Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások fejlesztése I

13. előadás

A grafikus nézet, animációk megvalósítása

Giachetta Roberto

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

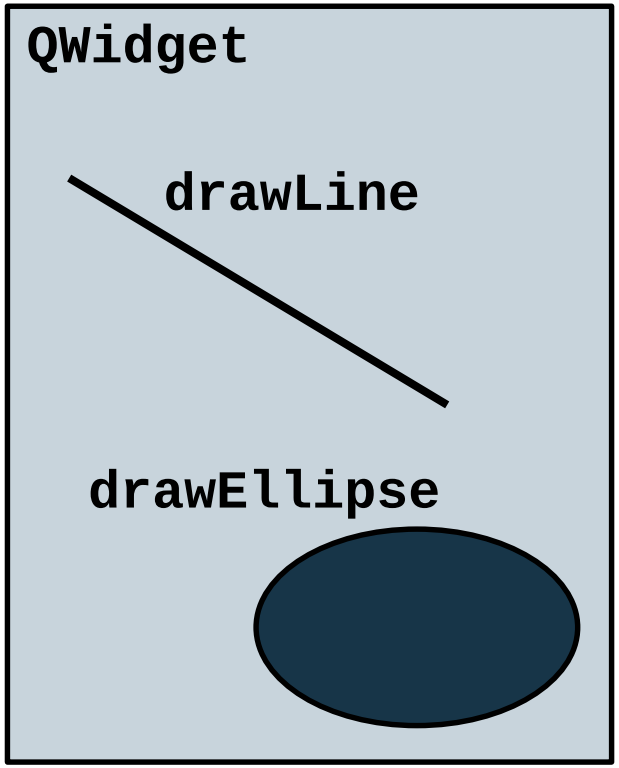
A grafikus nézet

Koncepciója

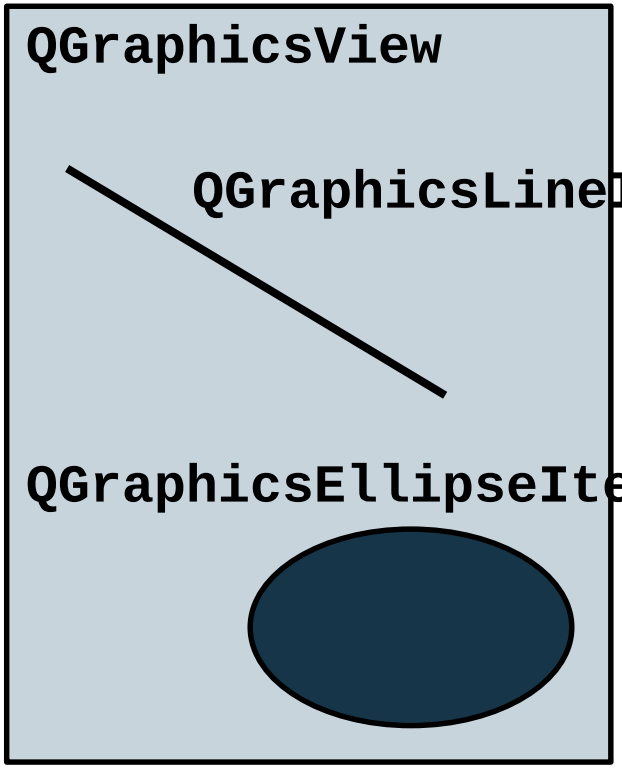
- A Qt lehetővé tesz egyedi, primitív alakzat rajzolást minden **`QWidget`** leszármazottra a **`paintEvent()`** felüldefiniálásával
  - az így létrehozott elemek kombinációjával tetszőleges bonyolultságú felületet hozhatunk létre, azonban az elemekkel nem lehet interakciót kezdeményezni
- A Qt bevezette az objektumként megjelenő grafikus elem koncepcióját, a *grafikus nézet*et (*Graphics View*), amely
  - mindenre képes, amire az alapvető rajzolás
  - lehetővé teszi az objektumként funkcionáló *grafikus elem*ek (**`QGraphicsItem`**) kezelését
  - lehetővé teszi animációk megvalósítását az elemeken

