```

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';
    …
}
```

Alkalmazások architektúrája

Az egyrétegű architektúra

- Az egyrétegű architektúrát is célszerű tagolni a funkciók mentén a procedurális paradigma elveinek megfelelően növelve az *áttekinthetőség*et és az *újrafelhasználhatóság*ot
  - az adatok szétbonthatóak háttér információkra, illetve a felületen megjelenő tartalomra
    - pl. a játéktábla értékei ábrázolhatóak egész számokkal, függetlenül a megjelenített vezérlőtől
  - a különböző tevékenységek szétválaszthatóak, egyes funkciók kiemelhetőek külön alprogramokba
    - pl.: a gomb lenyomásának eseménykezelője elválasztható a játékbeli lépésből

Alkalmazások architektúrája

Példa

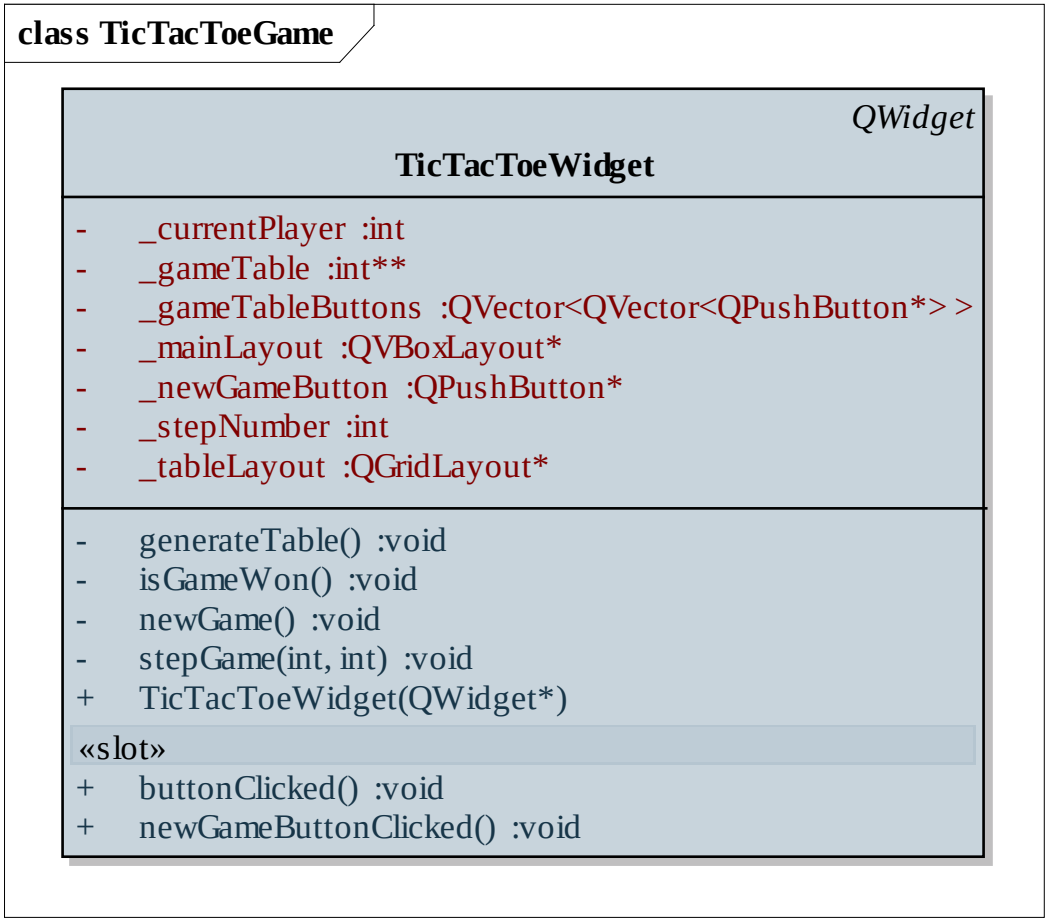
Feladat: Módosítsuk a Tic-Tac-Toe programot úgy, hogy áttekinthetőbb és tagoltabb legyen.

- a játékosok reprezentálhatóak számokkal (1: X, 2: 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`**)

