

# Eseményvezérelt alkalmazások architektúrája

Modell-nézet architektúra

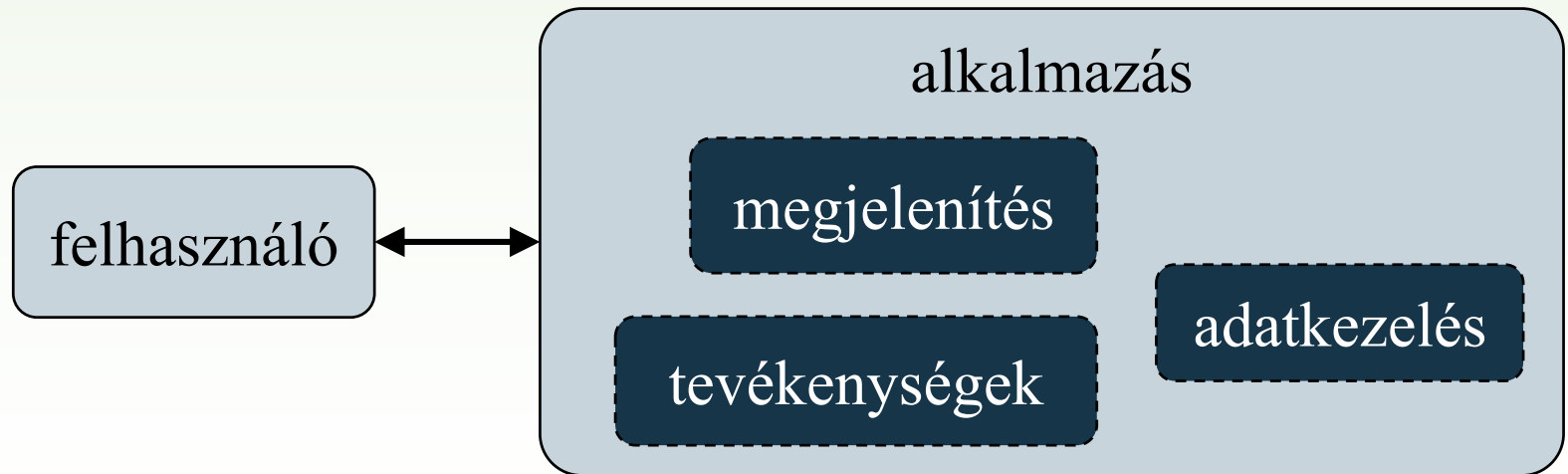
# Szoftver architektúra

---

- ❑ A **szoftver architektúra** elsődleges feladata *a rendszer magas szintű felépítésének és működésének meghatározása*:
  - megnevezi a szoftver fő komponenseit
  - megmutatja azok kapcsolatait a szolgáltatott és elvárt interfészek, a kommunikációs csatornák és csatlakozási pontok jellemzésével
- ❑ A szoftver architektúra megválasztása a szoftver fejlesztése során meghozott *elsődleges tervezési döntések* eredménye, amely
  - kihat a rendszer felépítésére, viselkedésére, kommunikációjára, nem funkcionális jellemzőire és megvalósítására,
  - amely későbbi megváltoztatása a szoftver jelentős újratervezését vonná maga után.

# Monolitikus architektúra

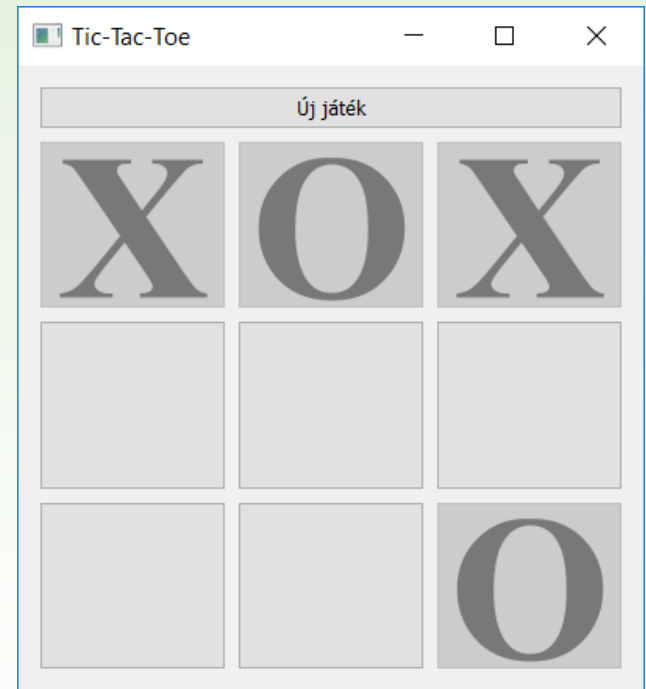
- A legegyszerűbb felépítéssel a **monolitikus architektúra** (*monolithic architecture*) rendelkezik, amely nem különíti el egymástól az egyes feladatköröket (pl. megjelenítés, adatkezelés).



# 1.Feladat

Készítsünk Tic-Tac-Toe játékot, amelyben két játékos küzdhet egymás ellen.

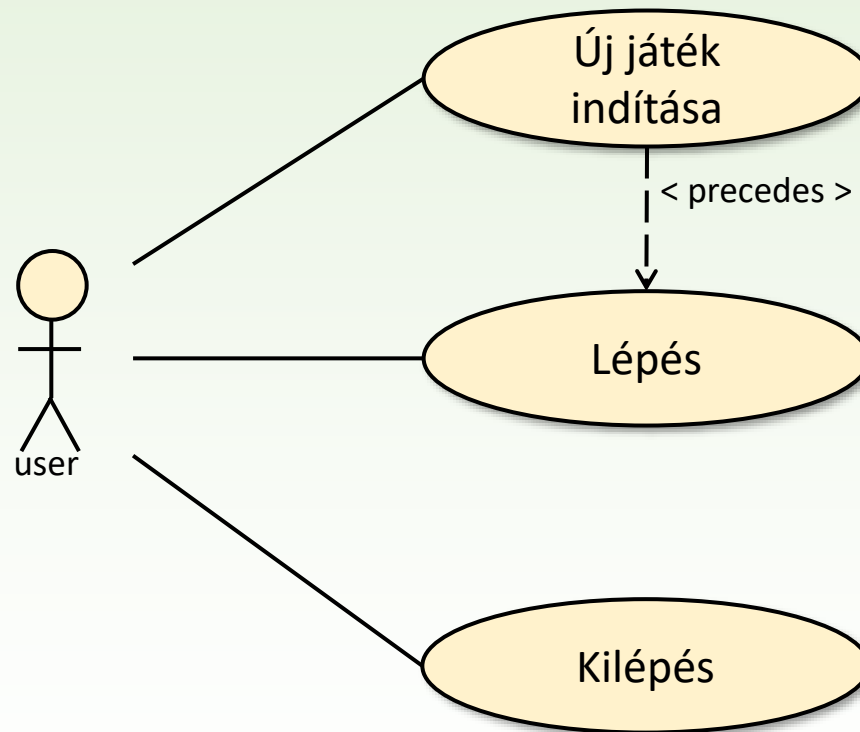
- A két játékos (akiket az ,X' és ,O' jelekkel ábrázolunk) felváltva tett lépéseire.
- A program előugró üzenetben jelez, ha vége a játéknak, majd új játékot kezd.
- A felhasználók bármikor indíthatnak új játékot.



- Az alkalmazás felületét nyomógombok segítségével valósítjuk meg (Kilenc játékgomb, valamint az új játék kezdésére szolgáló).