A grafikus nézet

Koncepciója



egyszerű rajzolás

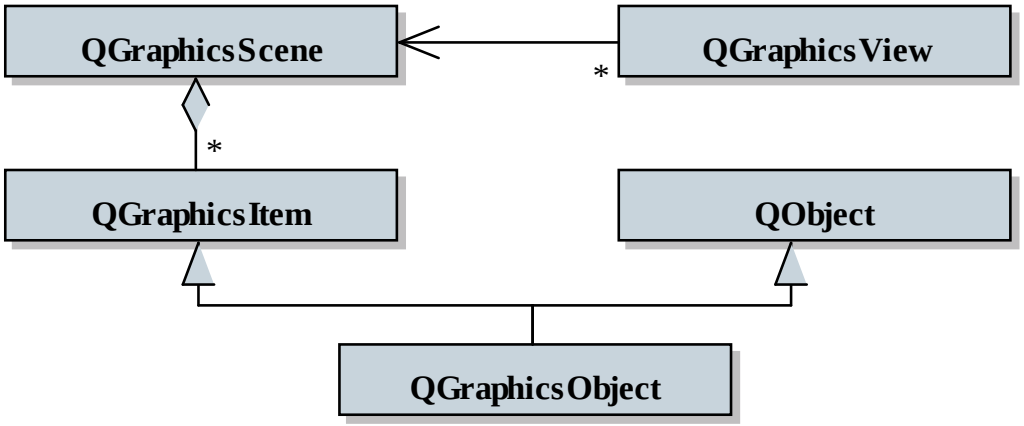


grafikus nézet

A grafikus nézet

Felépítése

- A grafikus nézet architektúrája két rétegből épül fel
  - a grafikus nézet logikai alapja a *jelenet* (**`QGraphicsScene`**)
  - a megjelenítést a *nézet* (**`QGraphicsView`**) biztosítja, a jelenetet bármennyi, különböző nézetben megjeleníthetünk
  - a jelenetbe tetszőleges sok **`QGraphicsItem`** leszármazottat helyezhetünk (pl. **`QGraphicsObject`**)



A grafikus nézet

Felépítése

- Pl.:

```
QGraphicsScene scene; // jelenet létrehozása
scene.setSceneRect(-100,-100, 300, 300);
   // koordinátarendszer beállítása
QGraphicsItem *rect =
   new QGraphicsRectItem(QRectF(0,0,100,100));
   // új téglalap elem létrehozása
scene.addItem(rect); // felvétel a jelenetbe
QPixmap *pix = new QPixmap("image.png");
scene.addItem(new QGraphicsPixmapItem(pix));
   // kép felvétele a jelenetben

QGraphicsView view(scene); // nézet a jelenetre
view.show(); // nézet megjelenítése
```

A grafikus nézet

A jelenet

- A jelenet biztosítja a grafikus alakzatok kezelését
  - felvenni elemeket az **`addItem(<`*`elem`*`>)`**, vagy speciálisabb (pl. **`addRect`**, **`addPixmap`**, **`addWidget`**) műveletekkel tudunk
  - ha a jelenet változik, kiváltja a **`changed()`** eseményt
  - az elemek később az **`items()`** és **`itemAt(<`*`pozíció`*`>)`** kérésekkel kérhetőek le
- A tartalomnak külön logikai koordinátarendszere van (**`sceneRect`**), amely végtelen is lehet, és lekérhető az elemek befoglaló téglalapja is (**`itemsBoundingRect`**)
- A jelenet az elemeket hatékonyan, indexeléssel kezeli a gyors keresés végett, amely szabályozható (**`itemIndexMethod`**)

A grafikus nézet

A jelenet

- A jelenetben számos esemény kezelhető felüldefiniálással, így célszerű saját jelenet osztályokat definiálni
- Pl.:

```
class MyGraphicsScene : QGraphicsScene{
  …
protected:
   void dragMoveEvent(
      QGraphicsSceneDragDropEvent*);
      // elemmozgatás
   void mouseMoveEvent(QGraphicsSceneMouseEvent*);
   void wheelEvent(QGraphicsSceneWheelEvent*);
      // egérgörgő
   void keyPressEvent(QKeyEvent*); // billentyű
};
```