Alkalmazások architektúrája

Példa

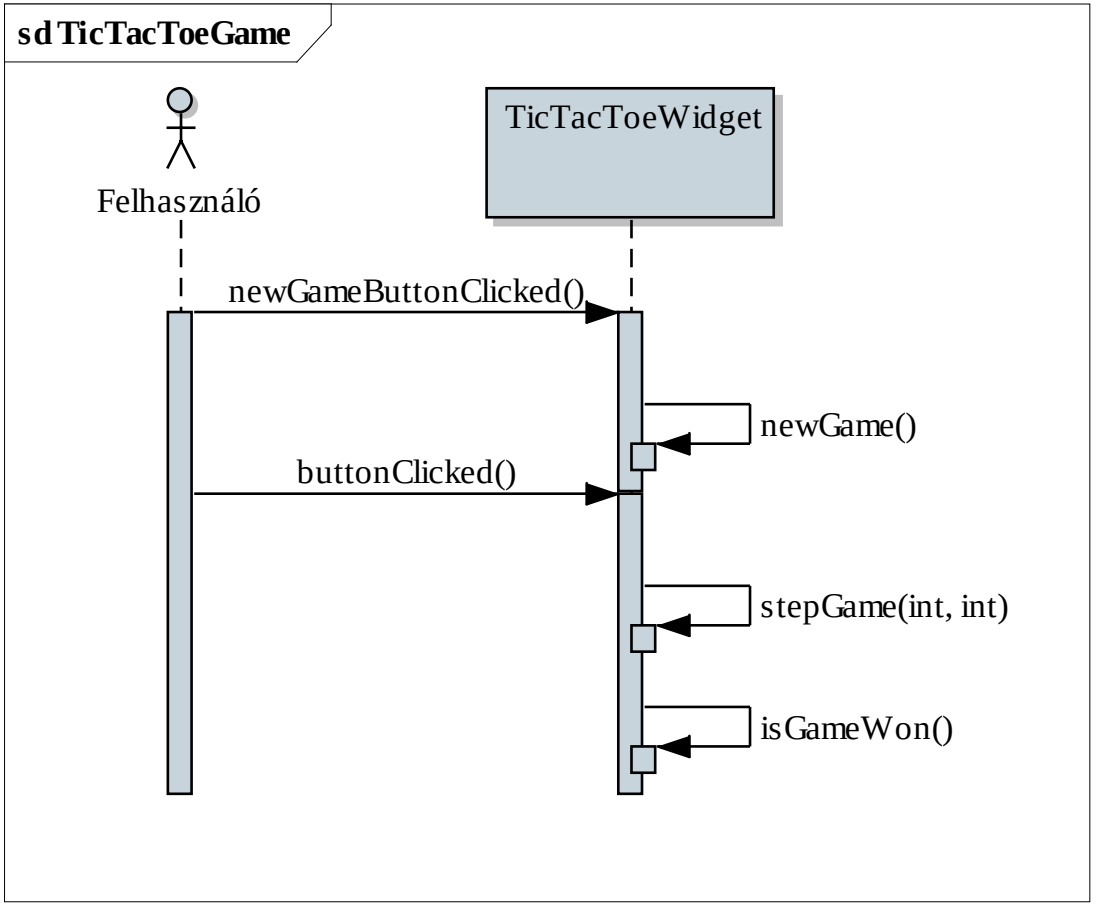
Tervezés (architektúra):



Alkalmazások architektúrája

Példa

Tervezés (viselkedés):



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);
}
```

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("0");
    _gameTableButtons[x][y]->setEnabled(false);

    _stepNumber++;
    _currentPlayer = _currentPlayer % 2 + 1;
    …
```

Alkalmazások architektúrája

A kétrétegű architektúra

- Összetettebb alkalmazásoknál az egyrétegű felépítés korlátozza a program
  - áttekinthetőségét, tesztelését (pl. nehezen látható át, hol tároljuk a számításokhoz szükséges adatokat)
  - módosíthatóságát, bővíthetőségét (pl. nehezen lehet a felület kinézetét módosítani)
  - újrafelhasználhatóságát (pl. komponens kiemelése és áthelyezése másik alkalmazásba)
- A legegyszerűbb felbontás a felhasználói felület leválasztása a háttérbeli tevékenységekről, ezt nevezzük *modell/nézet* (*MV*, *model-view*) architektúrának

