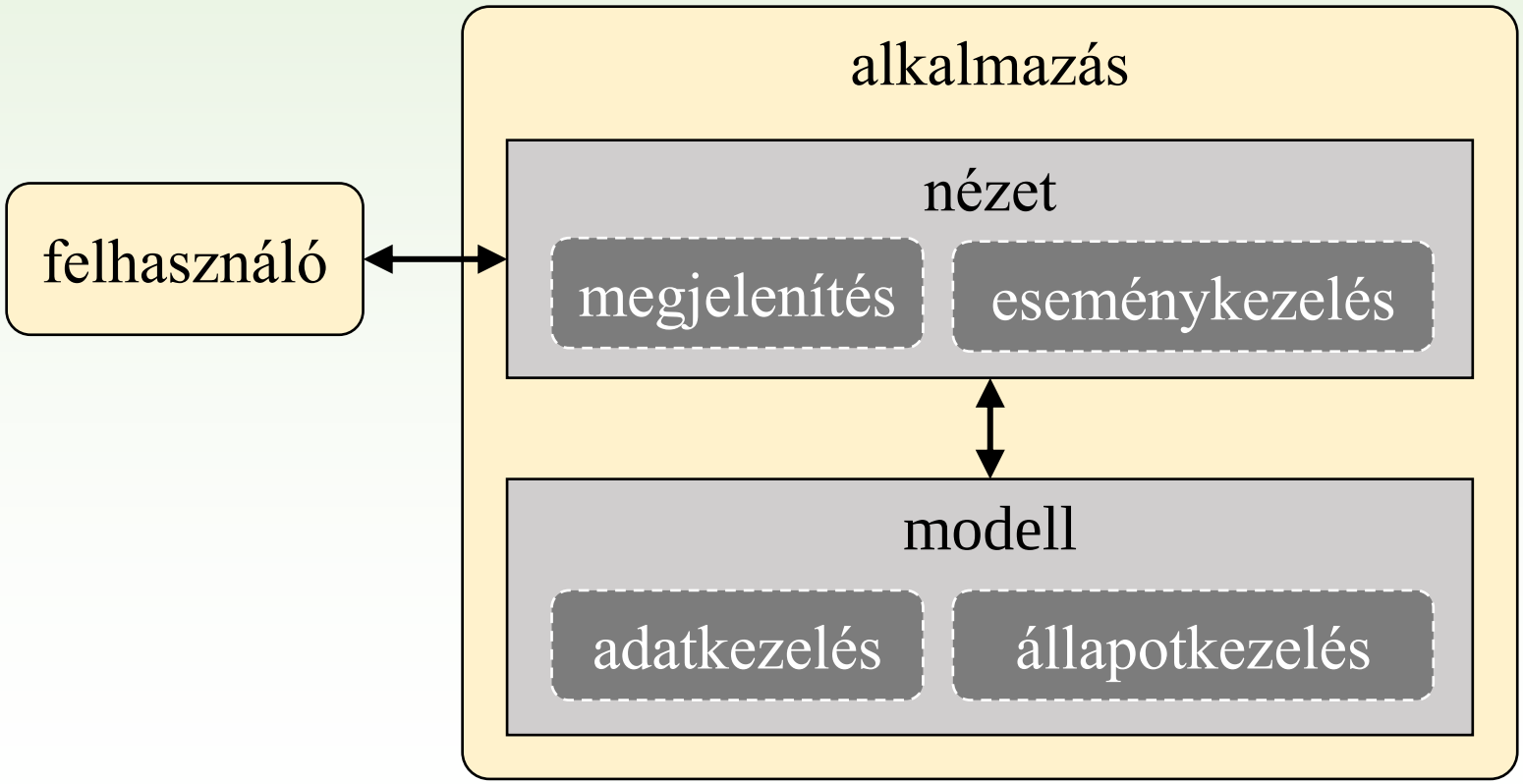
Modell-nézet architektúra

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 *alkalmazáslogika*nak, vagy *üzleti logika*nak.
  - 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 szolgáltat.