A grafikus nézet

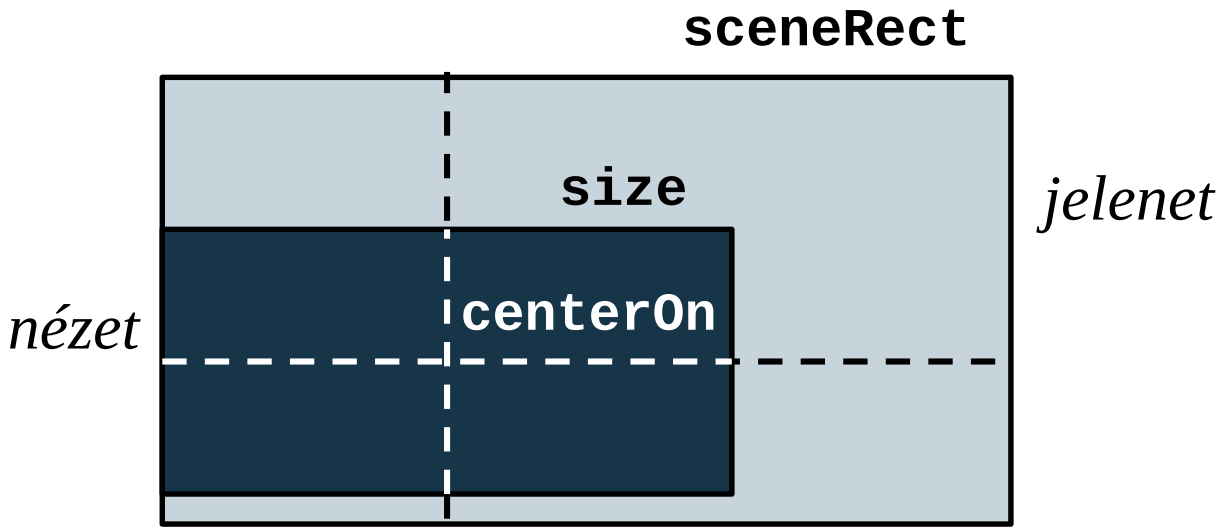
A nézet

- A nézet határozza meg a megjelenés tulajdonságait, pl.:
  - elhelyezések kezelése (**`QGraphicsLayout`**)
  - a megjelenített területet (**`size`**) és annak elhelyezkedését a jelenetben (**`centerOn(<`*`koordináta`*`>)`**)
  - jelenetkövetési mód (**`viewportAnchor`**)
  - frissítési terület (**`viewportUpdateMode`**)
  - a megjelenítő automatikusan a jelenet közepére áll, és megjeleníti a görgetősávot, amennyiben szükséges, de ez módosítható, továbbá lehet egérmozgatást is használni (**`dragMode`**)

A grafikus nézet

A nézet

- a jelenet és a nézet koordinátarendszere egymásra leképezhető (**`mapFromScene`**, **`mapToScene`**)
- lehetőséget ad a jelenet koordinátarendszerének affin transzformálására is (pl. **`rotate`**, **`translate`**)