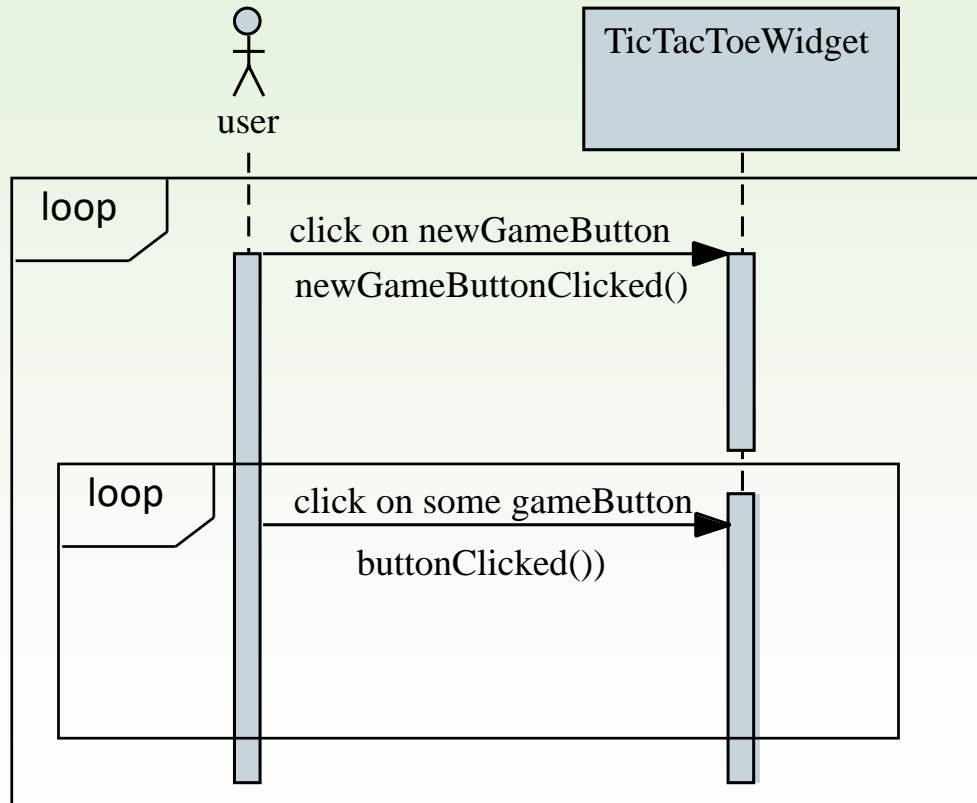
# 1.Feladat: elemzés

---



# 1.Feladat: elemzés

---

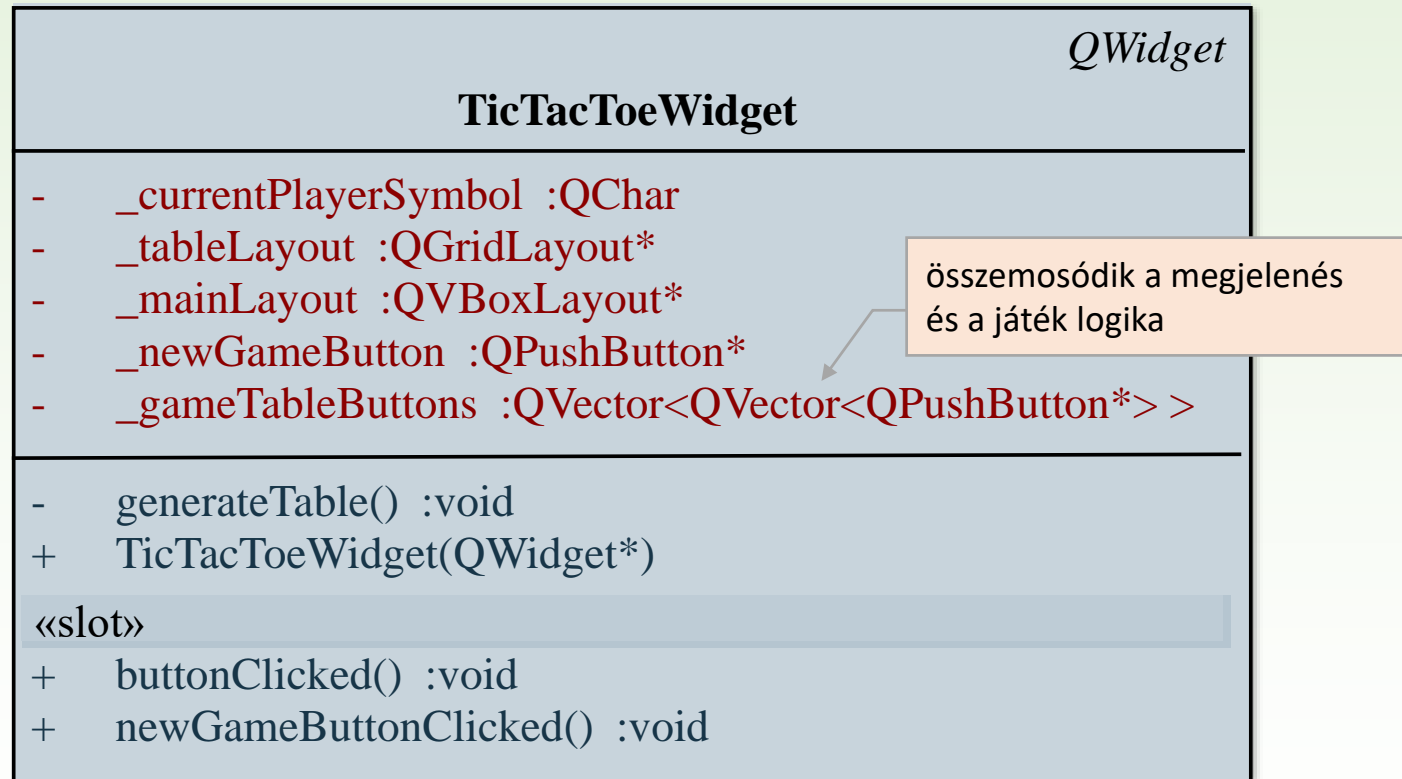


# 1.Feladat: tervezés

---

- ❑ Az alkalmazást **egyetlen osztályban** (**TicTacToeWidget**) valósítjuk meg, amely tartalmazza a grafikus felületet és a játék viselkedését.
- ❑ A felületet a konstruktor és a **generateTable** segédmetódus állítja elő, elrendezők segítségével.
- ❑ A felületen elhelyezzük az „új játék” gombját (**\_newGameButton**), valamint a játéktábla gombjait (**\_gameTableButtons**), továbbá egy karakterrel (**\_currentPlayerSymbol**) eltároljuk az aktuális játékos jelét.
- ❑ A játékot az eseménykezelők vezérlik.

# 1.Feladat: tervezés





# 1.Feladat: TicTacToeWidget()

```
TicTacToeWidget::TicTacToeWidget(QWidget *parent) : QWidget(parent)
{
    setMinimumSize(400, 400);
    setBaseSize(400,400);
    setWindowTitle(tr("Tic-Tac-Toe"));
    _newGameButton = new QPushButton(tr("Új játék"));
    connect( _newGameButton, SIGNAL(clicked()),
            this,                SLOT(newGameButtonClicked()));
    _mainLayout = new QVBoxLayout(); // vertikális elhelyezkedés
    _mainLayout->addWidget(_newGameButton);
    _tableLayout = new QGridLayout(); // rácsos elhelyezkedés mezőknek
    _mainLayout->addLayout(_tableLayout);
    generateTable();
    setLayout(_mainLayout);
    _currentPlayerSymbol = 'X'; // kezdő játékos
}
```

a felület statikus elemei

elrendezők

a felület statikus elemei

# 1.Feladat: generateTable()

```
void TicTacToeWidget::generateTable()
{
    _gameTableButtons.resize(3);
    for (int i = 0; i < 3; ++i){
        _gameTableButtons[i].resize(3);
        for (int j = 0; j < 3; ++j){
            _gameTableButtons[i][j]= new QPushButton(this);
            _gameTableButtons[i][j]->setFont(
                QFont("Times New Roman", 80, QFont::Bold));
            _gameTableButtons[i][j]->setSizePolicy(
                QSizePolicy::Ignored, QSizePolicy::Ignored);
            _tableLayout->addWidget(_gameTableButtons[i][j], i, j);
            // gombok felvétele az elhelyezésbe
            connect( _gameTableButtons[i][j], SIGNAL(clicked()),
                    this,                               SLOT(buttonClicked()));
        }
    }
}
```

vezérlők dinamikus létrehozása

# 1.Feladat: newGameButtonClicked()

---

```
void TicTacToeWidget::newGameButtonClicked()
{
    for (int i = 0; i < 3; ++i)
        for (int j = 0; j < 3; ++j){
            _gameTableButtons[i][j]->setText(""); // szöveg törlése
            _gameTableButtons[i][j]->setEnabled(true); // gombot aktív
        }

    _currentPlayerSymbol = 'X'; // kezdő játékos
}
```

dinamikus vezérlők  
tulajdonságainak változtatása

# 1.Feladat: buttonClicked() 1.

```
void TicTacToeWidget::buttonClicked()
{
```

```
    QPushButton* senderButton = qobject_cast<QPushButton*> (sender());
```

lekérjük az esemény küldőjét

```
    int location = _tableLayout->indexOf(senderButton);
```

```
    int x = location / 3;
```

```
    int y = location % 3;
```

a gomb rácson belüli pozíciója  
megadja a koordinátákat

```
    _gameTableButtons[x][y]->setText(_currentPlayerSymbol);
```

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

megjelenítés a gombon

```
    if (_currentPlayerSymbol == 'X') _currentPlayerSymbol = 'O';
```

```
    else _currentPlayerSymbol = 'X';
```

váltjuk a játékost

```
    ...
```

# 1.Feladat: buttonClicked() 2.

```
void TicTacToeWidget::buttonClicked()
{
    ...
    QString won = "";
    for (int i = 0; i < 3; ++i){
        if (_gameTableButtons[i][0]->text() != ""
            && _gameTableButtons[i][0]->text() == _gameTableButtons[i][1]->text()
            && _gameTableButtons[i][1]->text() == _gameTableButtons[i][2]->text())
            won = _gameTableButtons[i][0]->text();
    }
    for (int i = 0; i < 3; ++i){
        if (_gameTableButtons[0][i]->text() != ""
            && _gameTableButtons[0][i]->text() == _gameTableButtons[1][i]->text()
            && _gameTableButtons[1][i]->text() == _gameTableButtons[2][i]->text())
            won = _gameTableButtons[0][i]->text();
    }
    ...
}
```

Van-e azonos jelekből álló sor?

Van-e azonos jelekből álló oszlop?

# 1.Feladat: buttonClicked() 3.

```
void TicTacToeWidget::buttonClicked()
{
    ...
    if ( _gameTableButtons[0][0]->text() != ""
        && _gameTableButtons[0][0]->text() == _gameTableButtons[1][1]->text()
        && _gameTableButtons[1][1]->text() == _gameTableButtons[2][2]->text() )
        won = _gameTableButtons[0][0]->text() ;

    if ( _gameTableButtons[0][2]->text() != ""
        && _gameTableButtons[0][2]->text() == _gameTableButtons[1][1]->text()
        && _gameTableButtons[1][1]->text() == _gameTableButtons[2][0]->text() )
        won = _gameTableButtons[0][2]->text() ;

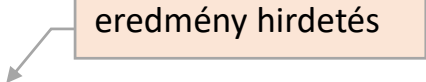
    ...
}
```

Azonos jelekből áll-e a főátló?

Azonos jelekből áll-e a mellékátló?

# 1.Feladat: buttonClicked() 4.

```
...
if (won == "X") {
    QMessageBox::information(this, tr("Játék vége!"), tr("Az X nyerte a
    játékot!")); newGameButtonClicked();
} else if (won == "O"){
    QMessageBox::information(this, trUtf8("Játék vége!"),
    tr("A O nyerte a játékot!")); newGameButtonClicked();
} else {
    int numberOfChars = 0;
    for (int i = 0; i < 3; ++i){
        for (int j = 0; j < 3; ++j)
            if (_gameTableButtons[i][j]->text() != "") numberOfChars++;
    }
    if (numberOfChars == 9){
        QMessageBox::information(this, tr("Játék vége!"),
        tr("A játék döntetlen lett!")); newGameButtonClicked();
    }
}
}
```



# Monolitikus architektúra korlátjai

- ❑ Összetettebb alkalmazásoknál a monolitikus felépítés korlátozza a program
  - **áttekinthetőségét** (nehezen találhatóak a számításhoz szükséges adatok).
  - **tesztelhetőségét** (nem ellenőrizhetők külön-külön az egyes funkciók).
  - **módosíthatóságát, bővíthetőségét** (a felület kinézetét csak a működés átírásával együtt tudjuk módosítani).
  - **újrafelhasználhatóságát** (a funkciók nem emelhetők ki és vihetők át másik alkalmazásba).
- ❑ Célszerű a program felépítését felbontani
  - **funkciók mentén**: a tevékenységeket külön alprogramokba tesszük.
    - pl.: a játékbeli lépést helyezhetjük külön alprogramba, így függetlenedik az eseménykezelőtől
  - **adatok mentén**: ne 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



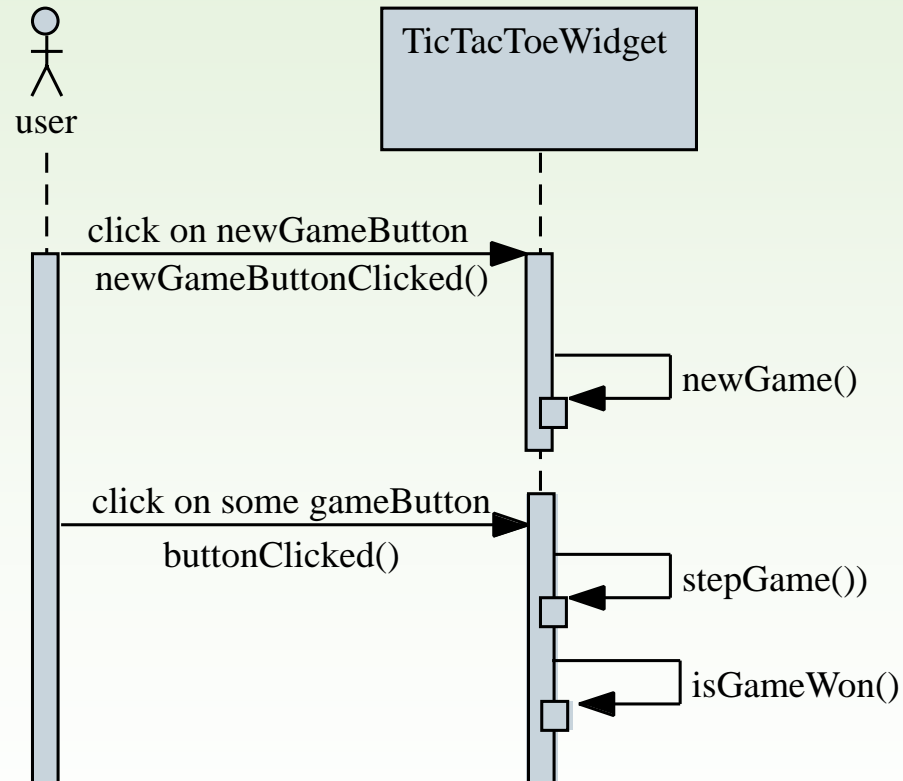
## 2.Feladat

---

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

- új metódusokat veszünk fel a játék kezelésére (**newGame**, **stepGame**, **isGameWon**)
- a játéktáblát egy külön mátrixban (**\_gameTable**) tároljuk, ahol a játékosok jeleit számok (1: X, 2: O, 0: még nincs érték) helyettesítik
- az aktuális játékost is számként ábrázoljuk (**\_currentPlayer**)
- elmentjük a lépések számát (**\_stepNumber**), így nem kell állandóan ellenőrizni, hogy van-e még szabad mező