M/V architektúra kommunikációja

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

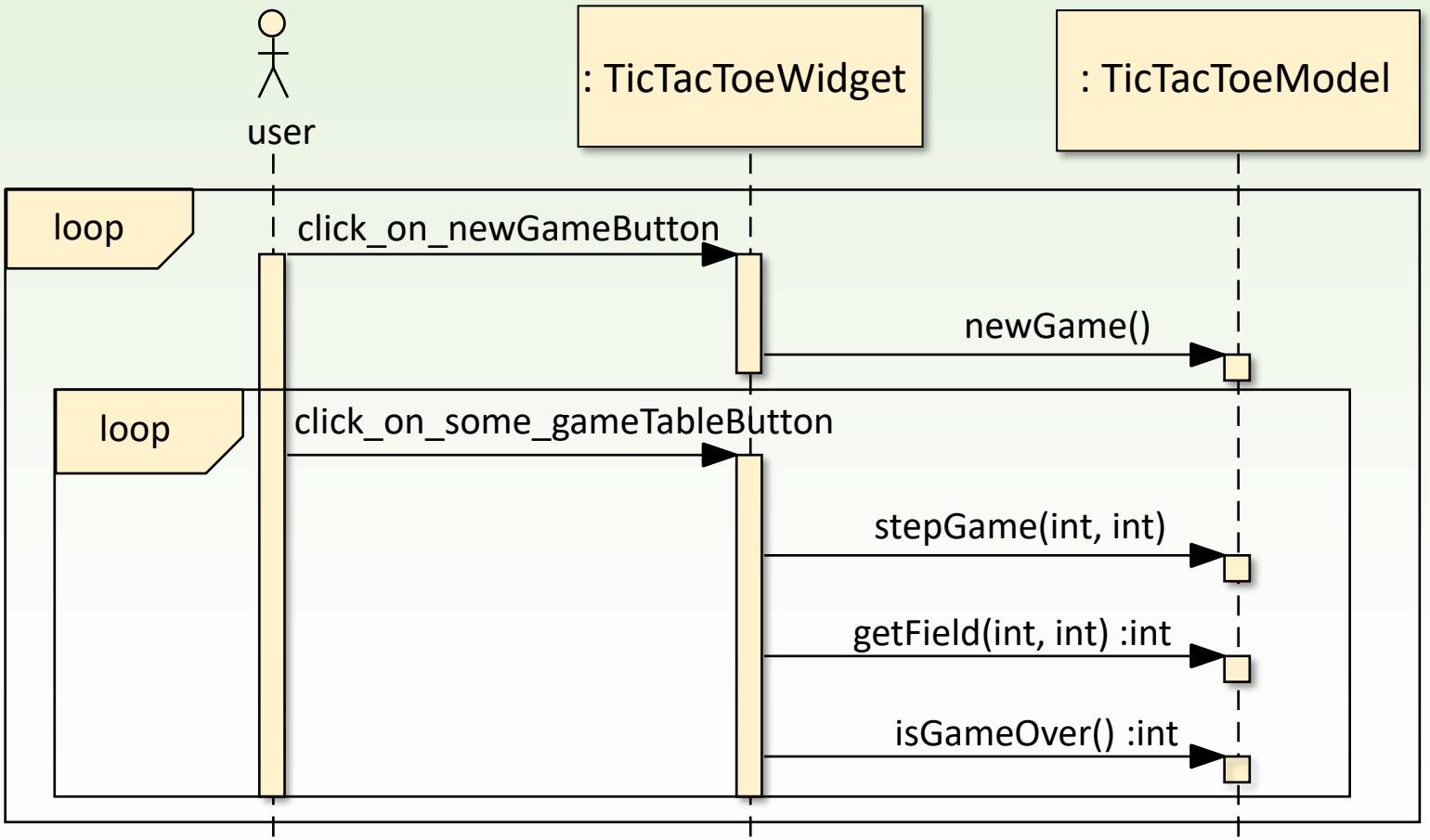


3.Megoldás

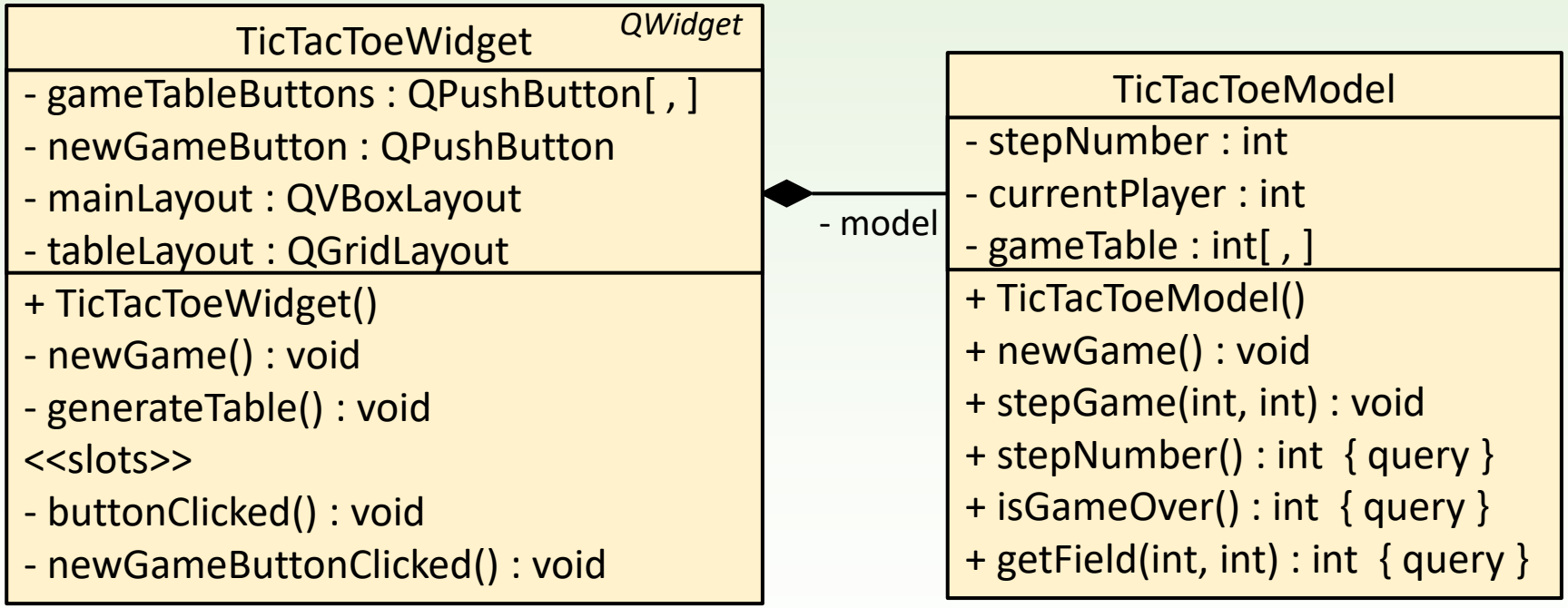
Helyezzük a Tic-Tac-Toe alkalmazást kétrétegű architektúrába.

- A játékért felelős programrészek a modellt megvalósító osztályba (**`TicTacToeModel`**) kerülnek, ahol
  - a játéktábla elemeit, függetlenül a felülettől, egész számokkal jelöljük
  - a játékműveletek publikusak, azokat a felületért felelős kód hívja.
  - a modellt ellenőrzésekkel kell ellátni (mivel leválasztottuk a tevékenységeit).
- A nézet (**`TicTacToeWidget`**) aggregálja a modellt, és biztosítja a grafikus megjelenítést, a felhasználó jelzéseire reagálva hívja a modell műveleteit, felel az eredmények megjelenítéséért.

3.Megoldás: elemzés



3.Megoldás: tervezés



3.Megoldás: 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) const
{
    if (x < 0 || x > 2 || y < 0 || y > 2) return 0;
    return _gameTable[x][y];
}
int TicTacToeModel::stepNumber() const { return _stepNumber; }
```

A játéktábla mátrixát a konstruktor hozza létre,
a destruktor szünteti meg

getter-eket kell bevezetni a a modell állapotának lekérdezéséhez

3.Megoldás: modell

Ezekkel a kódrészekkel már találkoztunk, de teljesen eltűntek a felületre vonatkozó utasítások.

```
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.Megoldás: modell

```
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.Megoldás: nézet

```
TicTacToeWidget::TicTacToeWidget(QWidget *parent) : QWidget(parent)
{
    …
    _model = new TicTacToeModel();
    _model->newGame(); // új játék indítása
}
```

A nézet hívja a modell megfelelő metódusát.