A grafikus nézet

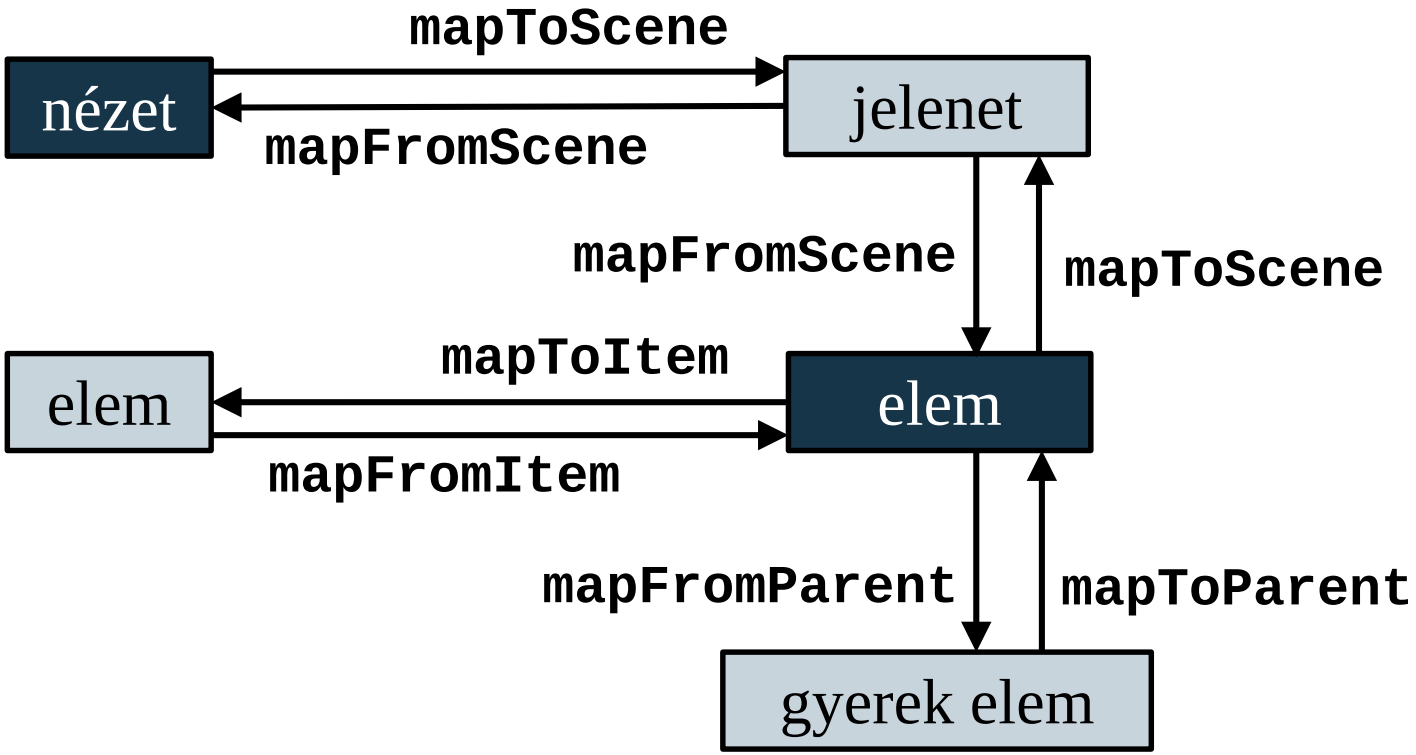
Elemek

- Az elemek számos funkcionalitást biztosítanak, pl.:
  - egyedi stílus (**`brush`**, **`pen`**), kurzorkezelés (**`cursor`**)
  - fókuszálás (**`focus()`**)
  - ütközés detektálás (**`collidesWithItem`**)
  - gyerekelemek (**`childItems`**) kezelése
  - grafikus effektek (**`graphicsEffect`**) kezelése, pl. elmosás, árnyék, áttetszőség
- Az elemek maguk 2 dimenziósak, de meghatározható sorrendjük a felületen (**`zValue`**), így átfedés esetén a magasabb értékkel rendelkező elemek kerülnek előtérbe

A grafikus nézet

Elemek és a koordinátarendszer

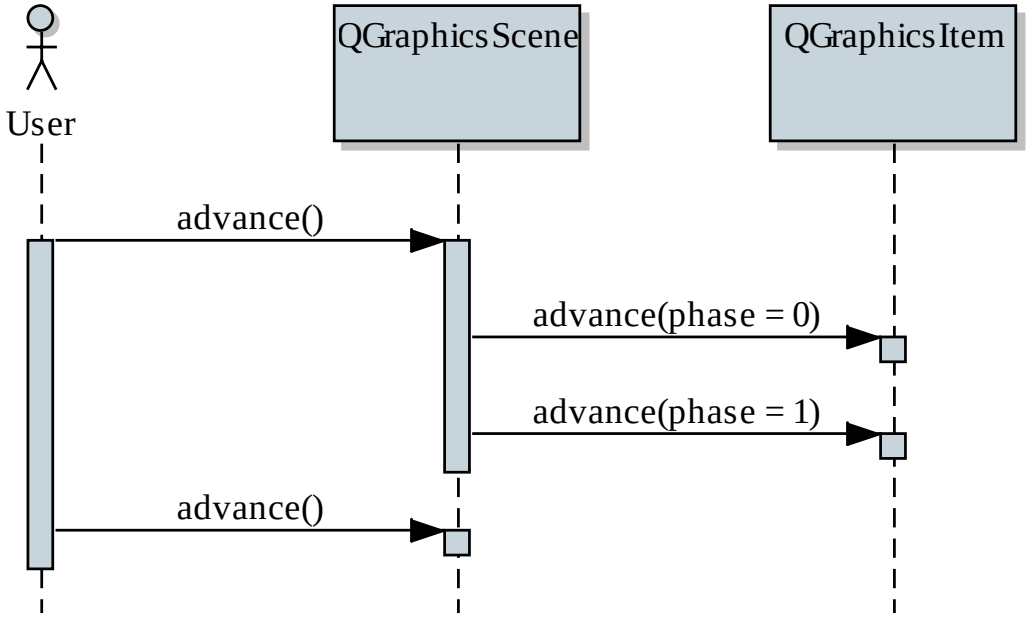
- Az elem rendelkezik saját koordinátarendszerrel, ismert a befoglaló téglalapja (**`boundingRect`**), ez leképezhető egyéb koordinátarendszerekre (**`mapFromScene`**, **`mapToScene`**, …)