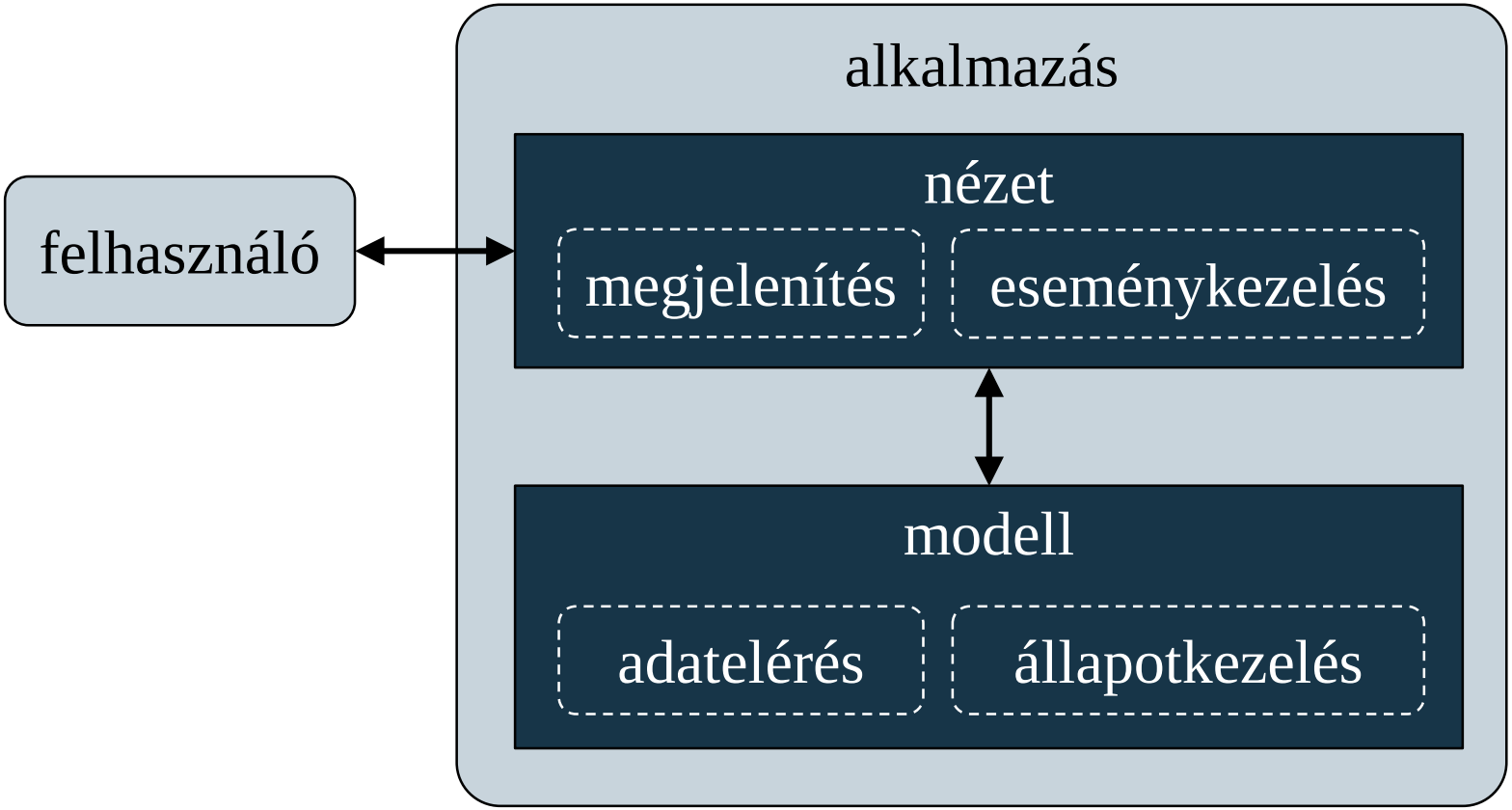
Alkalmazások architektúrája

A kétrétegű architektúra

- A modell/nézet architektúrában
  - a *modell* tartalmazza a háttérben futó logikát, azaz a tevékenységek végrehajtását, az állapotkezelést, valamint az adatkezelést, ezt nevezzük *alkalmazáslogiká*nak, vagy *üzleti logiká*nak
  - a *nézet* tartalmazza a grafikus felhasználói felület megvalósítását, beleértve a vezérlőket és eseménykezelőket
  - a felhasználó a nézettel kommunikál, a modell és a nézet egymással
  - a modell nem függ a nézettől, függetlenül, önmagában is felhasználható, ezért könnyen átvihető másik alkalmazásba, és más felülettel is üzemképes