```
TicTacToeWidget::~TicTacToeWidget()
{
    delete _model;
}
```

3.Megoldás: nézet

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

A nézet hívja a modell megfelelő metódusát.

3.Megoldás: 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 hívja a modell megfelelő metódusát.

kettős könyvelés

M/V architektúra megvalósítása

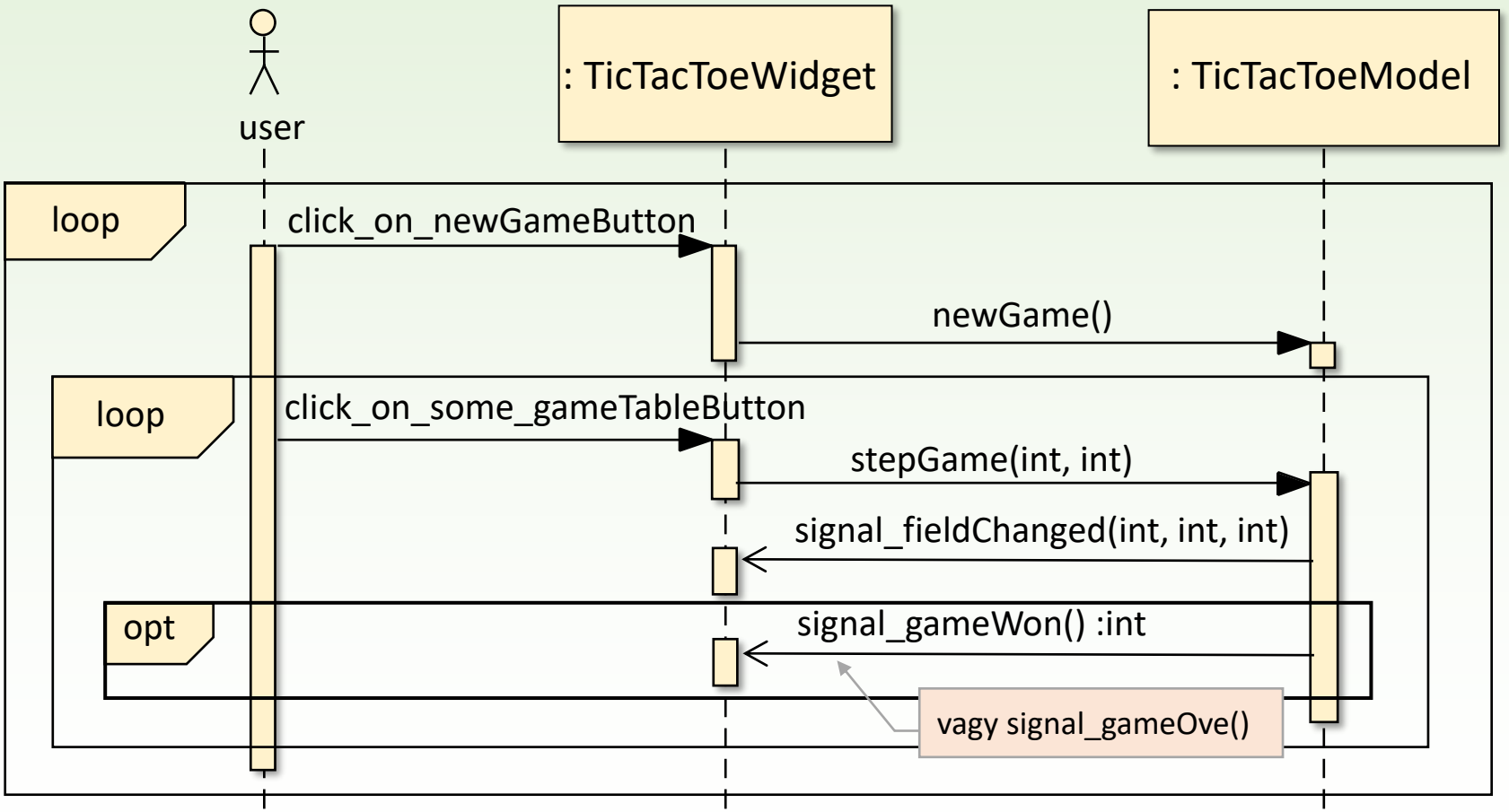
- A modell és a nézet kapcsolatát úgy kell megvalósítani, hogy ne a nézet, hanem a modell vezérelje az alkalmazást, anélkül, hogy ismernie kelljen nézetet:
  - a modell szignálok küldésével kommunikálhat a nézettel, anélkül hogy a nézetre hivatkozása lenne a nézet reagálni tud a szignálokra