A grafikus nézet

Jelenetek léptetése

- Amennyiben a jelenet tartalma nem rögzített, lehetőségünk van léptetni (**advance**) és így animációkat létrehozni
  - a jelenet léptetése a benne lévő elemek léptetése (az elem **advance** műveletével), amelynek két fázisa (**phase**) van



A grafikus nézet

Egyedi elemek

- Származtatatással az alap elemtípusok mellett tetszőleges elemet valósíthatunk meg
- Egyedi elemek esetén a leszármazott osztályban
  - felüldefiniálandó:
    - az elem kirajzolása (**`paint`**)
    - a befoglaló téglalap lekérdezése (**`boundingRect`**)
  - felüldefiniálható:
    - az alakzat lekérdezése (**`shape`**), amely pl. az ütközésdetektálásnál használatos
    - a jelenet előrehaladásának tevékenysége (**`advance(<`*`fázis`*`>)`**)

A grafikus nézet

Példa

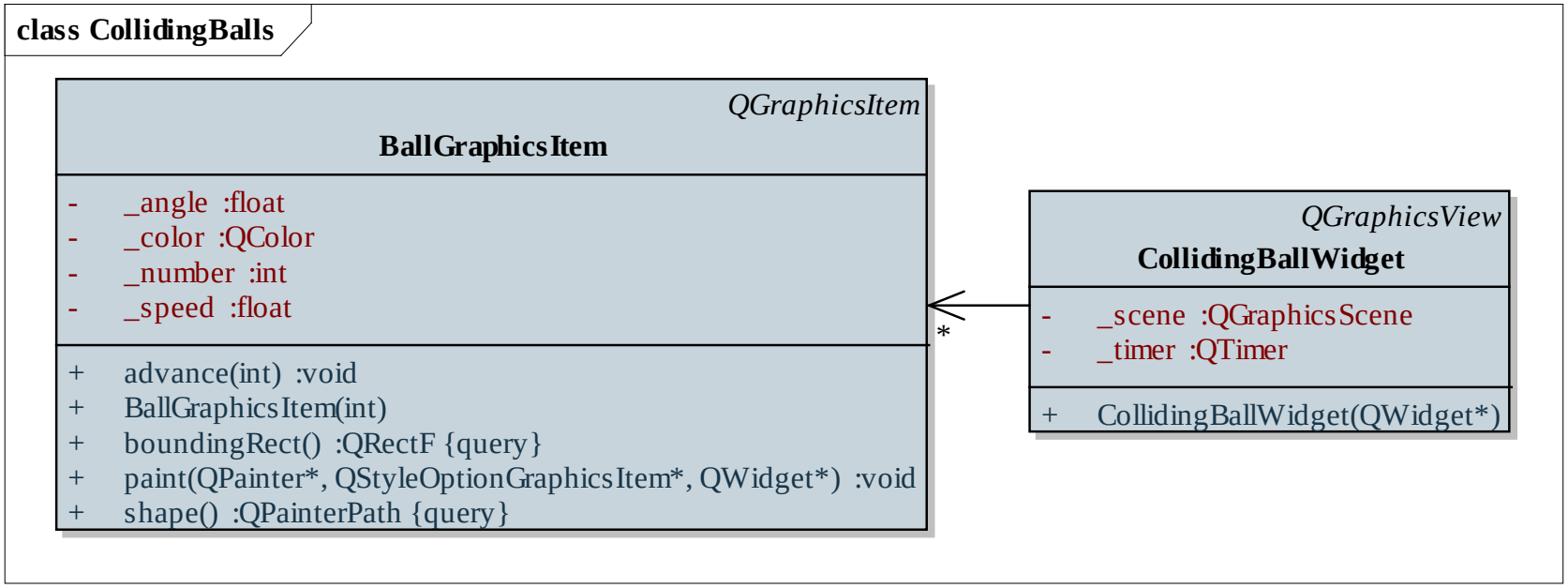
Feladat: Készítsünk egy golyómezőt, ahol különböző golyók mozognak úgy, hogy van-e egy gravitációs pont a képernyő közepén, így távolodva lassulnak, majd megfordulnak, továbbá ha egymással ütköznek, akkor elpattannak.