Alkalmazások architektúrája

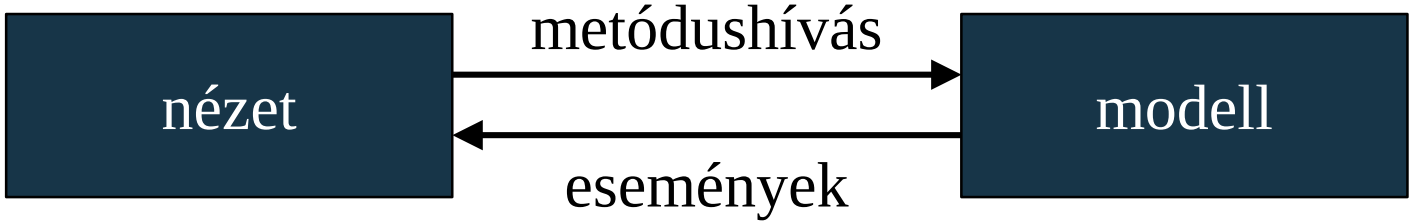
A kétrétegű architektúra



Alkalmazások architektúrája

A kétrétegű 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

Alkalmazások architektúrája

Példa

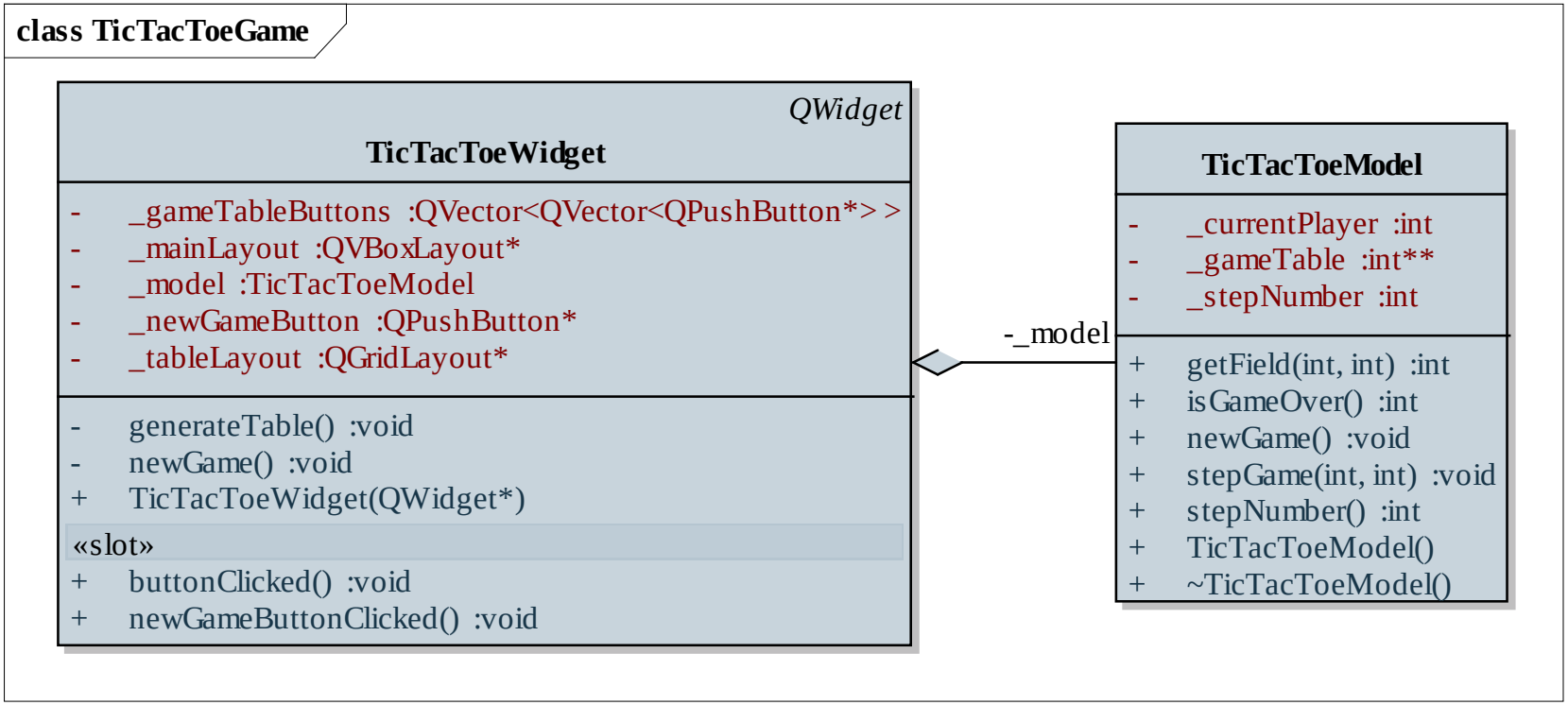
Feladat: Módosítsuk a Tic-Tac-Toe programot úgy, hogy kétrétegű architektúrában valósuljon meg.