## 2.Feladat: elemzés



## 2.Feladat: tervezés

TicTacToeWidget		<i>QWidget</i>
<ul style="list-style-type: none"><li>- _tableLayout :QGridLayout*</li><li>- _mainLayout :QVBoxLayout*</li><li>- _newGameButton :QPushButton*</li><li>- _gameTableButtons :QVector&lt;QVector&lt;QPushButton*&gt; &gt;</li><li>- _stepNumber :int</li><li>- _currentPlayer :int</li><li>- _gameTable :int**</li></ul>		
<ul style="list-style-type: none"><li>- generateTable() :void</li><li>- isGameWon() :void</li><li>- newGame() :void</li><li>- stepGame(int, int) :void</li><li>+ TicTacToeWidget(QWidget*)</li></ul>		
«slot»		
<ul style="list-style-type: none"><li>+ buttonClicked() :void</li><li>+ newGameButtonClicked() :void</li></ul>		

## 2.Feladat: megvalósítás

```
void TicTacToeWidget::newGame()
{
    for (int i = 0; i < 3; ++i)
        for (int j = 0; j < 3; ++j){
            _gameTable[i][j] = 0; // a játékosok pozícióit töröljük
            _gameTableButtons[i][j]->setText(""); // törlés
            _gameTableButtons[i][j]->setEnabled(true);
        }
    _stepNumber = 0;
    _currentPlayer = 1; // először az X lép
}

void TicTacToeWidget::newGameButtonClicked()
{
    newGame();
}
```

a felülettől elválasztott adattárolás

## 2.Feladat: megvalósítás

```
void TicTacToeWidget::buttonClicked()
{
    QPushButton* senderButton = qobject_cast<QPushButton*>(sender());
    int location = _tableLayout->indexOf(senderButton);
    stepGame(location / 3, location % 3);
}

void TicTacToeWidget::stepGame(int x, int y){
    _gameTable[x][y] = _currentPlayer;
    if (_currentPlayer == 1) _gameTableButtons[x][y]->setText("X");
    else _gameTableButtons[x][y]->setText("O");
    _gameTableButtons[x][y]->setEnabled(false);

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

    isGameWon();
}
```

játékosváltás kényelmesebb

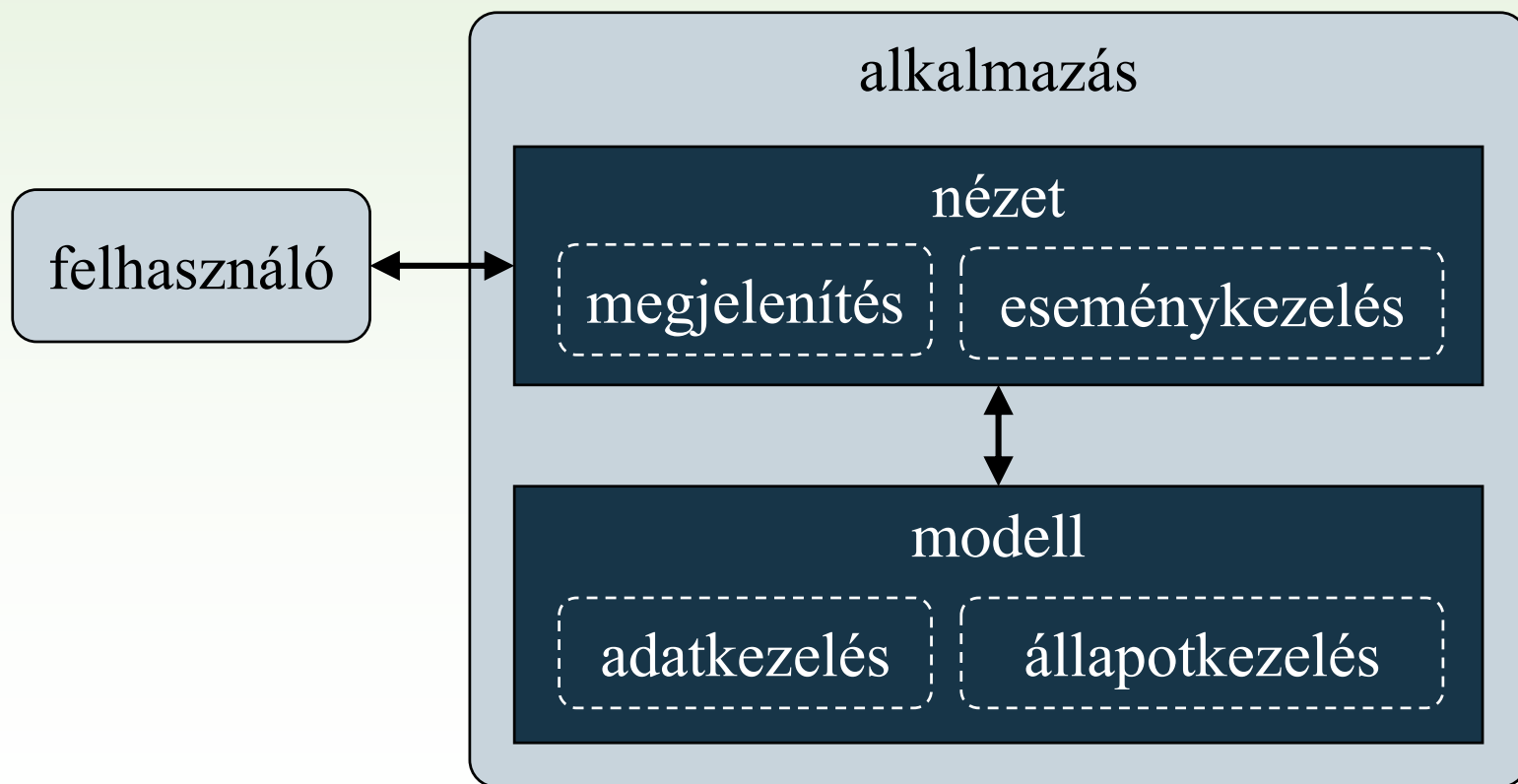
külön van a játékállás kiértékelése

# Modell/nézet architektúra

- ❑ A programszerkezet akkor ideális, ha külön programegységekbe tudjuk szétválasztani a felhasználói felülettel kapcsolatos részeket a feladat megoldását szolgáltató funkcionalitástól.
- ❑ Ezt a felbontást követve jutunk el a **modell/nézet** (*MV, model-view*) architektúrához, amelyben
  - a **modell** tartalmazza a feladat megoldásáért felelős 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.
- ❑ A modell és a nézet két önálló komponens:
  - mindkettő szorosan együttműködő **objektumok összetételei**
  - jól definiált interfészen keresztül kommunikálnak egymással: minden komponensről tudjuk, hogy **mit igényel** és **mit szolgáltató**.