- az ablakunk egy grafikus nézet lesz, amelybe behelyezzük a golyókat és egy időzítővel animáljuk
- megengedjük a jelenet pozícionálását az egérrel
- szükségünk van egy egyedi elemtípusra (**`BallGraphicsItem`**), amelyben felüldefiniáljuk a megfelelő műveleteket, pl. a léptetést (**`advance`**), ahol ellenőrizzük az ütközéseket (**`collidesWith`**), illetve a haladási irányt és sebességet figyeljük

A grafikus nézet

Példa

Tervezés:



A grafikus nézet

Példa

Megvalósítás (`ballgraphicsitem.cpp`*):*

```
…
QRectF BallGraphicsItem::boundingRect() const {
    // határoló téglalap lekérdezése
    return QRectF(-20, -20, 40, 40);
}

QPainterPath BallGraphicsItem::shape() const {
    // alakzat lekérdezése (pl. ütközéshez)
    QPainterPath path;
    path.addEllipse(-20, -20, 40, 40);
    return path;
}
```

A grafikus nézet

Példa

Megvalósítás (`ballgraphicsitem.cpp`*):*

```
void BallGraphicsItem::paint(QPainter *painter,
   const QStyleOptionGraphicsItem *, QWidget *) {
   // kirajzolás
   painter->setPen(Qt::black);
      // szokványos módon rajzolunk
   painter->setBrush(_color);
   painter->drawEllipse(-20, -20, 40, 40);
   painter->setFont(QFont("Courier New", 20));
   painter->drawText(-9, 9,
      QString::number(number));
}
```

A grafikus nézet

Példa

Megvalósítás (`ballgraphicsitem.cpp`*):*

```
void BallGraphicsItem::advance(int phase) {
   // léptetés (animálás)
   if (phase == 0) { // a 0-s fázisban vagyunk
      … // meghatározzuk a haladási irányt
   }
   else { // az 1-es fázisban vagyunk
      setPos(mapToScene(cos(_angle / 180 * PI) *
            _speed, -sin(_angle / 180 * PI) *
            _speed));
      // elküldjük az új irányba
   }
}
```