- a játékért 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

Alkalmazások architektúrája

Példa

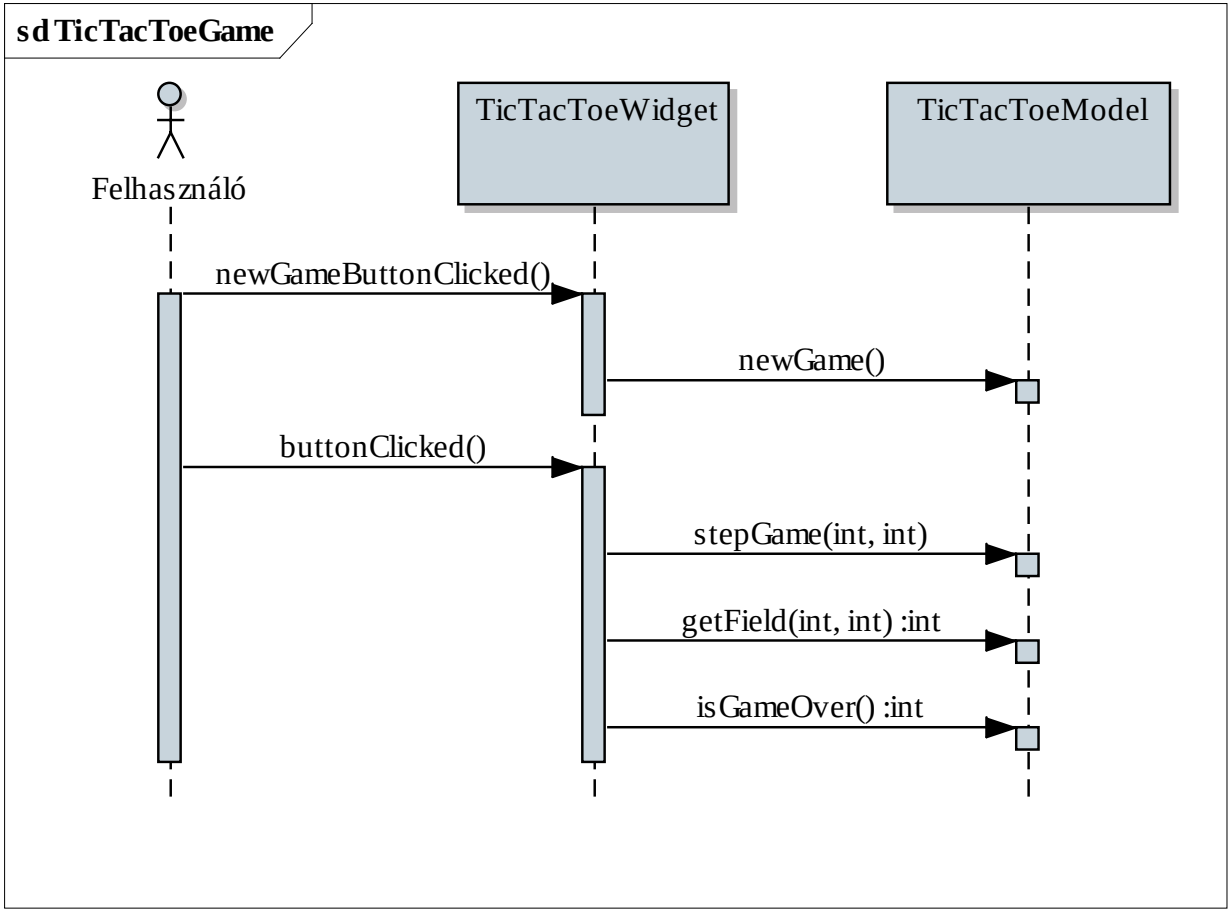
Tervezés (architektúra):



Alkalmazások architektúrája

Példa

Tervezés (viselkedés):



Alkalmazások architektúrája

Példa

Megvalósítás (`tictactoemodel.cpp`*):*

```
void TicTacToeModel::stepGame(int x, int y)
{
   // ellenőrzünk 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;
}
```

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
    …
```

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 **`Q_OBJECT`** 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

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
}
```

Alkalmazások architektúrája

Példa

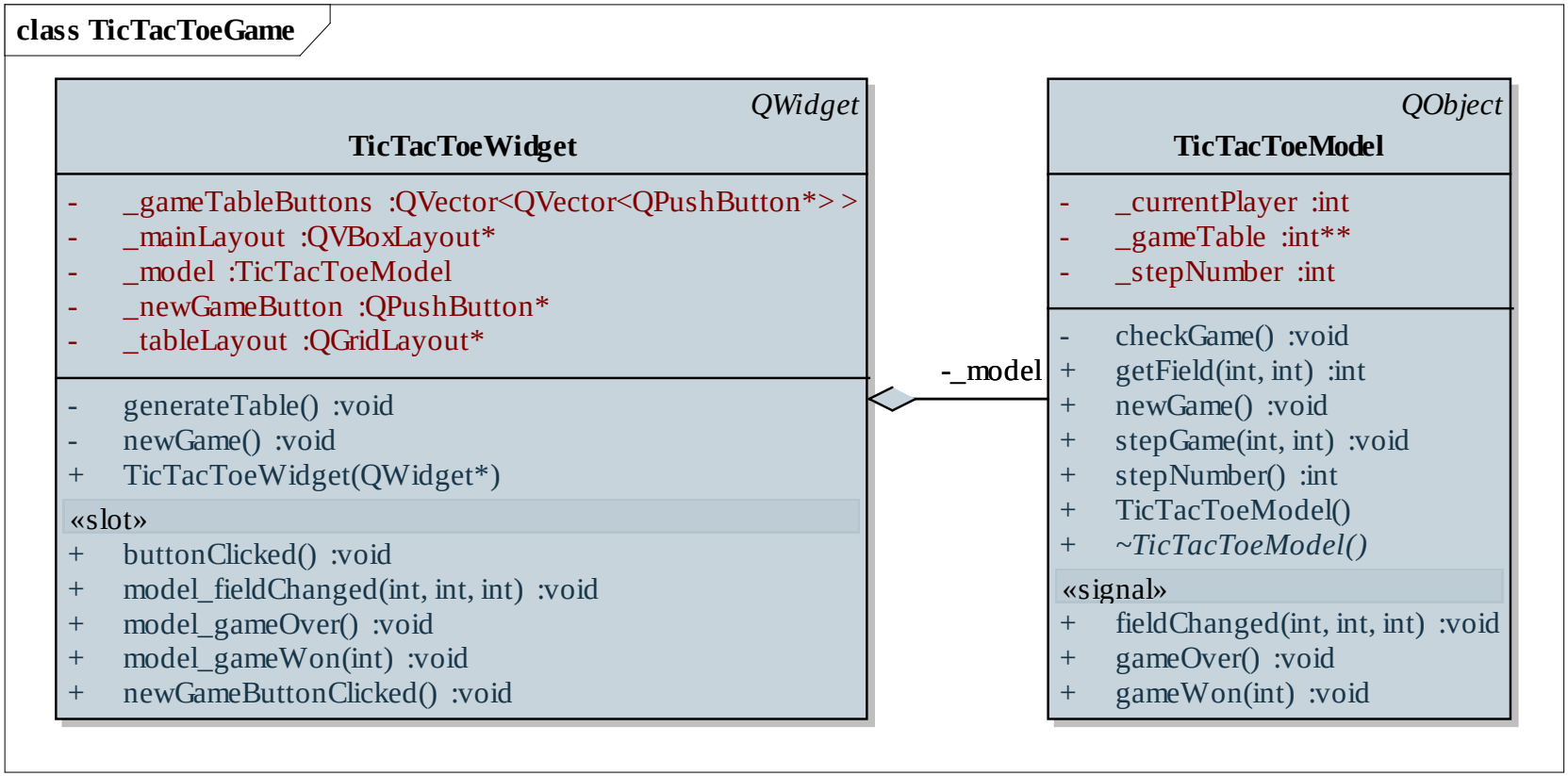
Feladat: Módosítsuk a Tic-Tac-Toe programot úgy, hogy a modell is kommunikáljon a nézettel, események segítségével.

