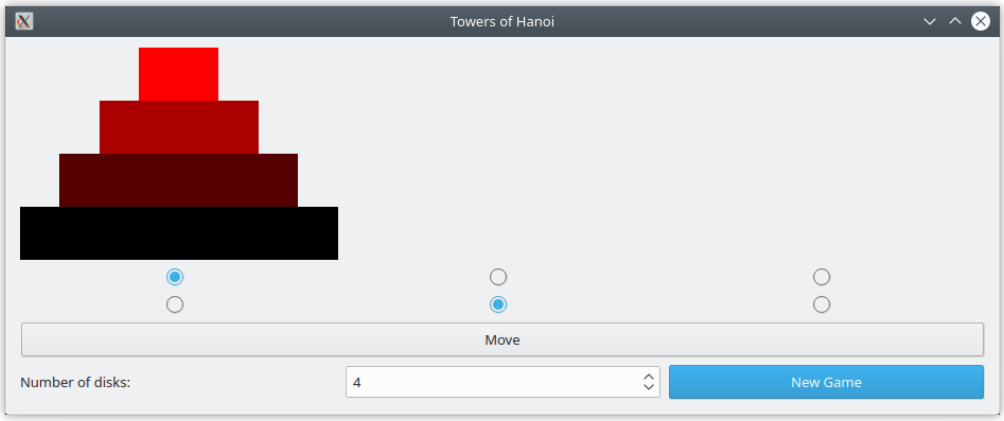
A harmadik gyakorlaton a Hanoi tornyai játékot valósítjuk meg kétrétegű (modell-nézet) architektúrában.

Első megoldás - rádiógombok

- A felhasználónak legyen lehetősége új játékot indítani a „New game” gomb megnyomásával. Új játék kezdetekor a kiválasztott számú korong jelenjen meg az első tornyon, a többi torony legyen üres.
- A „Move” gomb megnyomásakor a forrásként kiválasztott toronyról a legfelső korongot helyezzük át a cél toronyra. A forrás és cél torony kiválasztása az alattuk lévő rádiógombokkal lehetséges. A „Move” gomb csak olyankor nyomható meg, amikor a kiválasztott lépés szabályos.
- A játék érjen véget, ha minden korong az utolsó tornyon van.



Modell

A játék modellje a `GameManager` osztályból áll.

Az osztály egy példánya tárolja a játékban lévő korongok számát és az egyes tornyokon lévő korongokat. Egy tornyot egy `vector<int>` reprezentál, a vektor elemei a tornyon lévő korongok legnagyobbtól a legkisebbig.

A tornyok azonosítására a továbbiakban használjuk a 0..2 (jobbról balra), a korongok azonosítására az 1..n (legkisebbtől legnagyobbig) számokat.

A modell két eseményt definiál. Az `Update` esemény jelzi a nézet számára, hogy egy korong másik toronyra került, ezért szükséges a nézet frissítése. A `GameOver` esemény a játék végét jelzi.

A modell adjon lehetőséget a korongok számának és egy adott tornyon lévő korongok lekérdezésére.

A `StartGame(int diskCount)` eljárással a modell kezdjen új játékot. Új játék indításakor minden korong az első oszlopon van, a többi torony üres. Az eljárás részeként a modell váltsa ki az `Update` eseményt is, hogy a nézet megfelelően frissüljön.

A `CanMoveDisk(sourceTowerIndex, targetTowerIndex)` és `MoveDisk(sourceTowerIndex, targetTowerIndex)` függvények valósítják meg a játéklogikát. A `CanMoveDisk` függvény eldönti, hogy a játék aktuális állapota alapján szabályos-e a legfelső korongot áthelyezni a forrás toronyról a cél toronyra. A mozgatás engedélyezett, ha a forráson van legalább egy korong és a cél üres vagy a legfelső korong mérete nagyonbb, mint a mozgatni kívánt korong.

A `MoveDisk` függvény feladata, hogy végrehajtsa a kívánt mozgatást. A legfelső korong levételéhez használjuk a `vector` osztály `pop_back` eljárását, a torony tetejére helyezéséhez pedig a `push_back` eljárást. Amennyiben a mozgatás megtörtént, a függvény váltsa ki az `Update` eseményt. Sikeres mozgatást követően ellenőrizni kell azt is, hogy a játék véget ért-e, ennek legegyszerűbb módja az utolsó tornyon lévő korongok darabszámának vizsgálata. Amennyiben a játék véget ért, váltsuk ki a `GameOver` eseményt is.

