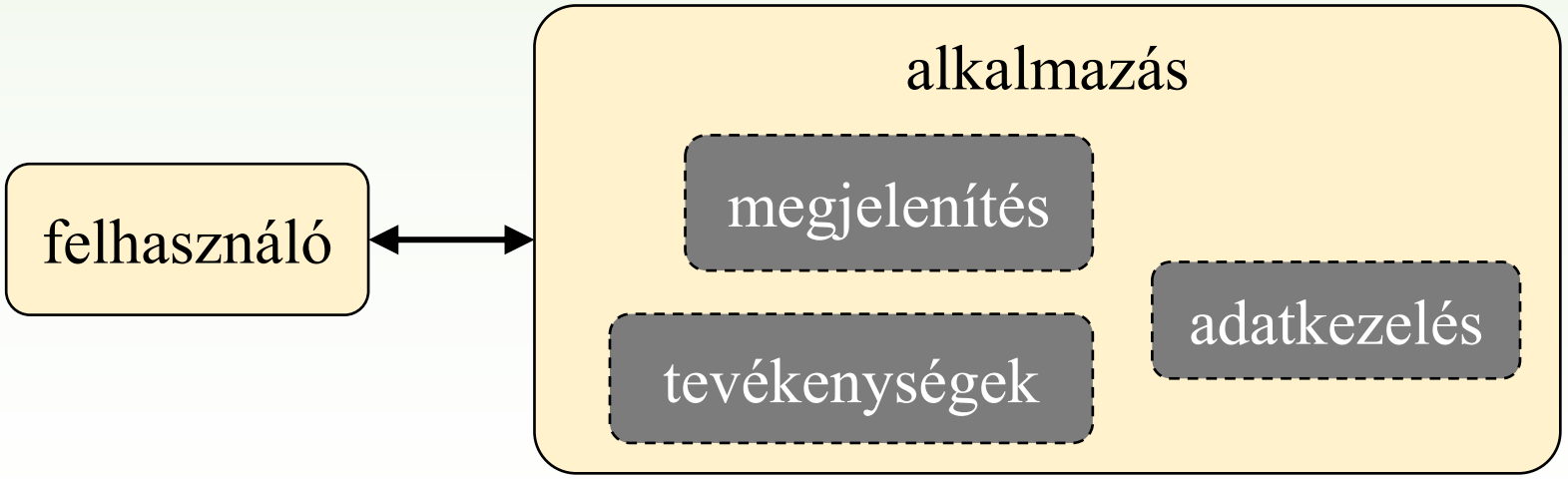
Monolitikus architektúra

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

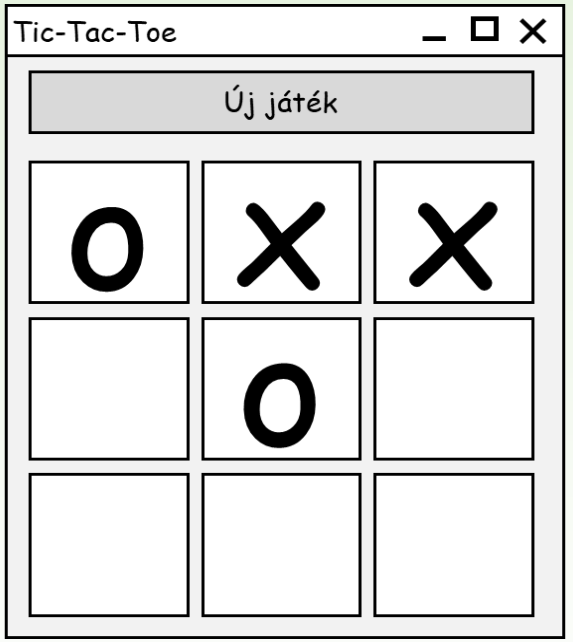
- A legegyszerűbb felépítéssel a monolitikus architektúra (*monolithic architecture*) rendelkezik, amely nem különíti el egymástól az egyes feladatköröket (pl. megjelenítés, adatkezelés).