- 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

Alkalmazások architektúrája

Példa

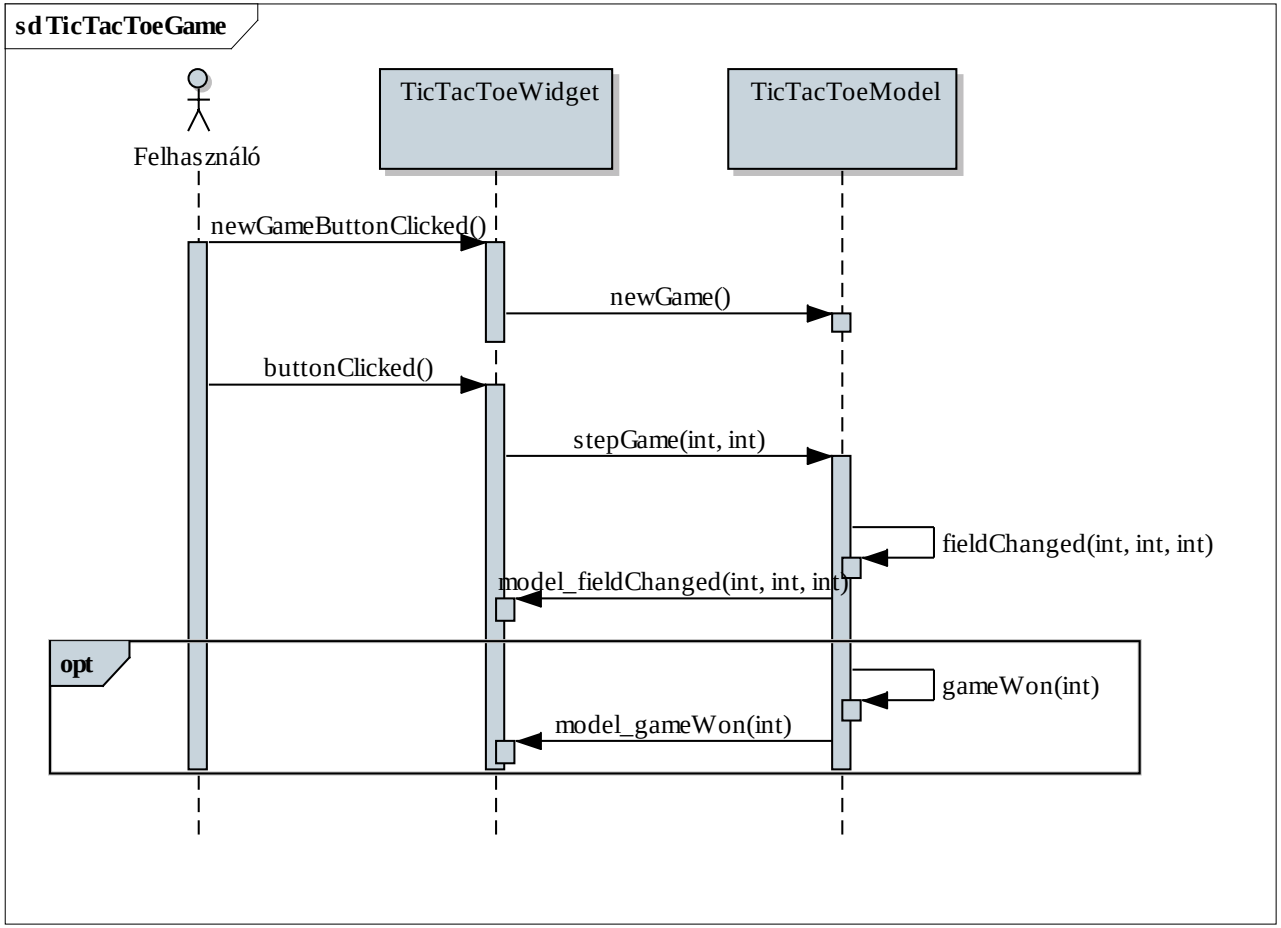
Tervezés (architektúra):