# M/V architektúra kommunikációja

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



# 3.Feladat

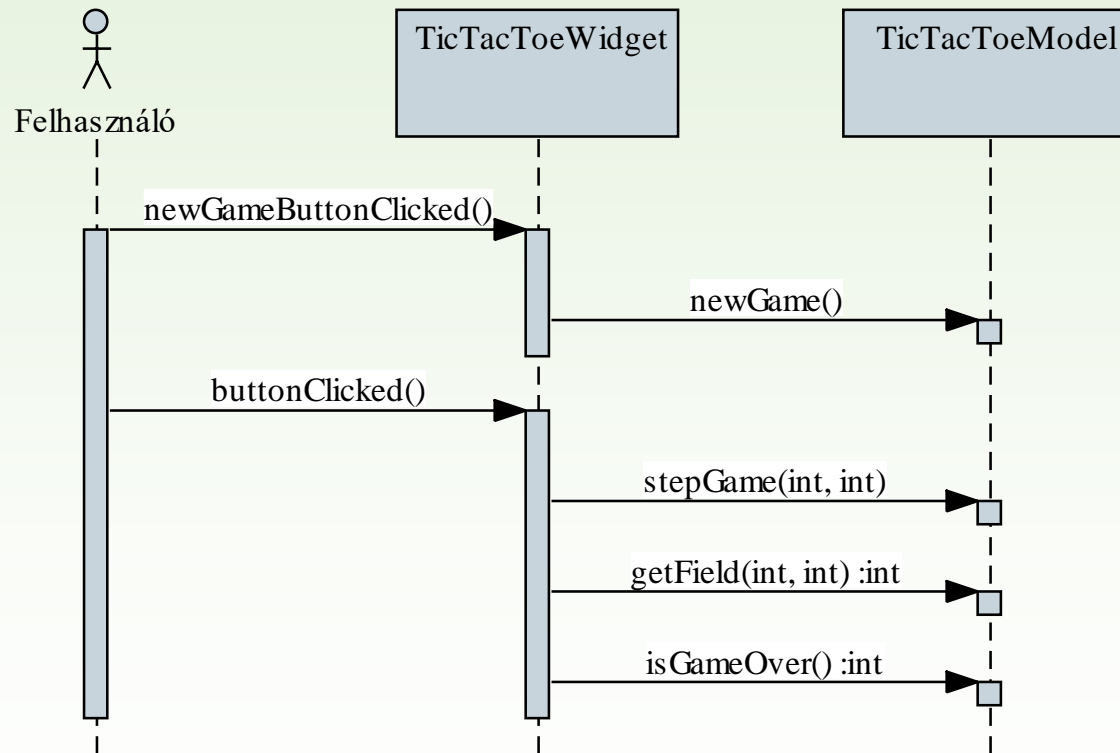
---

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

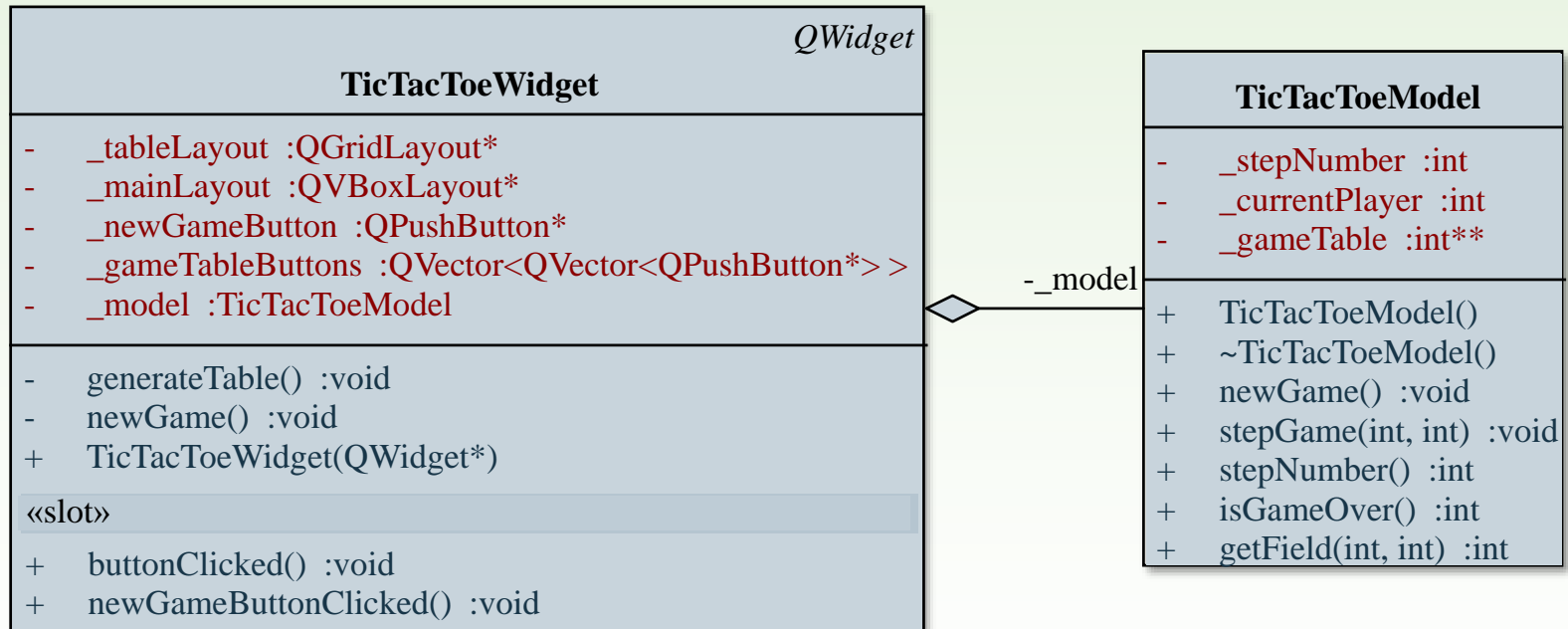
- ❑ A játékért felelős programrészeket a **modellt** megvalósító osztályba (**TicTacToeModel**) tesszük.
  - csak egész számokkal dolgozunk, függetlenül a felülettől;
  - a **játekműveletek** **publikusak** lesznek, így a felületért felelős kód könnyen hívhatja.
  - a modellt megfelelő **ellenőrzésekkel** kell ellátni (mivel leválasztottuk a tevékenységeit).
- ❑ A **nézet** (**TicTacToeWidget**) aggregálja a modellt, és biztosítja a grafikus megjelenítést, a játék műveleteinek hívását, az eredmények megjelenítését.



# 3.Feladat: elemzés



# 3.Feladat: tervezés



# 3.Feladat: modell


```
TicTacToeModel::TicTacToeModel()
{
    _gameTable = new int*[3];
    for (int i = 0; i < 3; ++i) {
        _gameTable[i] = new int[3];
    }
}

TicTacToeModel::~~TicTacToeModel()
{
    delete[] _gameTable;
}

int TicTacToeModel::getField(int x, int y)
{
    if (x < 0 || x > 2 || y < 0 || y > 2) return 0;
    return _gameTable[x][y];
}

int TicTacToeModel::stepNumber() { return _stepNumber; }
```

getter-ek a modell  
állapotának lekérdezéséhez



# 3.Feladat: modell

ezek a metódusok teljesen elváltak a felülettől

```
void TicTacToeModel::newGame() {  
    for (int i = 0; i < 3; ++i)  
        for (int j = 0; j < 3; ++j) _gameTable[i][j] = 0;  
        // a játékosok pozícióit töröljük  
    _stepNumber = 0;  
    _currentPlayer = 1; // először az X lép  
}  
  
void TicTacToeModel::stepGame(int x, int y){  
    if (_stepNumber >= 9) return;  
    if (x < 0 || x > 2 || y < 0 || y > 2) return;  
    if (_gameTable[x][y] != 0) return;  
    _gameTable[x][y] = _currentPlayer;  
  
    _stepNumber++;  
    _currentPlayer = _currentPlayer % 2 + 1;  
}
```

# 3.Feladat: modell

ez is a modell metódusa

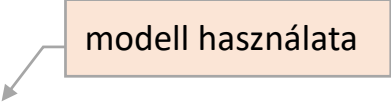
```
int TicTacToeModel::isGameOver()
{
    int won = 0;
    for(int i = 0; i < 3; ++i) {
        if (_gameTable[i][0]!=0 && _gameTable[i][0]==_gameTable[i][1]
            && _gameTable[i][1]==_gameTable[i][2]) won = _gameTable[i][0];
    }
    for(int j = 0; j < 3; ++j) {
        if (_gameTable[0][j]!=0 && _gameTable[0][j]==_gameTable[1][j]
            && _gameTable[1][j]==_gameTable[2][j]) won = _gameTable[0][j];
    }
    if (_gameTable[0][0]!=0 && _gameTable[0][0]==_gameTable[1][1]
        && _gameTable[1][1]==_gameTable[2][2]) won = _gameTable[0][0];
    if (_gameTable[0][2]!=0 && _gameTable[0][2]==_gameTable[1][1]
        && _gameTable[1][1]==_gameTable[2][0]) won = _gameTable[0][2];
    if (won==0 && _stepNumber==9) return 3; // ha döntetlen
    else return won;
}
```

# 3.Feladat: nézet

```
TicTacToeWidget::TicTacToeWidget(QWidget *parent) : QWidget(parent)
{
    setMinimumSize(400, 400);
    setBaseSize(400,400);
    setWindowTitle(tr("Tic-Tac-Toe"));

    _newGameButton = new QPushButton(tr("Új játék"));
    connect( _newGameButton, SIGNAL(clicked()),
            this, SLOT(newGameButtonClicked()));
    _mainLayout = new QVBoxLayout();
    _mainLayout->addWidget(_newGameButton);
    _tableLayout = new QGridLayout();
    _mainLayout->addLayout(_tableLayout);
    generateTable();
    setLayout(_mainLayout);

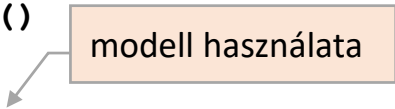
    _model.newGame(); // új játék indítása
}
```



# 3.Feladat: nézet

```
void TicTacToeWidget::newGameButtonClicked()
{
    newGame();
}

void TicTacToeWidget::newGame()
{
    _model.newGame();
    for (int i = 0; i < 3; ++i){
        for (int j = 0; j < 3; ++j){
            _gameTableButtons[i][j]->setText("");
            _gameTableButtons[i][j]->setEnabled(true);
        }
    }
}
```



# 3.Feladat: nézet

```
void TicTacToeWidget::buttonClicked()
{
    QPushButton* senderButton =
        dynamic_cast <QPushButton*>(QObject::sender());
    int location = _tableLayout->indexOf(senderButton);
    int x = location / 3;
    int y = location % 3;
    _model.stepGame(x, y); // játék léptetése
    if (_model.getField(x, y) == 1)
        _gameTableButtons[x][y]->setText("X");
    else _gameTableButtons[x][y]->setText("O");
    _gameTableButtons[x][y]->setEnabled(false);
    int won = _model.isGameOver(); // játék végének ellenőrzése
    ... // eredmény kiírása : QMessageBox::information(...)
    newGameButtonClicked();
}
```

a nézet vezérli az alkalmazást a modell megfelelő metódusainak hívásával.

modell használata

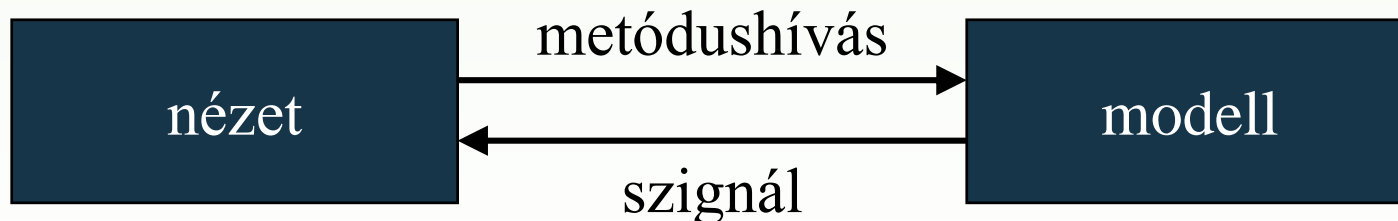
kettős könyvelés

modell használata



# M/V architektúra megvalósítása

- ❑ A modell és a nézet kapcsolatát úgy kell megvalósítani, hogy ne a nézet, hanem a modell vezérelje az alkalmazást, anélkül, hogy ismernie kelljen nézetet:
  - a modell események kiváltásával kommunikál a nézettel, de nem kötődik a nézet példányához
  - a nézet hozzáfér a modell publikus elemeihez (ismeri annak interfészét, hívhatja publikus metódusait), mert hivatkozhat a nézet példányára