Feladat

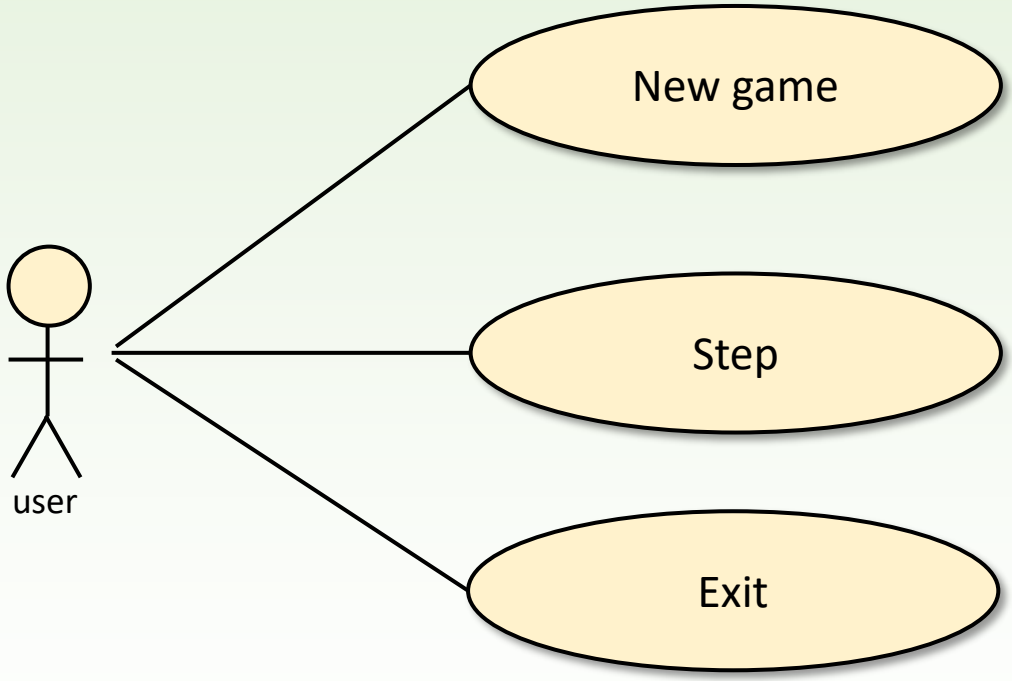
Készítsünk Tic-Tac-Toe játékot, amelyben két játékos küzdhet egymás ellen.

- A két játékos (akiket az ‚X' és ‚0' jelekkel ábrázolunk) felváltva tett lépéseire.
- A program előugró üzenetben jelez, ha vége a játéknak, majd új játékot kezd.
- A felhasználók bármikor indíthatnak új játékot.