Alkalmazások architektúrája

Példa

Tervezés (viselkedés):



Alkalmazások architektúrája

Példa

Megvalósítás (`tictactoemodel.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
   }
}
```

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();
}
```

Alkalmazások architektúrája

A modell megvalósítása

- A modell tervezése és megvalósítása során tekintettel kell lennünk a következőkre:
  - a modell független a nézettől, nem tudható előre, milyen módon, milyen körülmények között hívják meg műveleteit
    - ezért 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 egyértelműségre és a lehető legkevesebb hibalehetőségre
    - pl. korlátozott értékhalmazra használjunk felsoroló típusokat (**enum**)

Alkalmazások architektúrája

Példa

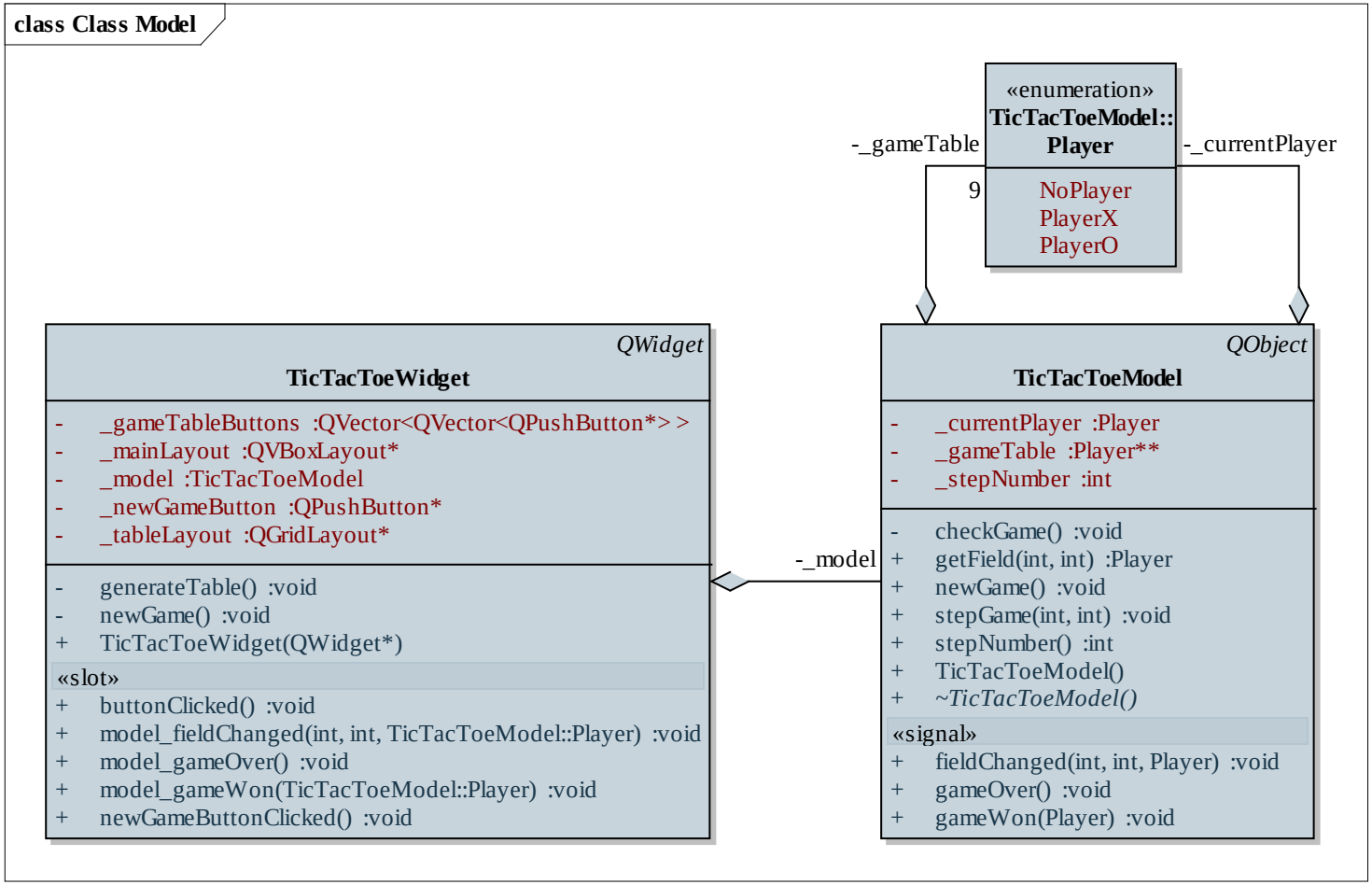
Feladat: Módosítsuk a Tic-Tac-Toe programot úgy, hogy a modellben a játékost és a szimbólumokat enumeráció segítségével valósítjuk meg.