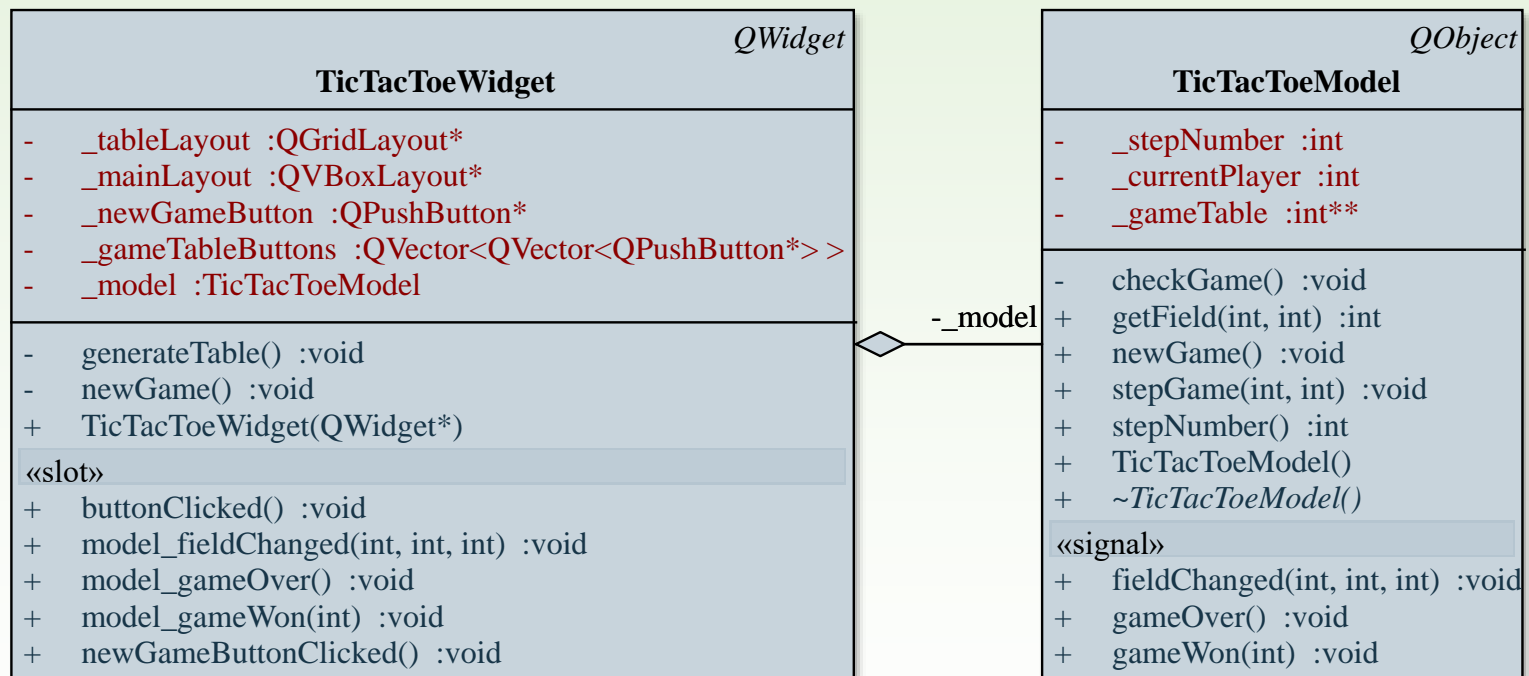
# 4.Feladat

---

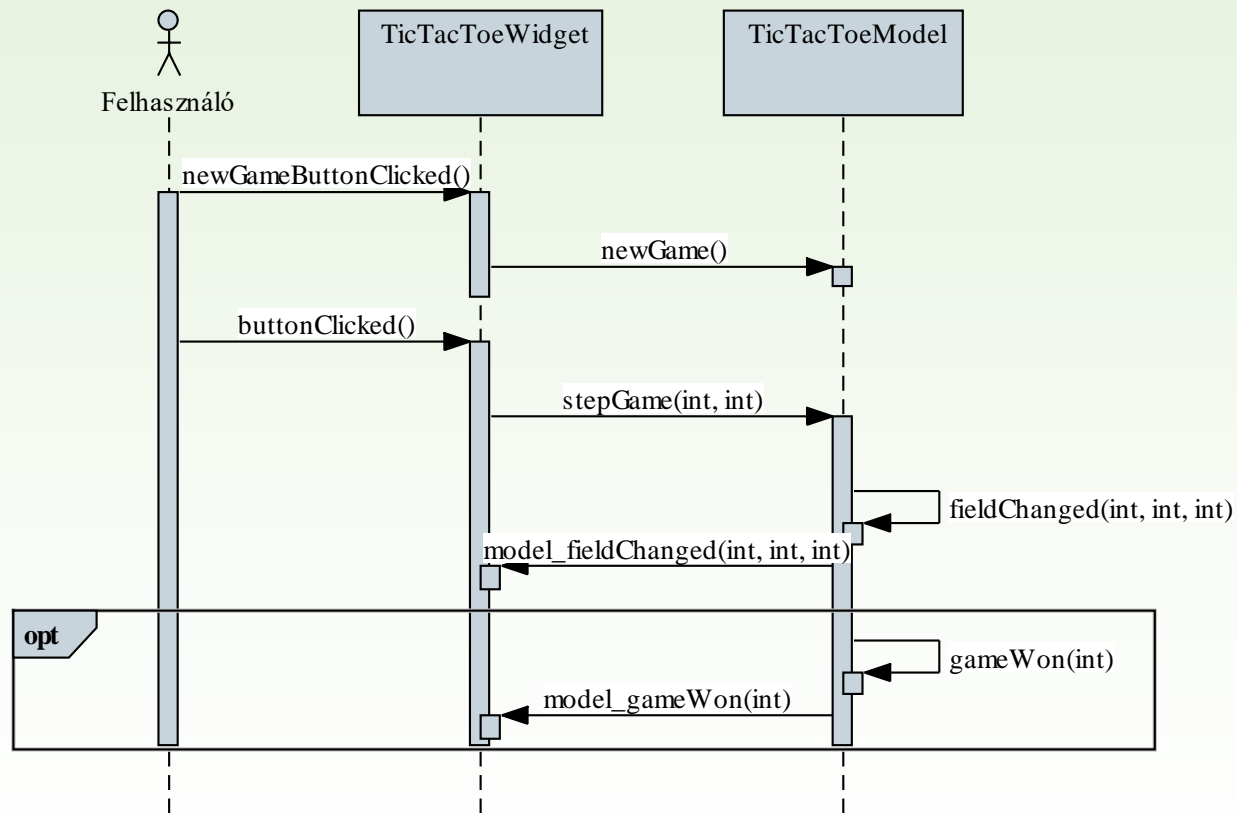
Módosítsuk a Tic-Tac-Toe programot úgy, hogy kétrétegű architektúrában valósuljon meg.

- ❑ A játékért felelős programrészeket a modellt megvalósító osztályba (**TicTacToeModel**) tesszük.
  - csak egész számokkal dolgozunk, függetlenül a felülettől;
  - a játékműveletek publikusak.
  - a modellt megfelelő ellenőrzésekkel kell ellátni.
  - **a modell három szignált vált ki:**
    - mező megváltozása (**fieldChanged**)
    - játék vége valamely játékos győzelmével (**gameWon**)
    - játék vége döntetlennel (**gameOver**)
- ❑ A nézet (**TicTacToeWidget**) aggregálja a modellt, és biztosítja a grafikus megjelenítést, a játék műveleteinek hívását, az eredmények megjelenítését, valamint a modell szignáljainak lekezelését.

# 4.Feladat: tervezés



# 4.Feladat: tervezés



# 4.Feladat: modell

```
void TicTacToeModel::stepGame(int x, int y){
    if (_stepNumber >= 9) return;
    if (x < 0 || x > 2 || y < 0 || y > 2) return;
    if (_gameTable[x][y] != 0) return;
    _gameTable[x][y] = _currentPlayer;
    fieldChanged(x, y, _currentPlayer);
    _stepNumber++;
    _currentPlayer = _currentPlayer % 2 + 1;
    checkGame();
}

void TicTacToeModel::checkGame()
{
    int won = 0;
    ...
    if (won > 0) gameWon(won);
    else if (_stepNumber == 9) gameOver();
}
```

jelzi a nézetnek (szignált küld),  
hogy egy mező megváltozott

szignált küld, ha valaki győzött

szignált küld, ha döntetlen

## 4.Feladat: nézet

---

```
void TicTacToeWidget::buttonClicked()
{
    QPushButton* senderButton =
        dynamic_cast <QPushButton*>(QObject::sender());
    int location = _tableLayout->indexOf(senderButton);
    int x = location / 3;
    int y = location % 3;
    _model.stepGame(x, y); // játék léptetése
}

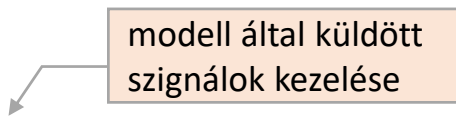
void TicTacToeWidget::newGameButtonClicked()
{
    newGame();
}
```

Értesíti a modellt az aktuális lépésről,  
minden más teendőt a modell végez.

## 4.Feladat: nézet

```
TicTacToeWidget::TicTacToeWidget(QWidget *parent) : QWidget(parent)
{
    ...
    generateTable();
    ...
    connect(&_amp;_model, SIGNAL(gameWon(int)),
            this,    SLOT(model_gameWon(int)));
    connect(&_amp;_model, SIGNAL(gameOver()),
            this,    SLOT(model_gameOver()));
    connect(&_amp;_model, SIGNAL(fieldChanged(int, int, int)),
            this,    SLOT(model_fieldChanged(int, int, int)));

    _model.newGame(); // új játék indítása
}
```



modell által küldött  
szignálok kezelése

## 4.Feladat: nézet

```
void TicTacToeWidget::model_gameWon(int player)
{
    if (player == 1)
        QMessageBox::information(this, tr("Játék vége!"), ... );
    else
        QMessageBox::information(this, tr("Játék vége!"), ... );
    newGame();
}
```

X nyert

Y nyert

```
void TicTacToeWidget::model_gameOver()
{
    QMessageBox::information(this, tr("Játék vége!"), ... );
    newGame();
}
```

döntetlen

```
void TicTacToeWidget::model_fieldChanged(int x, int y, int player)
{
    if (player == 1) _gameTableButtons[x][y]->setText("X");
    else             _gameTableButtons[x][y]->setText("O");
    _gameTableButtons[x][y]->setEnabled(false);
}
```



# Modell újrahasznosítása

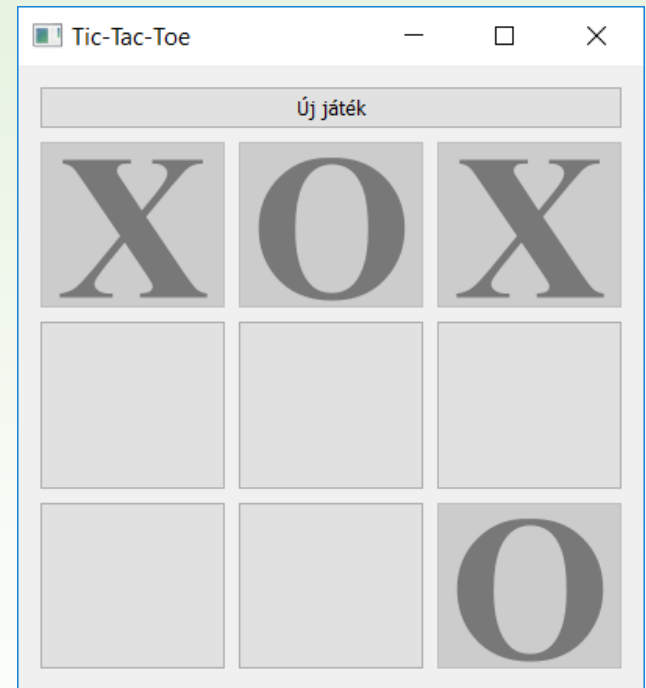
---

- ❑ A modell/nézet architektúrában a modell újrahasznosítható, azaz lecserélhető előre a nézet egy másik nézetre. Ennélfogva az nem tudható előre, milyen módon, milyen körülmények között hívják meg a modell metódusait.
- ❑ A modell elkészítésekor a **modell és a nézet közötti kommunikációban** mindkét irányban törekedni kell a lehető legkevesebb hibalehetőségre:
  - a metódus hívás **paramétereit ellenőrizni** kell, hogy értelmesek-e
  - a modell **állapotát vizsgálni** kell, hogy a metódus tevékenysége végrehajtható-e
  - a kommunikációnak **egyértelműnek** kell lenni (pl. korlátozott értékhalmozra használjunk felsoroló típusokat (**enum**))

# 5.Feladat

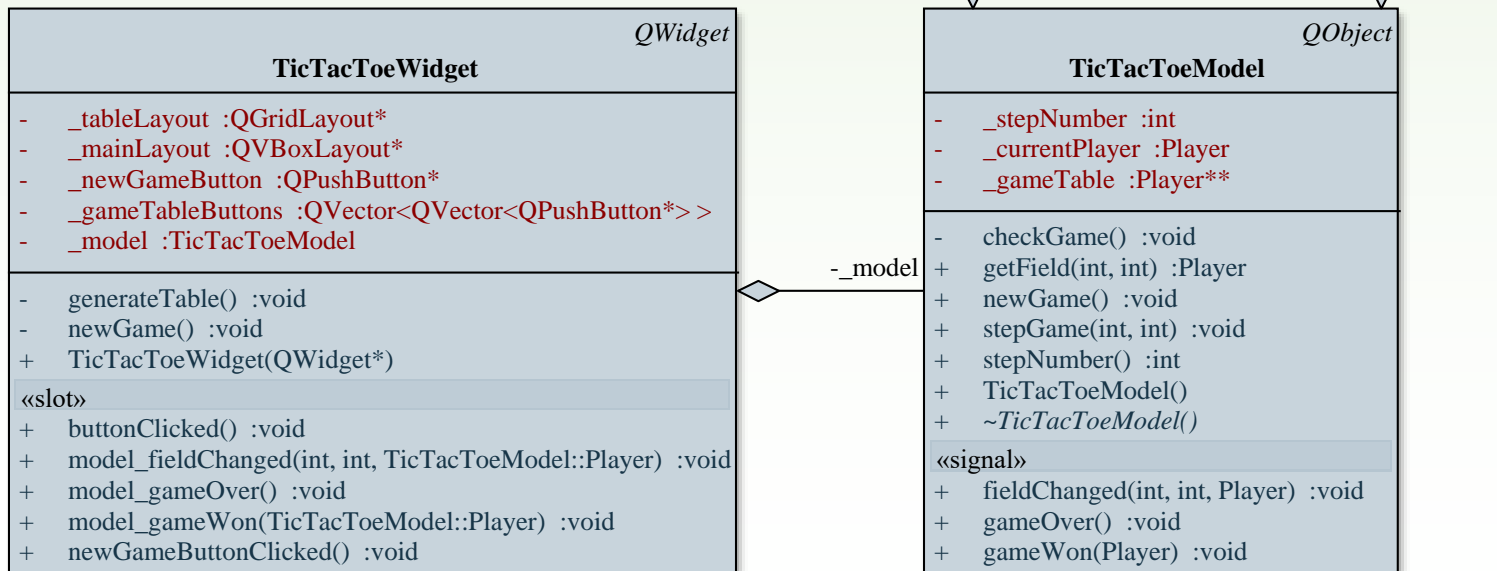
---

Módosítsuk a Tic-Tac-Toe programot úgy, hogy a modellben a játékosokat és a tábla szimbólumait enumeráció segítségével valósítsuk meg.