  - a nézet továbbra is hozzáfér a modell publikus elemeihez (ismeri annak interfészét, hívhatja publikus metódusait)

4.Megoldás

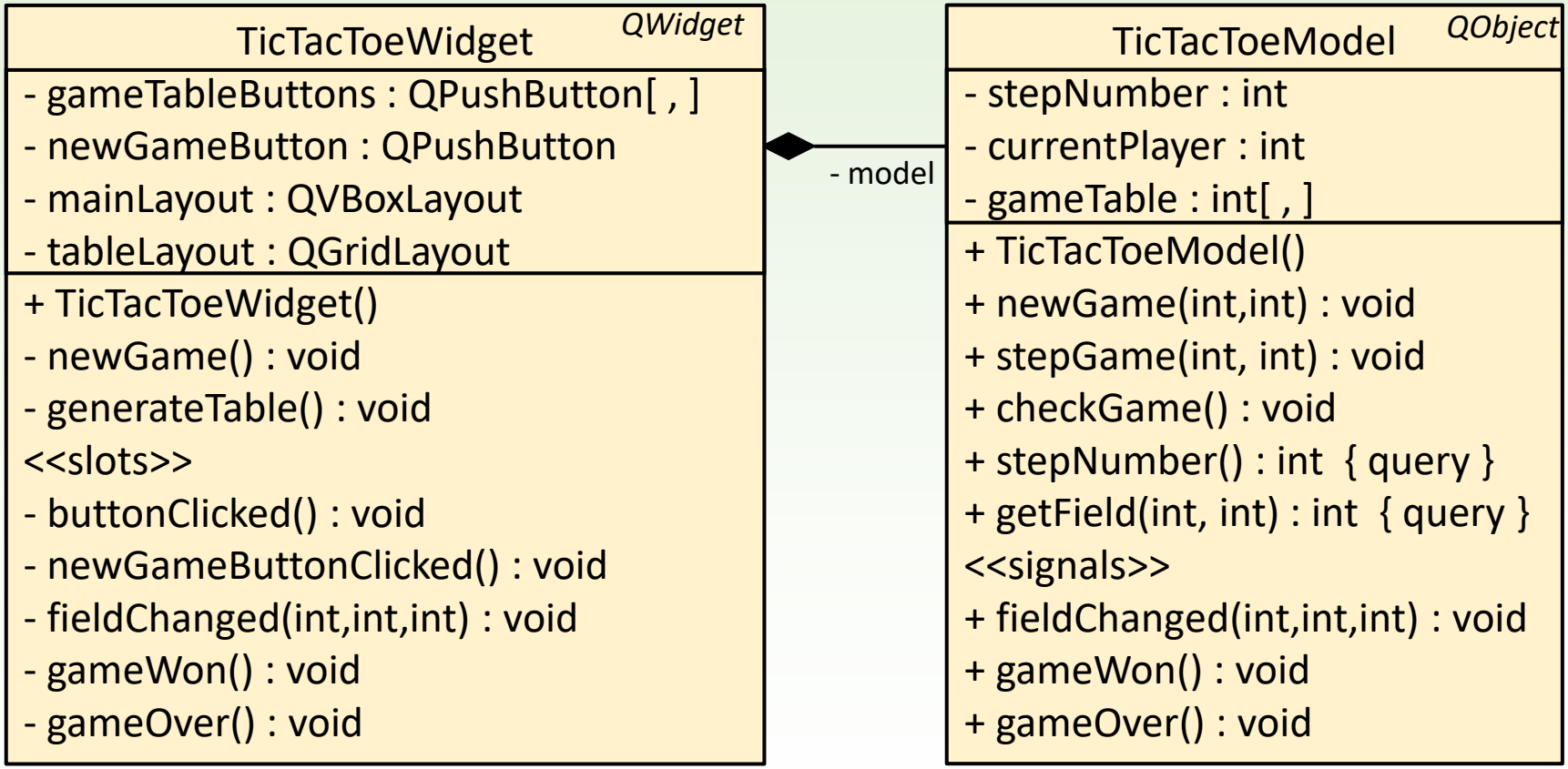
Módosítsuk a Tic-Tac-Toe programot úgy, hogy a modell vezérelje az alkalmazást.

- A modellt megvalósító osztályban (**`TicTacToeModel`**) három szignál kiváltására adunk lehetőséget:
  - mező megváltozása (**`fieldChanged`**)
  - játék vége valamely játékos győzelmével (**`gameWon`**)
  - játék vége döntetlennel (**`gameOver`**)
- A nézet (**`TicTacToeWidget`**) metódusaiból teljesen kikerül a játéklogika, és kiegészül a modell szignáljainak lekezelésével.

4.Megoldás: elemzés



4.Megoldás: tervezés



4.Megoldás: 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.Megoldás: nézet