- 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

Alkalmazások architektúrája

Példa

Tervezés (architektúra):



Alkalmazások architektúrája

Példa

Megvalósítás (`tictactoemodel.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
    …
```

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;
   …
   }
}
```

Alkalmazások architektúrája

Példa

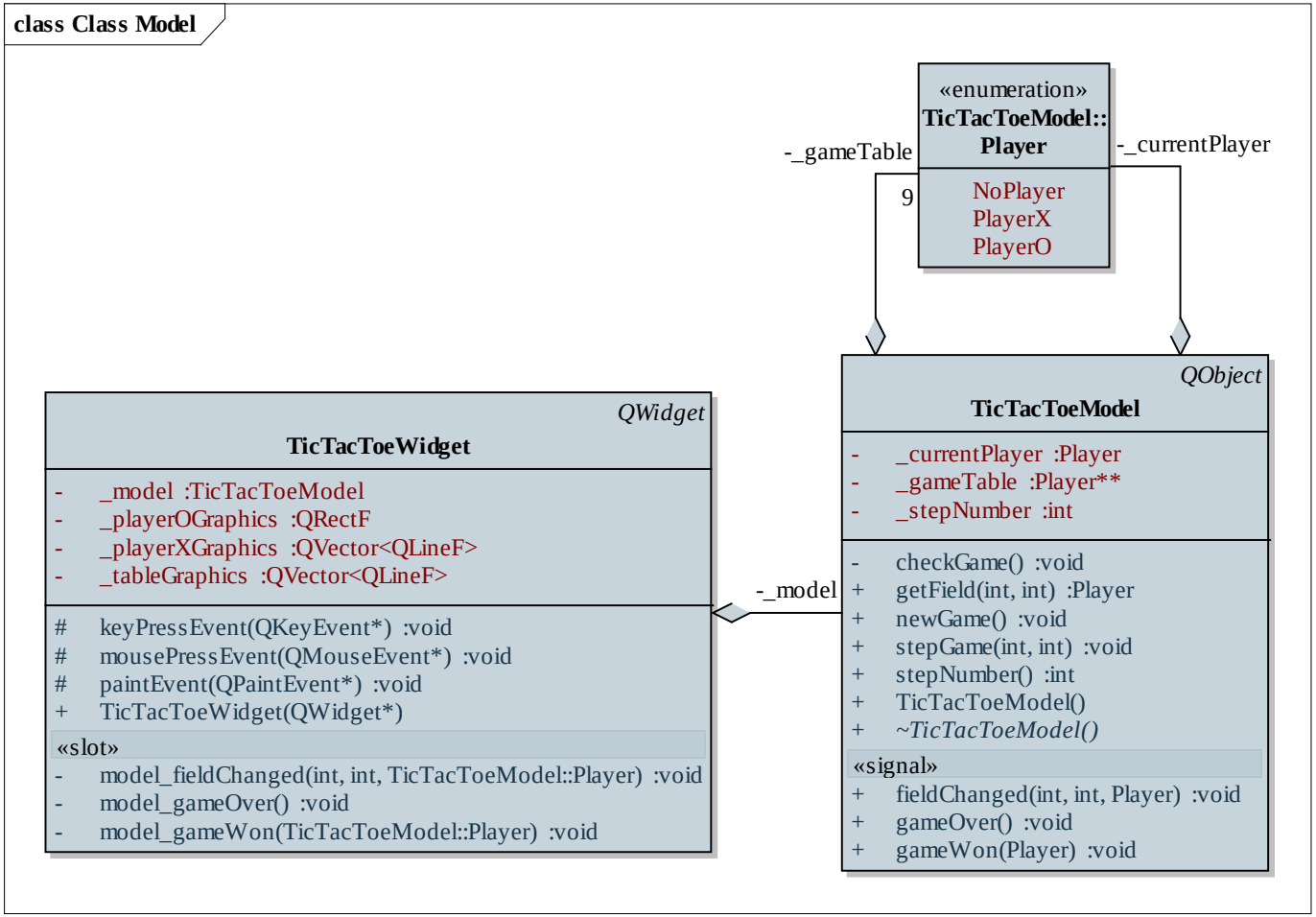
Feladat: Készítsünk új nézetet a Tic-Tac-Toe játékhoz, amelyben nem karakterekkel jelenítjük meg a játékállást, hanem alakzatokat rajzolunk.

- 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

Alkalmazások architektúrája

Példa

Tervezés (architektúra):



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();
    }
}
```

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
}
```