Nézet

A nézet létrehozásához használjuk a `QGridLayout` osztályt, a vezérlőket 5 sorba és 3 oszlopba helyezzük. Az első sorba `QBoxLayout` példányok kerülnek, ezek feladata a tornyon lévő korongok elrendezése lesz. Ezek létrehozásakor állítsuk be, hogy az egyes elemek alulról felfele kerüljenek bele és a `QGridLayout` cellájában alul és középen helyezkedjen el. Állítsuk be az egyes elemek közötti távolságot 0-ra, hogy a korongok közvetlenül egymáson helyezkedjenek el.

```
QBoxLayout* tower = new
   QBoxLayout(QBoxLayout::BottomToTop);
layout->addLayout(tower, 0, i, Qt::AlignBottom |
   Qt::AlignHCenter);
tower->setSpacing(0);
```

A létrehozott `QBoxLayout` objektumokat tároljuk el egy `QVector`-ban, hogy a nézet frissítésekor ezeket el tudjuk érni.

A mozgatandó korong és a cél torony kiválasztásához készítsünk rádiógombokat (`QRadioButton`)! A gombokat helyezzük el a `QGridLayout` második és harmadik soraiba. A gombok logikai csoportosításához rendez-

zük őket két QButtonGroup-ba. A csoporton belül az egyes gombok egyedi azonosítóval rendelkeznek (int-ek), ez a csoport addButton eljárásának második paramétere.

```
QRadioButton* btn = new QRadioButton();
btnGroup->addButton(btn, id);
```

Az aktuálisan kiválasztott gomb azonosítója lekérhető a checkedId függvénnyel. A felvett gombok azonosítóit állítsuk be a hozzájuk tartotó tornyok azonosítójának megfelelően. A csoportok létrehozásával a keretrendszer automatikusan biztosítja, hogy egy csoporton belül nem lehet egyszerre több gomb kiválasztva. A program indulásakor állítsunk be mindkét csoportban egy alapértelmezetten kiválasztott rádiógombot, erre használjuk a setChecked eljárást.

A rádiógombok alatt helyezzük el a lépést elvégző „Move” gombot. Az utolsó sorban helyzzük el az új játék kezdéséhez és beállításaihoz szükséges vezérlőket. Helyezzünk le egy QSpinBox-ot a korongok számának beállításához és egy QPushButton-t az új játék elindításához.

DiskWidget

A korongok megjelenítéséhez készítsünk egy új osztályt DiskWidget néven, mely a QWidget leszármazottja. Az osztály statikus adattagjaként vegyük fel egy korong maximális szélességét (pixelben) és magasságát. Az osztály konstruktorában kapja meg a játékban lévő lemezek számát (ez a modelltől lekérhető) és a korong méretét $(1..n)$.

```
class DiskWidget : public QWidget {
 Q_OBJECT
public:
 DiskWidget(int diskSize, int totalDiskCount);

 static const int MaxDiskWidth = 300;
 static const int DiskHeight = 50;
};
```

Az osztály konstruktorában állítsuk be a méretét rögzítettre a setFixedSize eljárás segítségével. A korong szélessége az azonosító és a korongok összes számának hányadosával számítható ki. A korong megjelenítéséhez állítsuk be a háttérszínét is. Ha szeretnénk, a korong szélességéhez hasonlóan a háttérszínt is meghatározhatjuk a korong mérete alapján.

```
DiskWidget::DiskWidget(int diskSize, int totalDiskCount) :
   QWidget(nullptr) {
```

```
  this->setFixedSize(MaxDiskWidth * diskSize /
     totalDiskCount, DiskHeight);
  QPalette pal = this->palette();
  pal.setColor(QPalette::Background, QColor(255, 0, 0));
  this->setAutoFillBackground(true);
  this->setPalette(pal);
  this->show();
}
```

A játékablaknak állítsunk be egy minimális szélességet a `MaxDiskWidth` függvényében. Ehhez az ablak konstruktorában vegyük fel az alábbi utasítást. A minimális szélesség meghatározásakor hagyunk egy kis plusz helyet a tornyok közötti margónak.

```
setMinimumWidth((DiskWidget::MaxDiskWidth + 20) * 3);
```

Frissítés

Készítsünk a `GameWindow` osztályban egy eseménykezelőt, mely elvégzi a nézet frissítését. Ezt kössük a `GameManager Update` eseményéhez.