```
TicTacToeWidget::TicTacToeWidget(QWidget *parent) : QWidget(parent)
{
    …
    _mode = new TicTacToeModel();

// modell eseményeinek feldolgozása
    connect(_model, SIGNAL(gameWon(TicTacToeModel::Player)),
            this,   SLOT   (gameWon(TicTacToeModel::Player)));
    connect(_model, SIGNAL(gameOver()),
            this,   SLOT   (gameOver()));
    connect(_model, SIGNAL(fieldChanged(int,int,TicTacToeModel::Player)),
            this,   SLOT   (fieldChanged(int,int, TicTacToeModel::Player)));

    _model->newGame(); // új játék indítása
}
```

a nézet felkészül a modell szignáljainak fogadására kelli

4.Megoldás: 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.Megoldás: nézet

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

döntetlen

```
void TicTacToeWidget::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);
}
```

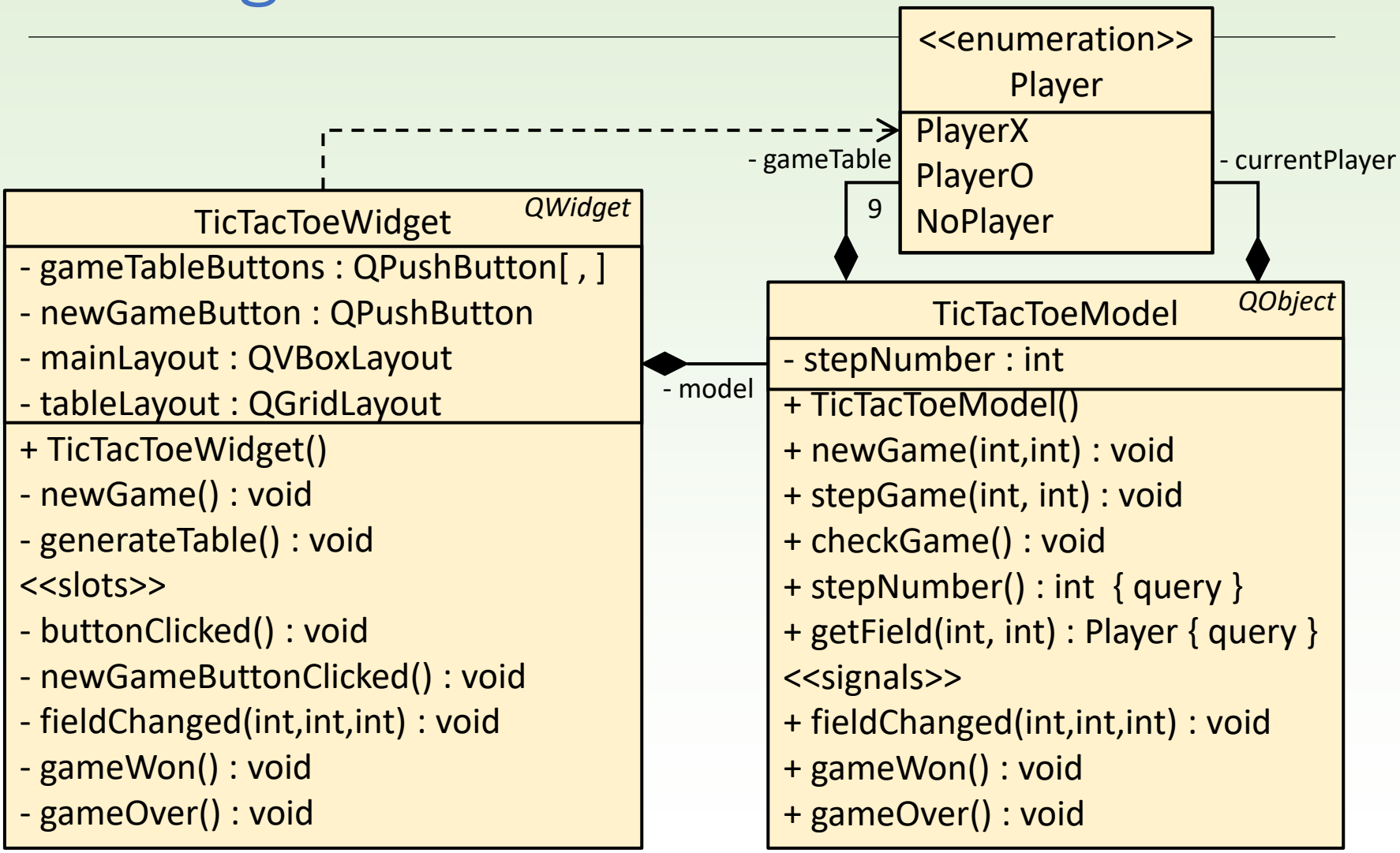
Modell újrahasznosítása

- A modell/nézet architektúrában a modell újrahasznosítható, azaz lecserélhető előle 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 értékhalmazra használjunk felsoroló típusokat (**`enum`**))

5.Megoldás

- 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.Megoldás: tervezés



5.Megoldás: modell

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

játékosok tömbje

játékos szimbólum

játékos szimbólum

5.Megoldás: 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

konverzió

5.Megoldás: modell