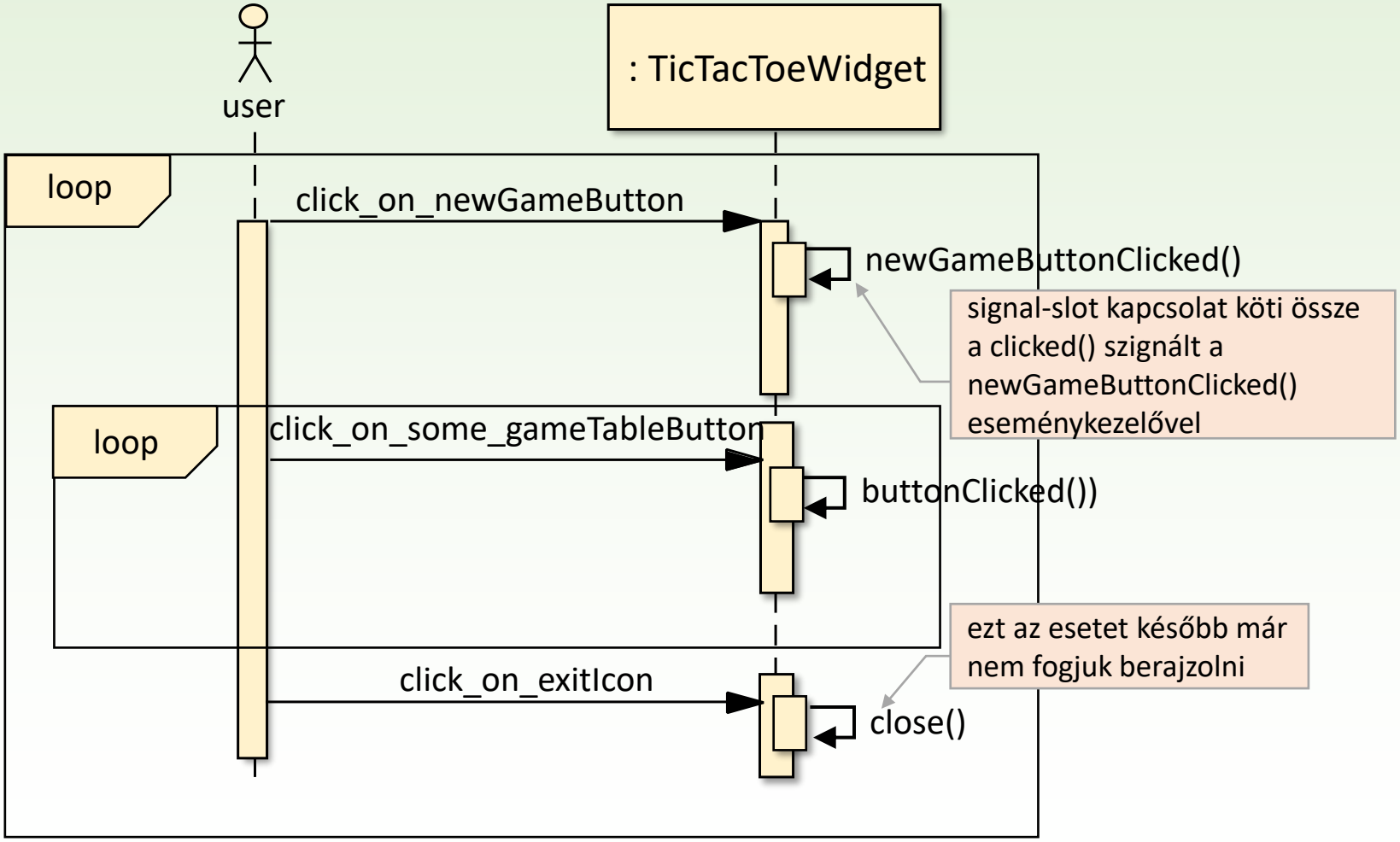


- Az alkalmazás felületét nyomógombok segítségével valósítjuk meg (Kilenc játékgomb, valamint az új játék kezdésére szolgáló).