A nézet frissítésekor az egyszerűbb megvalósítás kedvéért az összes korongot töröljük, majd a modell állapota alapján újból előállítjuk. Lehetőség lenne arra is, hogy a modell az eseményben megadja, hogy melyik toronyról melyik toronyra történt mozgatás, ilyenkor megoldható lenne csak egy korong áthelyezése is.

A nézet frissítését két lépésben hajtjuk végre, először töröljük a torony tartalmát (a `tower` változó az éppen feldolgozott `QBoxLayout` példány)

```
while (tower->count() != 0) {
  QLayoutItem* disk = tower->takeAt(0);
  delete disk->widget();
  delete disk;
}
```

Ezt követően pedig egy ciklusban létrehozzuk a korongokat (`diskSize` a modelltől lekért méreteket veszi fel)

```
DiskWidget* disk = new DiskWidget(diskSize,
   totalDiskCount, tower);
tower->addWidget(disk, 1, Qt::AlignHCenter);
```

Új játék indítása

Új játék indításához készítsük el a „New game” gomb eseménykezelőjét. Az eseménykezelő hívja meg a `gameManager StartGame` eljárását. A játék

mérete lekérhető a `QSpinBox` példánytól a `value` függvény segítségével.

Ha mindent jól csináltunk, el tudunk indítani egy tetszőleges méretű új játékot és a képernyőn megjelennek az első tornyon a korongok.

Korongok mozgatása

A korongok mozgatásához első lépésben csináljuk meg, hogy a „Move" gomb csak olyankor legyen aktív (azaz megnyomható), amikor a kiválasztott mozgatás szabályos. Készítsünk egy eseménykezelőt, mely elvégzi ezt a frissítést és csatlakoztassuk az összes `QRadioButton clicked` eseményéhez, valamint a `gameManager Update` eseményéhez. A gomb letiltásához és engedélyezéséhez a `setEnabled` eljárás használható. Annak megállapításához, hogy szabályos-e a kiválasztott mozgás használjuk a `gameManager CanMoveDisk` függvényét. A kiválasztott rádiógombokat a két `QButtonGroup`-tól a `checkedId` függvénnyel kaphatjuk meg.

Hozzunk létre egy újabb eseménykezelőt, mely feladata a mozgatás végrehajtása lesz, ezt kapcsoljuk a „Move" gomb `clicked` eseményéhez. A kiválasztott tornyokat az előző függvényhez hasonlóan kérjük le, majd hívjuk meg a `gameManager MoveDisk` eljárását.

Játék vége

Készítsünk egy eseménykezelőt, melyet a `gameManager GameOver` eseményéhez kötünk. Az eseménykezelőben egy `QMessageBox` használatával jelezzük a játékosnak, hogy nyert.

Második megoldás - Drag and Drop

A feladat következő részében átalakítjuk a programot, hogy rádiógombok helyett Drag and Drop használatával mozgathassuk a tornyok között a korongokat.

TowerWidget

Hozzunk létre egy új osztályt `TowerWidget` néven, mely a `QWidget` leszármazottja. Az osztály a konstruktorban kapja meg és tárolja el az azonosítóját és egy pointert a `GameManager` példányra. Az azonosító lekérésére készítsünk egy publikus függvényt.

Az osztály leváltja az eddig toronyként használt `QBoxLayout`-okat, így azok létrehozásának helyét frissítsük. A `TowerWidget` tartalmazni fog egy `QBoxLayout`-ot, melyet a konstruktorban létre is hozunk. A `setAcceptDrops`

eljárás meghívásával jelezzük, hogy a tornyok Drag and Drop célként tudnak működni.

```
TowerWidget::TowerWidget(GameManager* gm, int id, QWidget*
    parent)
    : QWidget(parent), gameManager(gm), id(id) {
  layout = new QBoxLayout(QBoxLayout::BottomToTop, this);
  layout->setSpacing(0);
  setAcceptDrops(true);
}
```

A tornyon lévő korongok frissítését mozgassuk át ebbe az osztályba, a játékablakon lévő eseménykezelő helyett a `gameManager Update` eseményét kapcsoljuk közvetlenül ezekhez. A frissítést elvégző függvényen végezzünk el néhány módosítást.