```
void TicTacToeModel::checkGame(){
     Player won = NoPlayer;
     for(int i = 0; i < 3; ++i) {
          if (_gameTable[i][0] != NoPlayer
                    && _gameTable[i][0] ==  _gameTable[i][1]
                    && _gameTable[i][1] == _gameTable[i][2])
               won = _gameTable[i][0];
     }
     …
     if (won != NoPlayer) gameWon(won);
     else if (_stepNumber == 9) gameOver();
}
```

játékos szimbólum

5.Megoldás: nézet

```
void TicTacToeWidget::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;
        case TicTacToeModel::NoPlayer: break;
    }
    QMessageBox::information(this, tr("Játék vége!"), tr(str));
    _model->newGame()
}
```

játékos szimbólum

```
void TicTacToeWidget::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;
        case TicTacToeModel::NoPlayer: break;
    }
}
```

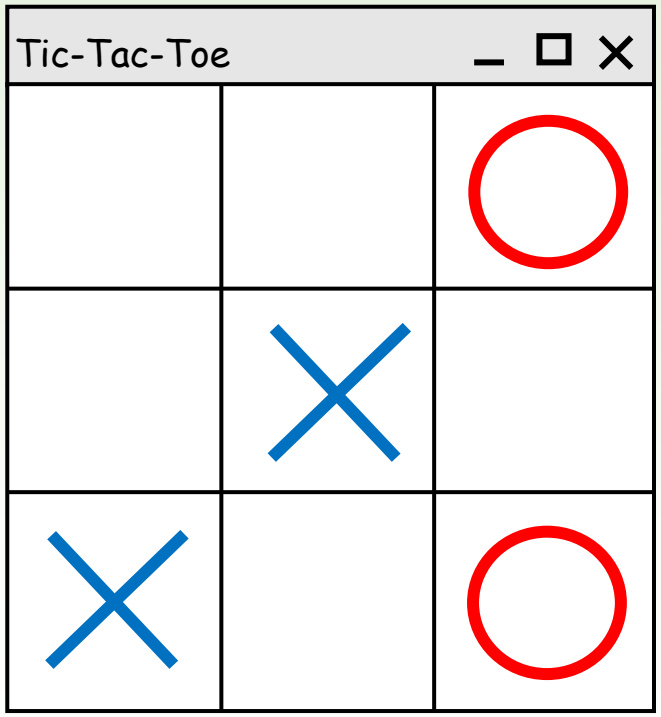
játékos szimbólum

6.Megoldás

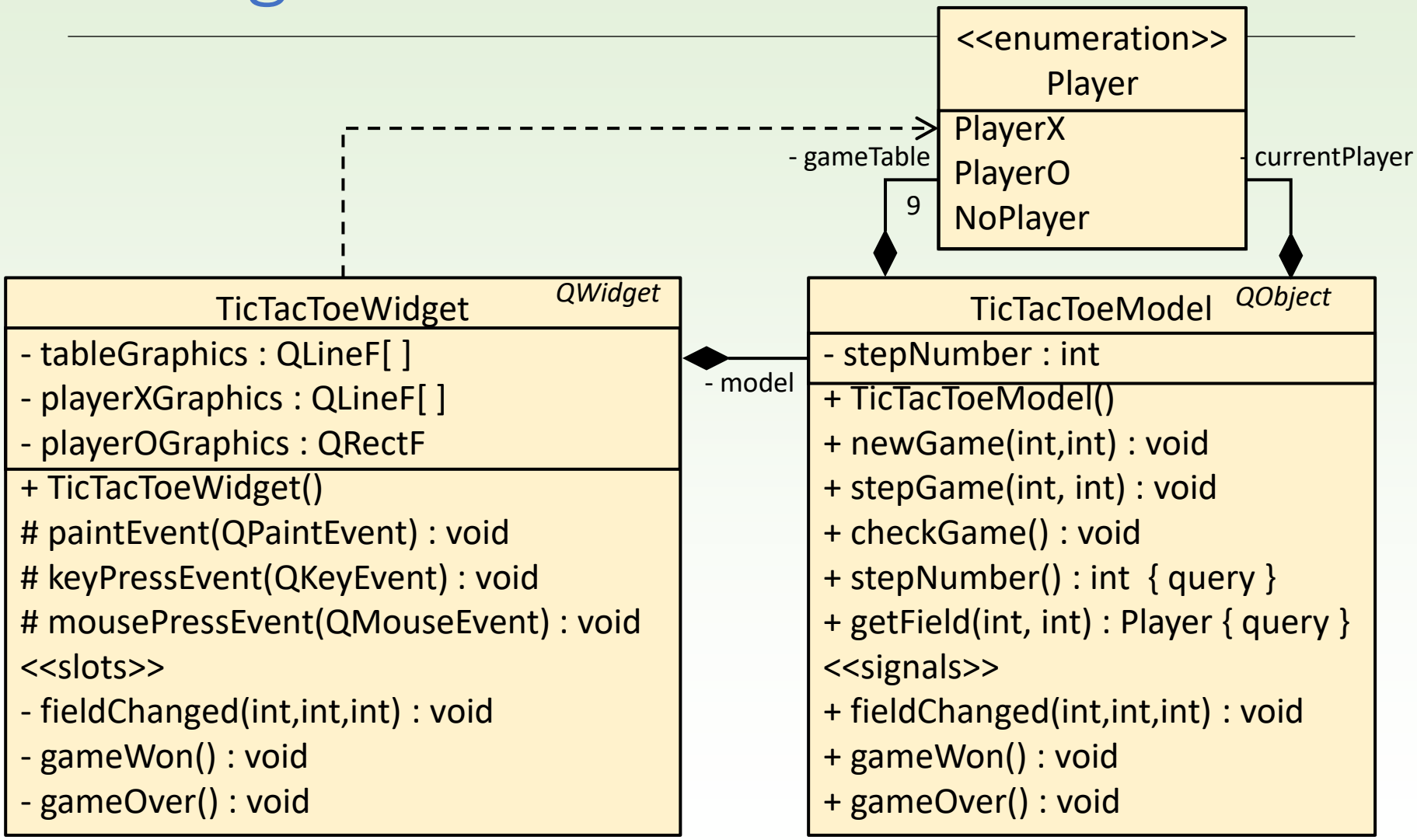
Módosítsuk a Tic-Tac-Toe program nézetét úgy, hogy a felületen alakzatok rajzaival jelenítjük meg a játékállást.

A mezőket a játékosok egér segítségével foglalhatják el.

Új játékot a **`Ctrl+N`** billentyű-kombinációval indíthatnak.



6.Megoldás: tervezés



6.Megoldás: nézet

```
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);

    _mode = new TicTacToeModel();
    …
}
```

a tábla grafikájához négy egyenes kelli

6.Megoldás: nézet

```
void TicTacToeWidget::fieldChanged()
{
    update();
}
```

nem kell paraméter

```
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.drawLines(_tableGraphics);     // tábla kirajzolása

    …
```

6.Megoldás: nézet

```
    …
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            painter.save();
            painter.translate(i * 200.0 / 3 , j * 200.0 / 3);
            switch (_model->getField(i, j)){
                case TicTacToeModel::PlayerX:
                    painter.setPen(QPen(Qt::blue, 4));
                    painter.drawLines(_playerXGraphics);
                    break;
                case TicTacToeModel::PlayerO:
                    painter.setPen(QPen(Qt::red, 4));
                    painter.drawEllipse(_playerOGraphics);
                    break;
                case TicTacToeModel::NoPlayer: break;
            }
            painter.restore();
        }
    }
}
```

6.Megoldás: nézet

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

lekezeli a Ctrl+N kombinációt

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