# 5.Feladat: tervezés

- ❑ 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.



# 5.Feladat: modell

```
TicTacToeModel::TicTacToeModel() {  
    _gameTable = new Player*[3];  
    for (int i = 0; i < 3; ++i)  játékosok tömbje  
        _gameTable[i] = new Player[3];  
}  
  
TicTacToeModel::~~TicTacToeModel() {  
    delete[] _gameTable;  
}  
  
void TicTacToeModel::newGame() {  
    for (int i = 0; i < 3; ++i)  
        for (int j = 0; j < 3; ++j)  játékos szimbólum  
            _gameTable[i][j] = NoPlayer;  
    _stepNumber = 0;  
    _currentPlayer = PlayerX;  
}  
  
TicTacToeModel::Player TicTacToeModel::getField(int x, int y) {  
    if (x < 0 || x > 2 || y < 0 || y > 2) return NoPlayer;  
    return _gameTable[x][y];  játékos szimbólum  
}
```

# 5.Feladat: modell

```
void TicTacToeModel::stepGame(int x, int y){  
    if (_stepNumber >= 9) return;  
    if (x < 0 || x > 2 || y < 0 || y > 2) return;  
    if (_gameTable[x][y] != 0) return;  
  
    _gameTable[x][y] = _currentPlayer;  
    fieldChanged(x, y, _currentPlayer);  
  
    _stepNumber++;  
    _currentPlayer = (Player) (_currentPlayer % 2 + 1);  
    checkGame();  
}
```

ellenőrzések

// ellenőrizzük a lépésszámot  
// ellenőrizzük a tartomány  
// ellenőrizzük a mezőt

szignál küldése a nézetnek

konverzió

# 5.Feladat: modell

```
void TicTacToeModel::checkGame() {
    Player won = NoPlayer;
    for(int i = 0; i < 3; ++i) {
        if (_gameTable[i][0] != 0 && _gameTable[i][0] == _gameTable[i][1]
            && _gameTable[i][1] == _gameTable[i][2])
            won = _gameTable[i][0];
    }
    for(int i = 0; i < 3; ++i) {
        if (_gameTable[0][i] != 0 && _gameTable[0][i] == _gameTable[1][i]
            && _gameTable[1][i] == _gameTable[2][i])
            won = _gameTable[0][i];
    }
    if (_gameTable[0][0] != 0 && _gameTable[0][0] == _gameTable[1][1]
        && _gameTable[1][1] == _gameTable[2][2])
        won = _gameTable[0][0];
    if (_gameTable[0][2] != 0 && _gameTable[0][2] == _gameTable[1][1]
        && _gameTable[1][1] == _gameTable[2][0])
        won = _gameTable[0][2];

    if (won != NoPlayer) gameWon(won);
    else if (_stepNumber == 9) gameOver();
}
```



# 5.Feladat: nézet

```
void TicTacToeWidget::model_gameWon(TicTacToeModel::Player player){
    char* str;
    switch (player){
        case TicTacToeModel::PlayerX: str = "Az X nyerte a játékot!"; break;
        case TicTacToeModel::PlayerO: str = "Az O nyerte a játékot!"; break;
    }
    QMessageBox::information(this, tr("Játék vége!"), tr(str));
    newGame();
}
```

játékos szimbólum

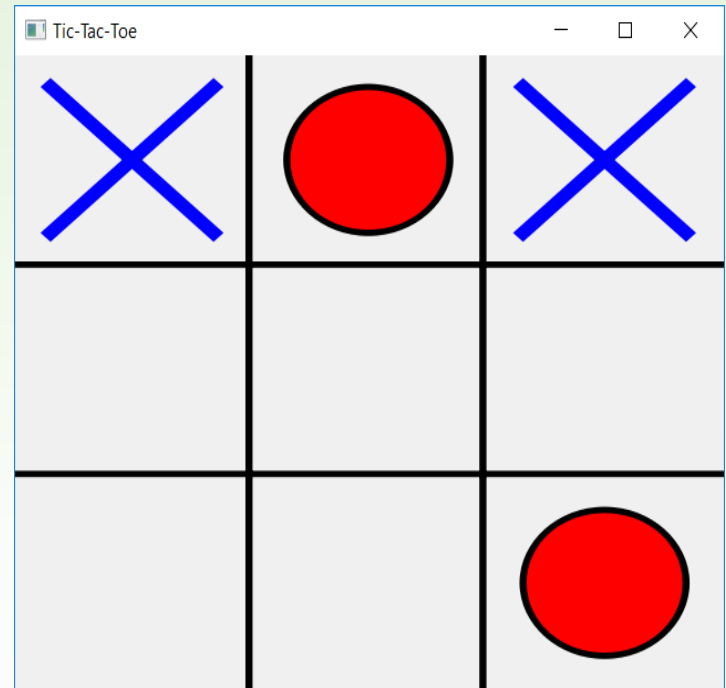
```
void TicTacToeWidget::model_fieldChanged(int x, int y,
TicTacToeModel::Player player){
    switch (player){
        case TicTacToeModel::PlayerX:
            _gameTableButtons[x][y]->setText("X");
            _gameTableButtons[x][y]->setEnabled(false);
            break;
        case TicTacToeModel::PlayerO: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

játékos szimbólum

# 6.Feladat

---

Módosítsuk a Tic-Tac-Toe programot úgy, hogy változtassuk a nézetet, és a felületen alakzatok rajzaival jelenítsük meg a játékállást.



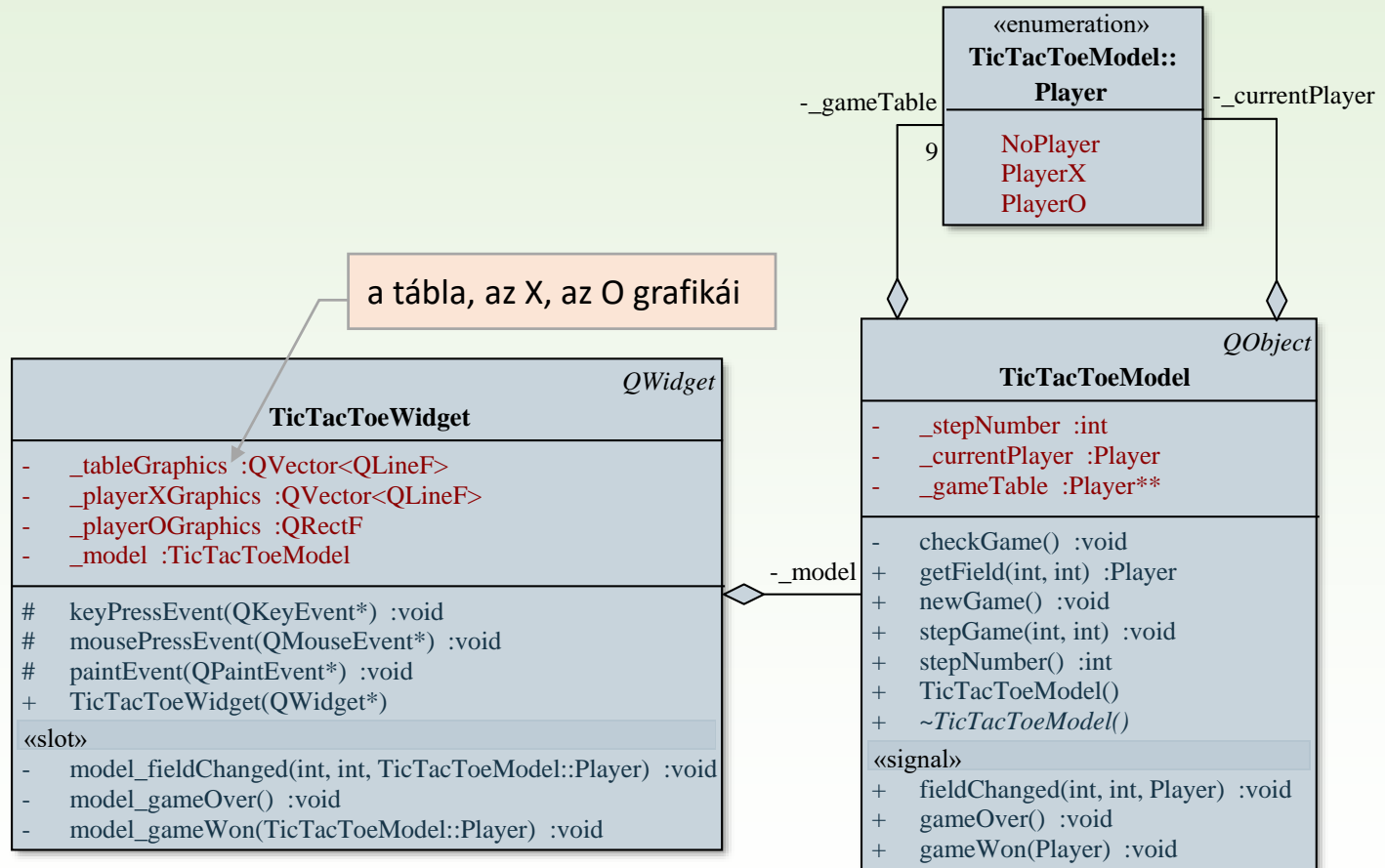


# 6.Feladat: tervezés

---

- ❑ Csak a nézet változik.
- ❑ A képernyőről levesszük az összes vezérlőt, a megjelenítését rajzolás segítségével (**QPainter**) valósítjuk meg.
  - A játékosok egér segítségével foglalhatják el a mezőket, új játékot pedig a **Ctrl+N** billentyűkombinációval indíthatnak.
  - Ehhez felüldefiniáljuk a billentyű- és egérlenyomás eseménykezelőket (**keyPressEvent**, **mousePressEvent**).

# 6.Feladat: tervezés



# 6.Feladat: nézet konstruktora

---

```
TicTacToeWidget::TicTacToeWidget(QWidget *parent) : QWidget(parent)
{
    setMinimumSize(400, 400);
    setBaseSize(400,400);
    setWindowTitle(tr("Tic-Tac-Toe"));
    // mezők grafikája:
    _tableGraphics.append(QLineF(0, 66, 200, 66));
    _tableGraphics.append(QLineF(0, 132, 200, 132));
    _tableGraphics.append(QLineF(66, 0, 66, 200));
    _tableGraphics.append(QLineF(132, 0, 132, 200));
    // játékosok jeleinek grafikái:
    _playerXGraphics.append(QLineF(10, 10, 56, 56));
    _playerXGraphics.append(QLineF(10, 56, 56, 10));
    _playerOGraphics = QRectF(10.0, 10.0, 46.0, 46.0);
    ...
}
```

## 6.Feladat: nézet konstruktora

---