1.Feladat: elemzés



1.Feladat: elemzés

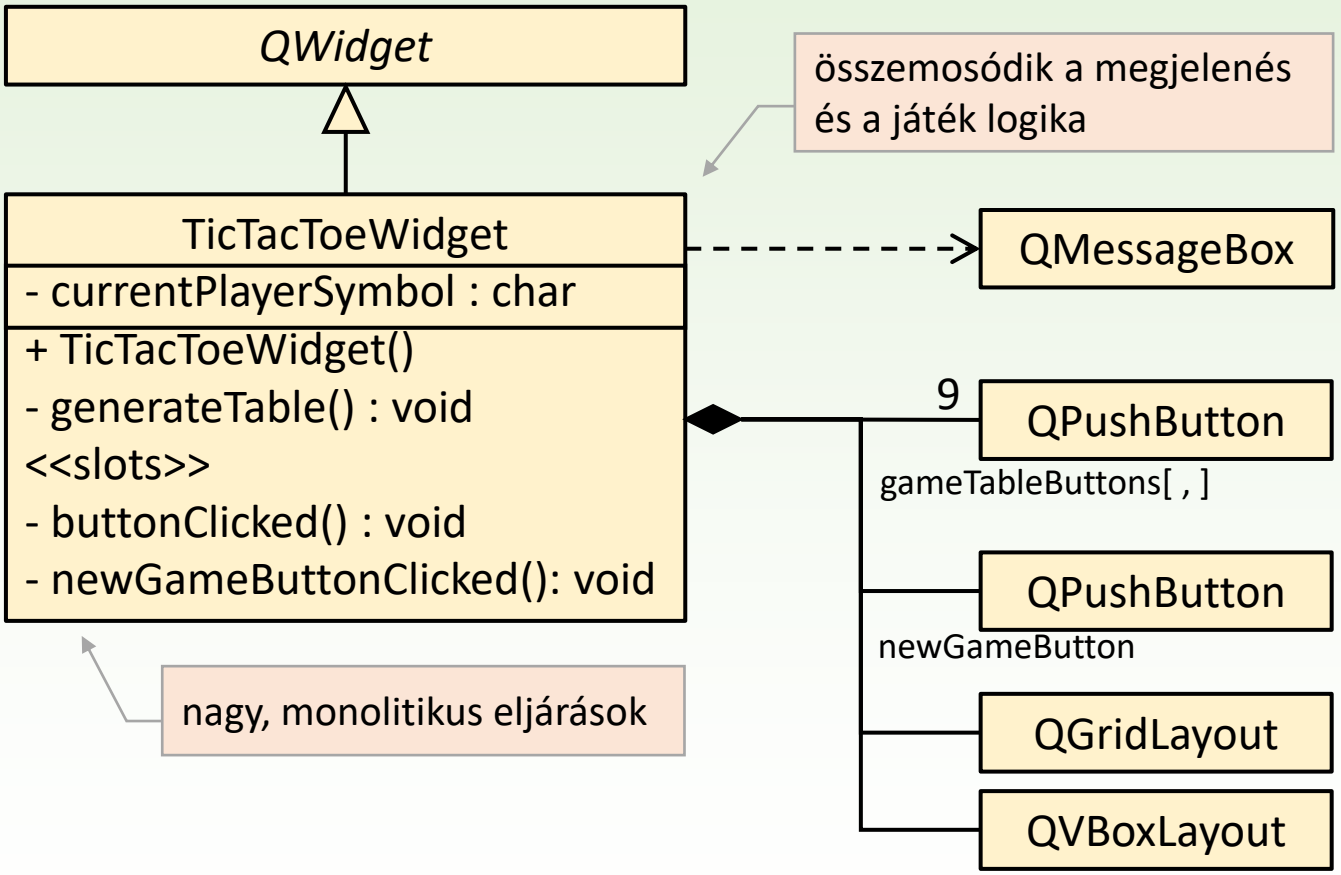


1.Megoldás: tervezés

- Az alkalmazást egyetlen osztályban (**`TicTacToeWidget`**) valósítjuk meg, amely leírja a grafikus felületet és a játék viselkedését.
- A felületen elrendezőket használva elhelyezzük az „új játék” gombját (**`_newGameButton`**), a játéktábla gombjait (**`_gameTableButtons`**),
- Egy karakterként (**`_currentPlayerSymbol`**) tároljuk el az aktuális játékos jelét.
- A felületet a konstruktor, illetve az általa meghívott **`generateTable()`** segédmetódus állítja elő.
- A játékot az eseménykezelők vezérlik.

sender	signal	reciever	slot
buttonTable[i,j]	*clicked()*	*nézet*	*buttonClicked()*
newGameButton	*clicked()*	*nézet*	*newGameButtonClicked()*

1.Megoldás: tervezés



1.Megoldás: 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
}
```

1.Megoldás: 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()));
        }
    }
}
```