Cseréljük le a `delete disk->widget();` utasítást az alábbiakra:

```
if (disk->widget() != nullptr) {
  disk->widget()->deleteLater();
}
```

A `deleteLater` hívásával egy késleltetett `delete` hívást valósítunk meg. Ez a késleltetés szükséges, mert a frissítést a drop váltja ki, melynek szülője az egyik korong. Ha itt azonnal törölnénk a korongokat, akkor a program megbízhatatlanul működne, bizonyos esetekben a program összeomolhatna.

A `DiskWidget` létrehozásánál a konstruktort egészítsük ki egy paraméterrel, mely a `QWidget* parent` (természetesen ennek megfelelően frissítsük a konstruktor kódját is), itt adjuk át az aktuális tornyot. A tartalmazott `QBoxLayout`-hoz adjuk hozzá a létrehozott korongot. A `DiskWidget setEnabled` eljárásának segítségével mindegyik torony esetében csak a legfelső (utolsóként létrehozott) korong legyen aktív. Ennek a beállításnak köszönhetően csak a tornyok legfelső korongjára kattinthat a felhasználó.

Mivel a `QBoxLayout` kitölti az ablakon található `QGridLayout` által adott helyet, ezért a korongok feletti helyet töltsük ki!

```
for (unsigned int i = 0; i < disks.size(); ++i) {
  DiskWidget* disk =
      new DiskWidget(disks[i], gameManager->GetDiskCount(),
          this);
  layout->addWidget(disk, 1, Qt::AlignHCenter |
      Qt::AlignBottom);
  disk->setEnabled(i == disks.size() - 1);
}
```

```
layout->addStretch(gameManager->GetDiskCount() -
    disks.size());
```

DiskWidget

A `DiskWidget` osztályban írjuk felül a megörökölt `mousePressEvent` eseményt az alábbi függvény felvételével:

```
virtual void mousePressEvent(QMouseEvent* event) override;
```

Az esemény bekövetkezésekor hozzunk létre egy új `QDrag` objektumot! A `QDrag` osztály segítségével tudjuk megvalósítani a Drag and Drop funkcionalitást.

Az eseményhez tartozó adatok ezen kívül, egy `QMimeData` osztályban helyezkednek el, melyet megadhatunk a `QDrag` objektumnak. Az esemény részeként tároljuk el a korong szülőjét, ami az egyik torony. Az esemény kezelésekor ehhez az eltárolt információhoz hozzáférünk.

```
QDrag* drag = new QDrag(parentWidget());
QMimeData* mimeData = new QMimeData();

mimeData->setParent(parentWidget());
drag->setMimeData(mimeData);

drag->exec();
```

TowerWidget - Drag and Drop kezelése

A `TowerWidget` osztályt egészítsük ki az alábbi függvényekkel:

```
virtual void dragEnterEvent(QDragEnterEvent* event)
    override;
virtual void dropEvent(QDropEvent* event) override;
```

A `dragEnterEvent` akkor fut le, amikor a vezérlő fölé ér az egér egy Drag and Drop közben, míg a `dropEvent` akkor, amikor a felhasználó a vezélrő felett elengedi korongot.

A `dragEnterEvent`-ben a vezérlőnek el kell döntenie, hogy tudja-e kezelni, elfogadja-e az eseményt. A függvényben azt ellenőrizzük, hogy az eseményhez beállított `QMimeData parent` mezője ki van-e töltve és az egy `TowerWidget`-e. Amennyiben igen, úgy elfogadjuk az eseményt az `acceptProposedAction` meghívásával. Az eseménykezelő teljes kódja:

```
const TowerWidget* parentTower =
    qobject_cast<TowerWidget*>(event->mimeData()->parent());
```

```
if (parentTower != nullptr) {
  event->acceptProposedAction();
}
```

A `dropEvent`-ben már biztosak lehetünk benne, hogy az eseményt tudjuk kezelni, így itt már csak azt kell ellenőrizni, hogy a lépés szabályos-e. A forrás torony azonosítóját kérjük le a `TowerWidget`-től, melyet az előző függvényben látható módon érhetünk el.

Szabályos lépés esetén hívjuk meg a `MoveDisk` metódust és fogadjuk el az eseményt az `event accept` eljárásával. Amennyiben nem szabályos a lépés, hívjuk meg az `ignore` függvényt.

Játékablak

A játékablakról töröljük a rádiógombokat, a „Move” gombot és az ezekhez tartozó eseményeket.