```
TicTacToeWidget::TicTacToeWidget(QWidget *parent) : QWidget(parent)
{
    ...
    // modell eseményeinek feldolgozása
    connect( &_model, SIGNAL(gameWon(TicTacToeModel::Player)),
             this,    SLOT(model_gameWon(TicTacToeModel::Player)));
    connect( &_model, SIGNAL(gameOver()),
             this,    SLOT(model_gameOver()));
    connect( &_model, SIGNAL(fieldChanged(int,int,TicTacToeModel::Player)),
             this,    SLOT(model_fieldChanged(int,int, TicTacToeModel::Player)));

    _model.newGame(); // új játék indítása
}
```

# 6.Feladat: nézet eseménykezelése

```
void TicTacToeWidget::model_gameWon(TicTacToeModel::Player player)
{
    char* str;
    switch (player){
        case TicTacToeModel::PlayerX: str = "Az X nyerte a játékot!"; break;
        case TicTacToeModel::PlayerO: str = "Az O nyerte a játékot!"; break;
    }
    QMessageBox::information(this, tr("Játék vége!"), tr(str));
    _model.newGame();
}
```

```
void TicTacToeWidget::model_fieldChanged(int x, int y,
                                          TicTacToeModel::Player player)
```

```
{
    update();
}
```

```
void TicTacToeWidget::model_gameOver()
{
    QMessageBox::information(this, tr("Játék vége!"),
                           tr("A játék döntetlen lett!"));
    _model.newGame();
}
```

## 6.Feladat: nézet paintEvent()

---

```
void TicTacToeWidget::paintEvent(QPaintEvent *)
{
    QPainter painter(this); // rajzoló objektum
    painter.setRenderHint(QPainter::Antialiasing); // élsimítás
    painter.scale(width() / 200.0, height() / 200.0); // skálázás

    painter.setPen(QPen(Qt::black, 2));
    painter.setBrush(Qt::red);
    painter.drawLines(_tableGraphics); // tábla kirajzolása

    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            painter.save(); // elmentjük a rajztulajdonságokat
            ...
        }
    }
}
```

## 6.Feladat: nézet paintEvent()

```
void TicTacToeWidget::paintEvent(QPaintEvent *)
{
    ...
    painter.translate(i * 200.0 / 3 , j * 200.0 / 3);
    switch (_model.getField(i, j)){ // mező kirajzolása
        case TicTacToeModel::PlayerX:
            painter.setPen(QPen(Qt::blue, 4));
            painter.drawLines(_playerXGraphics);
            break;
        case TicTacToeModel::PlayerO:
            painter.setPen(QPen(Qt::black, 2));
            painter.drawEllipse(_playerOGraphics);
            break;
    }
    painter.restore(); // visszatöltjük a korábbi állapotot
}
}
```

elmozdítja a rajzpontot a megfelelő mezőre

# 6.Feladat: a nézet billentyű- és egérkezelése

```
void TicTacToeWidget::keyPressEvent(QKeyEvent *event) {  
    if (event->key() == Qt::Key_N &&  
        QApplication::keyboardModifiers() == Qt::ControlModifier) {  
        _model.newGame();  
        update();  
    }  
}  
  
void TicTacToeWidget::mousePressEvent(QMouseEvent *event) {  
    int x = event->pos().x() * 3 / width();  
    int y = event->pos().y() * 3 / height();  
  
    _model.stepGame(x, y);  
}
```

lekezezi a Ctrl+N kombinációt

az event->pos() megadja az egérpozíciót, ami QPoint típusú, ebből kiszámolható, melyik mezőn vagyunk:



# Tesztelés

# Tesztelés célja és módja

---

- ❑ A tesztelés célja a szoftverhibák felfedezése és a szoftverrel szemben támasztott minőségi elvárások ellenőrzése.
- ❑ A tesztelés során különböző **teszteseteket** (*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*).

# Tesztelés szakaszai

---

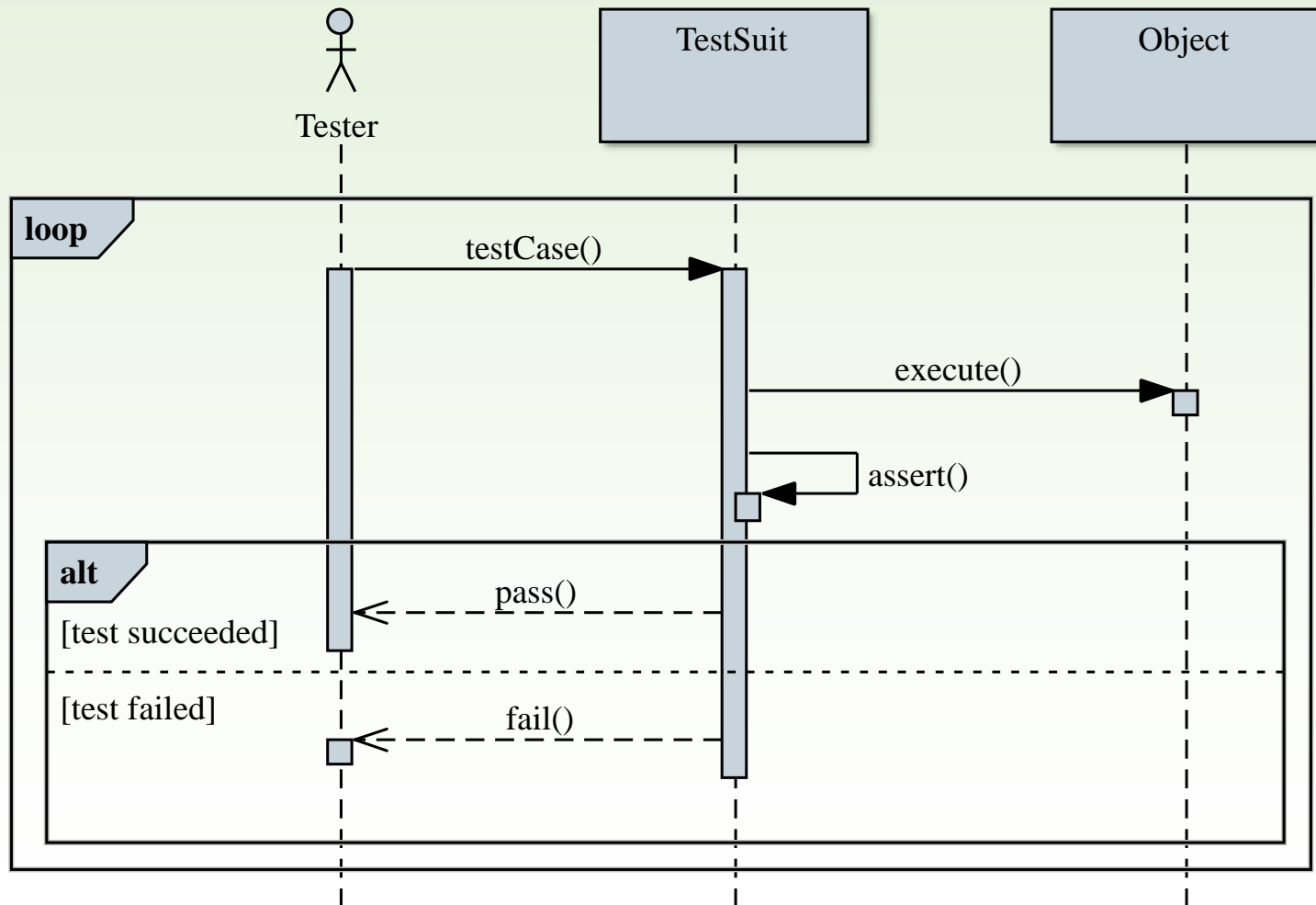
- ❑ A tesztelés nem a teljes program elkészülte után, egyben történik, hanem általában 3 lépésből áll: **fejlesztői teszt** (*development testing*), **kiadásteszt** (*release testing*), **felhasználói teszt** (*acceptance testing*).
- ❑ A fejlesztői tesztnek további három szakasza van:
  - **egységteszt** (*unit test*): a programegységeket (osztályok, metódusok) külön-külön, egymástól függetlenül teszteljük
  - **integrációs teszt** (*integration test*): a programegységek együttműködésének tesztje, a rendszer egy komponensének vizsgálata
  - **rendszer teszt** (*system test*): az egész rendszer együttes tesztje, a rendszert alkotó komponensek közötti kommunikáció vizsgálata

# Automatikus tesztelés

---

- ❑ A tesztelés egy része automatizálható, bizonyos részét azonban mindenképpen manuálisan kell végrehajtanunk.
- ❑ 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 fejlesztésnek (*Test Driven Development, TDD*).

# Egységtesztek



# Tesztelés Qt keretrendszerben

---

- ❑ A Qt keretrendszer tartalmaz egy beágyazott tesztelő modult (*QTestLib*), amely lehetőségeket ad egységtesztek és teljesítménytesztek könnyű megfogalmazására, és végrehajtására.
  - A tesztekhez szükséges funkciókat a `QTest` könyvtárban találjuk.
  - A tesztkörnyezetet `QObject` leszármazott osztályokban valósítjuk meg (amelyeket ellátunk `Q_OBJECT` makróval).
  - A **tesztesetek eseménykezelők** lesznek, amelyekben ellenőrzéseket végzünk. Az eseménykezelőket előidéző szignálokat a tesztkörnyezet váltja ki.
  - A projektben megjelöljük a modul használatát (`QT += testlib`).

# Makrók és futtatás

---

- ❑ 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 a `QTEST_APPLESS_MAIN(<osztálynév>)` makró végzi, amely automatikusan legenerál egy főprogramot, és végrehajtja a teszteseteket (kiváltja a teszteseteket tartalmazó eseménykezelőket előidéző szignálokat), így a tesztek egyszerű konzolos alkalmazásként futtathatók.

# Példa tesztelendő osztályra

---

```
class MyClass {  
private:  
    int _value;  
public:  
    MyClass (int v) { _value = v; }  
  
    void add(int v) { _value += v; }  
    int getValue() const { return _value; }  
}
```

tesztelendő osztály

publikus metódusokat teszteljük



# Példa tesztkörnyezetre

```
class MyClassTest : QObject {
    Q_OBJECT
private slots:
    void testGetValue() {
        MyClass mc(10);
        QVERIFY(mc.getValue() == 10);
        // másképp: QCOMPARE(mc.getValue(), 10);
    }
    void testAdd() {
        MyClass mc(10);
        mc.add(5);
        QCOMPARE(mc.getValue(), 15);
        mc.add(15);
        QCOMPARE(mc.getValue(), 30);
    }
    ...
}
```

tesztkörnyezet

tesztesetek, mint eseménykezelők

tetszőleges sok ellenőrzést végezhetünk

# Tesztprojekt

---

- ❑ 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, tesztetenként láthatjuk az eredményt, az esetleges hibajelenséget, valamint a hiba helyét:

```
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
```

# Tesztkörnyezet beállítása

---

- ❑ Lehetőségünk van a tesztkörnyezet konfigurálására:
  - A tesztkörnyezetet adó osztály **adattagjaként** bármilyen adatot eltárolhatunk.
  - Az adattagok értékét **speciális eseménykezelővel** állíthatjuk:
    - az első teszteset előtt 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**)

# Példa tesztkörnyezet beállítására