1.Megoldás: 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
}
```

1.Megoldás: buttonClicked() 1.

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

    int location = _tableLayout->indexOf(senderButton);
    int x = location / 3;
    int y = location % 3;

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

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

    …
```

a gomb rácson belüli pozíciója megadja a koordinátákat

megjelenítés a gombon

váltjuk a játékost

1.Megoldás: 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.Megoldás: 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.Megoldás: 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();
        }
    }
}
```

eredmény hirdetés

Monolitikus architektúra korlátjai

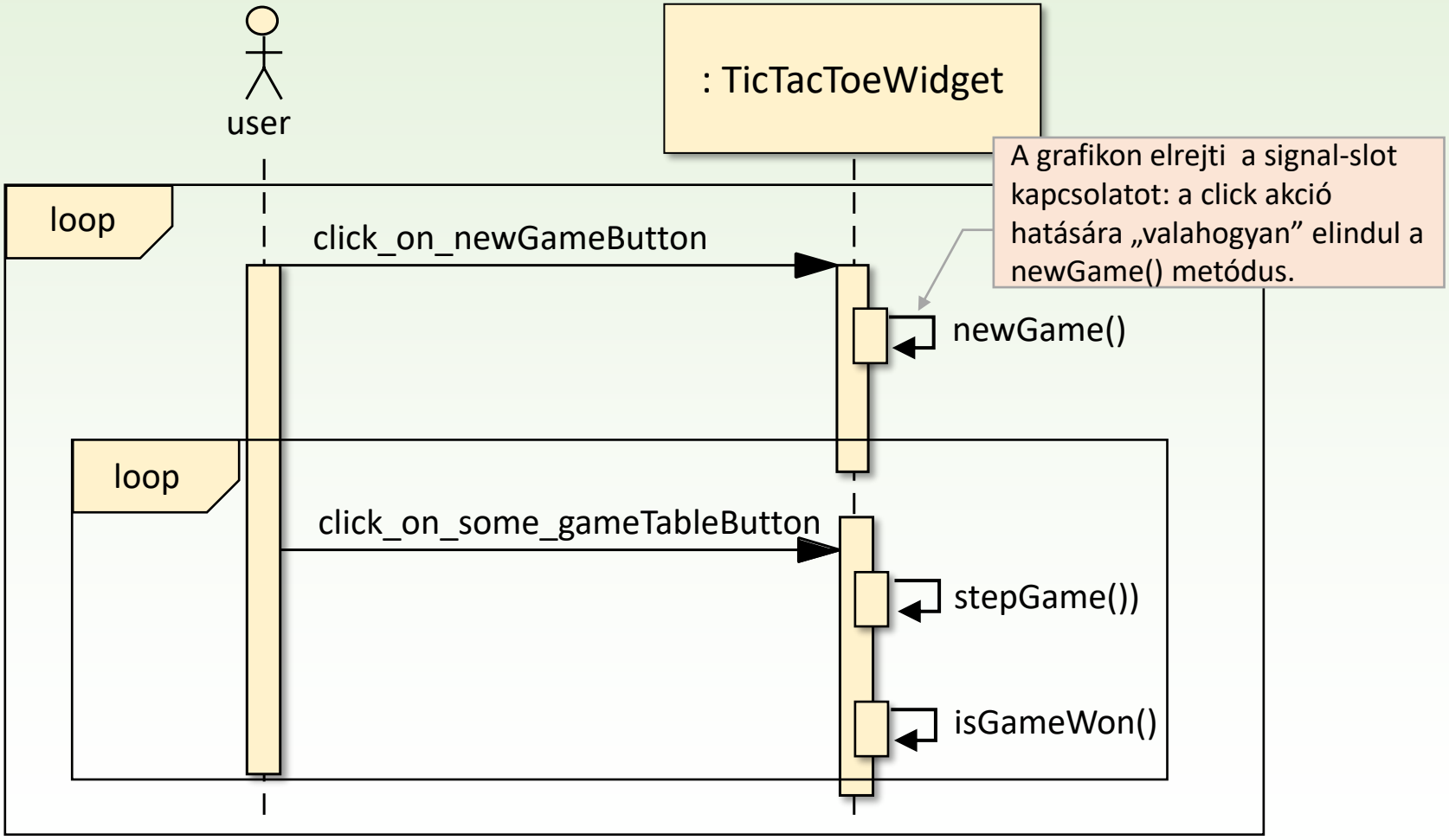
- Összetettebb alkalmazásoknál a monolitikus felépítés korlátozza a program
  - áttekinthetőségét (nehezen lelhetők fel a számítás adatai)
  - tesztelhetőségét (nem ellenőrizhetők külön-külön az egyes funkciók)