```
class MyClassTest : QObject {  
    Q_OBJECT  
private:  
    MyClass* _mc;  
private slots:  
    void initTestCase() {  
        _mc = new MyClass(10);  
    }  
    void cleanupTestCase() {  
        delete _mc;  
    }  
    ...  
}
```

tesztkörnyezet értékei

inicializálás

megsemmisítés

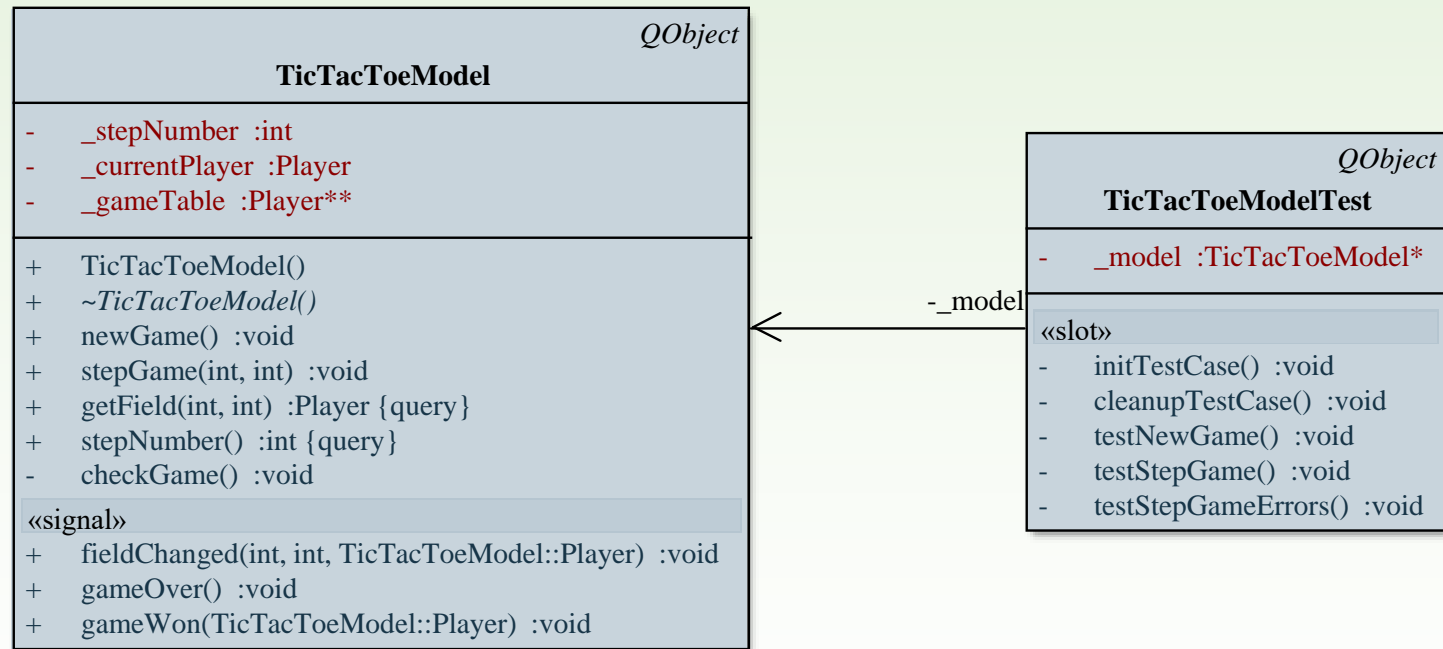
# Feladat

---

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

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

# Feladat: tervezés



# Teszt osztály

---

```
#include <QtTest>
#include "tictactoemodel.h"
class TicTacToeModelTest : public QObject{
    Q_OBJECT
private:
    TicTacToeModel* _model;
private slots:
    void initTestCase();
    void cleanupTestCase();
    void testNewGame();
    void testStepGame();
    void testStepGameErrors();
};

...

QTEST_APPLESS_MAIN(TicTacToeModelTest)
#include "tictactoemodeltest.moc"
```

# Teszt osztály metódusai

```
void TicTacToeModelTest::initTestCase() {  
    _model = new TicTacToeModel();  
}
```

tesztkörnyezet létrehozása

```
void TicTacToeModelTest::cleanupTestCase() {  
    delete _model;  
}
```

tesztkörnyezet megsemmisítése

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

tesztesetek

ellenőrizzük, hogy kezdetben minden mező üres, és a lépésszám 0



# Teszt osztály metódusai

```
void TicTacToeModelTest::testStepGame() {
```

```
    _model->newGame();
```

```
    _model->stepGame(0, 0);
```

```
    QCOMPARE(_model->stepNumber(), 1);
```

```
    QCOMPARE(_model->getField(0, 0), TicTacToeModel::PlayerX);
```

```
    for (int i = 0; i < 3; i++)
```

```
        for (int j = 0; j < 3; j++)
```

```
            QVERIFY((i == 0 && j == 0) ||  
                    (_model->getField(i, j) == TicTacToeModel::NoPlayer));
```

```
    _model->stepGame(0, 1);
```

```
    QCOMPARE(_model->stepNumber(), 2);
```

```
    QCOMPARE(_model->getField(0, 1), TicTacToeModel::PlayerO);
```

```
    _model->stepGame(0, 2);
```

```
    QCOMPARE(_model->stepNumber(), 3);
```

```
    QCOMPARE(_model->getField(0, 2), TicTacToeModel::PlayerX);
```

```
}
```

ellenőrizzük, hogy kezdetben minden mező üres, és a lépésszám 0

ellenőrizzük, hogy közben más mező nem változott

ellenőrizzük, hogy ezután O következik

majd ismét az X

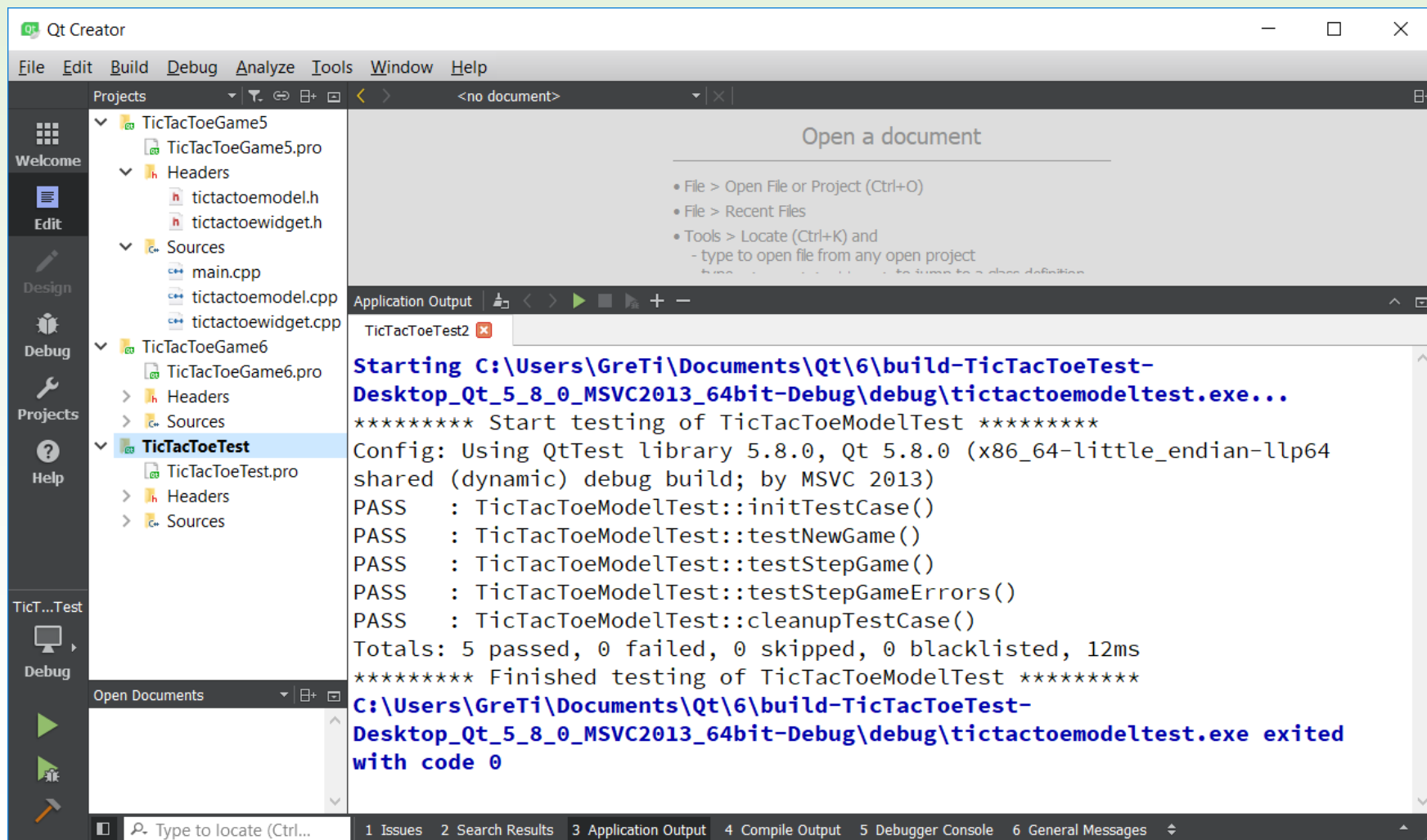
# Teszt osztály metódusai

```
void TicTacToeModelTest::testStepGameErrors() {
    _model->newGame();
    _model->stepGame(-1, 0);
    _model->stepGame(0, -1);
    _model->stepGame(3, 0);
    _model->stepGame(0, 3);
    QCOMPARE(_model->stepNumber(), 0);
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            QVERIFY(_model->getField(i, j) == TicTacToeModel::NoPlayer);
    _model->stepGame(0, 0);
    _model->stepGame(0, 0);
    QCOMPARE(_model->stepNumber(), 1);
    QCOMPARE(_model->getField(0, 0), TicTacToeModel::PlayerX);
}
```

ellenőrizzük, hogy nem tudunk rossz mezőre lépni

ellenőrizzük, hogy kétszer nem tudunk lépni ugyanarra a mezőre

# Teszt eredmények



# Teszt eredmények

The screenshot shows the Qt Creator IDE with the following components:

- Left Sidebar:** Contains icons for Welcome, Edit, Design, Debug, Projects, and Help. The 'Projects' view is active, showing a tree structure with 'TicTacToeGame5', 'TicTacToeGame6', and 'TicTacToeTest'. The 'TicTacToeTest' project is expanded, showing its headers and sources.
- Top Bar:** Displays the menu bar (File, Edit, Build, Debug, Analyze, Tools, Window, Help) and the current document status ('<no document>').
- Application Output Window:** Shows the output of the test execution. The text is as follows:

```
Starting C:\Users\GreTi\Documents\Qt\6\build-TicTacToeTest-Desktop_Qt_5_8_0_MSVC2013_64bit-Debug\debug\tictactoemodeltest.exe...
***** Start testing of TicTacToeModelTest *****
Config: Using QTest library 5.8.0, Qt 5.8.0 (x86_64-little_endian-llp64
shared (dynamic) debug build; by MSVC 2013)
PASS  : TicTacToeModelTest::initTestCase()
PASS  : TicTacToeModelTest::testNewGame()
FAIL!  : TicTacToeModelTest::testStepGame() Compared values are not the same
    Actual   (_model->stepNumber()): 2
    Expected (1)                   : 1
..\..\06\TicTacToeTest1\tictactoemodeltest.cpp(58) : failure location
PASS  : TicTacToeModelTest::testStepGameErrors()
PASS  : TicTacToeModelTest::cleanupTestCase()
Totals: 4 passed, 1 failed, 0 skipped, 0 blacklisted, 1ms
***** Finished testing of TicTacToeModelTest *****
C:\Users\GreTi\Documents\Qt\6\build-TicTacToeTest-Desktop_Qt_5_8_0_MSVC2013_64bit-Debug\debug\tictactoemodeltest.exe exited
with code 1
```
- Bottom Bar:** Contains a search bar and a list of tabs: 1 Issues, 2 Search Results, 3 Application Output (active), 4 Compile Output, 5 Debugger Console, 6 General Messages.