  - módosíthatóságát, bővíthetőségét (a felület kinézetét csak a működés átírásával együtt 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.Megoldás

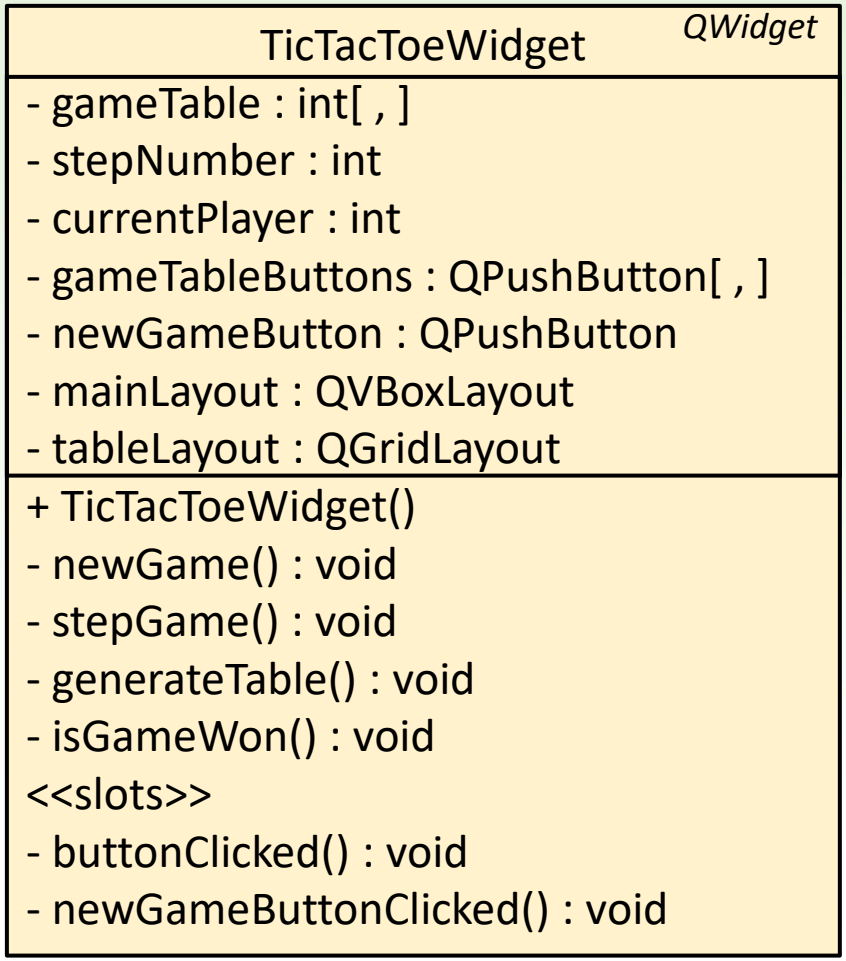
Módosítsuk a Tic-Tac-Toe programot úgy, hogy áttekinthetőbb és tagoltabb legyen. Az adattagokat és metódusokat két csoportba rendezzük aszerint, hogy azok a megjelenésért (nézet) vagy a működésért (modell) felelnek

- 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
- új metódusokat veszünk fel a játék kezelésére (**`newGame`**, **`stepGame`**, **`isGameWon`**), és ezeket fogják a felület eseménykezelői hívni

2.Megoldás: elemzés



2.Megoldás: tervezés



2.Megoldás: 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, de a nézet és a modell még együtt van

2.Megoldás: megvalósítás

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

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;

}
```

szakaszokra bontott tevékenység

játékosváltás kényelmesebb