A grafikus nézet

Példa

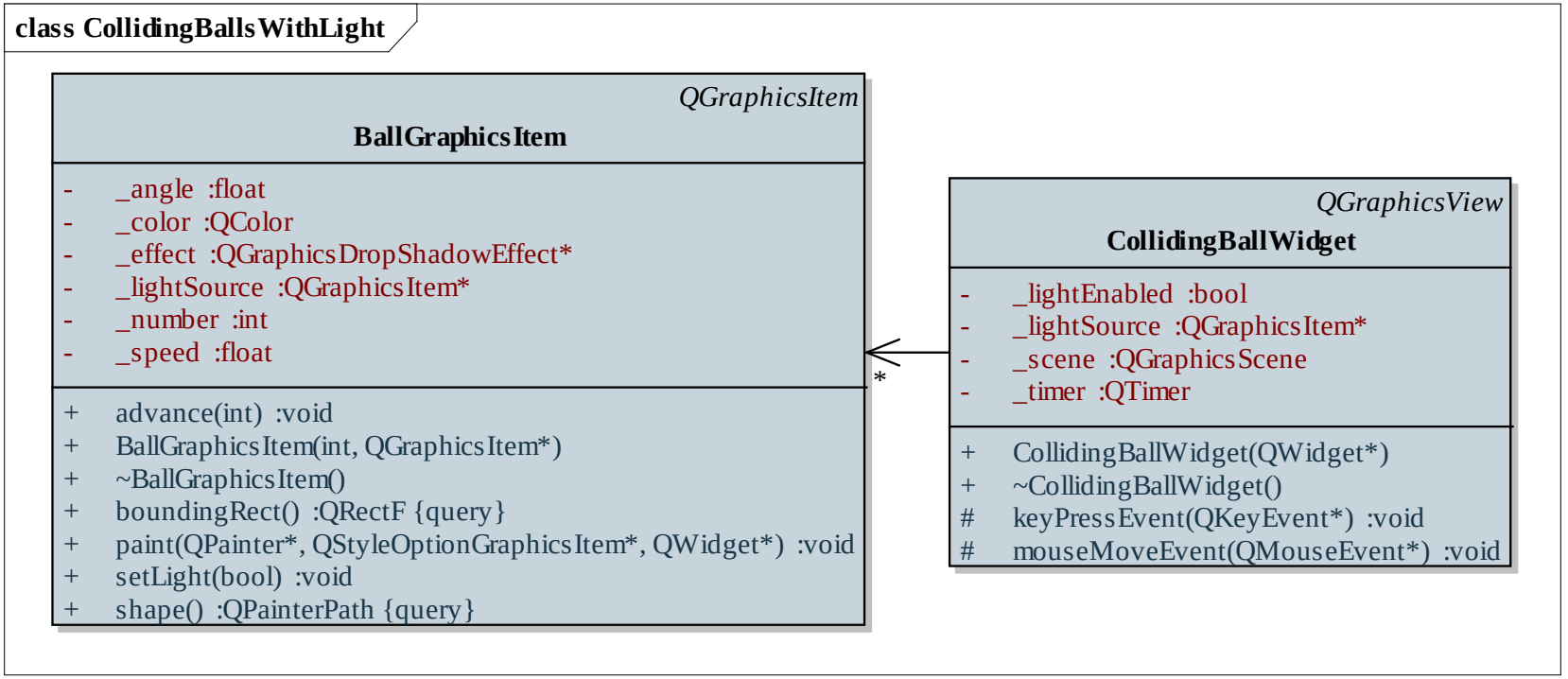
Feladat: Módosítsuk az előző példát úgy, hogy egy fényforrást is felveszünk az ablakba, amely árnyékot ad a golyóknak.

- az árnyékot effektként vesszük fel az elemhez, amely folyamatosan követni fogja azt, és a léptetésben megfelelően alakítja az árnyékot a fényforráshoz képest
- felveszünk egy fényforrást, amely egy egyedileg megrajzolt kép (**`QPixmap`**) lesz áttetszőséggel, a fényforrást az elemek elé helyezzük (**`zValue`**)
- a fényforrást a „szóköz” billentyű hatására tudjuk ki-be kapcsolni, ezért kezeljük a billentyűeseményeket (**`keyPressEvent`**), és amennyiben lenyomtuk az egeret, akkor az egérpozícióba helyezzük (**`mouseMoveEvent`**)

A grafikus nézet

Példa

Tervezés:



A grafikus nézet

Példa

Megvalósítás (`collidingballwidget.cpp`*):*

```
…
QPixmap pixmap(60, 60); // a fénypont egy kép lesz
pixmap.fill(Qt::transparent);
   // amely kezdetben áttetsző
QPainter painter(&pixmap); // erre rajzolunk
…
painter.drawEllipse(0, 0, 60, 60);
   // egy ellipszist rajzolunk
…
_lightSource = _scene.addPixmap(pixmap);
   // felvesszük a jelenethez a képet, amely
   // visszaadja a felvett grafikus objektumot
…
```

Animációk megvalósítása

Tulajdonságok animálása

- A grafikus nézet animációs megoldása igényli, hogy a fejlesztő programozza az animáció minden lépését, lehetőségünk van azonban tényleges animációk lejátszására az *animációs keretrendszer*rel (*Animation Framework*)
- Az animációt a **`QAbstractAnimation`** és leszármazott osztályok biztosítják, elsősorban a **`QPropertyAnimation`**
  - az animáció egy objektum (**`targetObject`**) megadott tulajdonságát (**`propertyName`**) tudja egy kezdőállapotból (**`startValue`**) egy végállapotba (**`endValue`**) vinni valamennyi idő alatt (**`duration`**), automatikusan
  - a célobjektumnak a **`QObject`** leszármazottnak kell lennie, hogy tulajdonságait felhasználhassuk (**`Q_PROPERTY`**)

Animációk megvalósítása

Tulajdonságok animálása

- Pl.:

```
QPropertyAnimation *anim =
    new QPropertyAnimation(); // új animáció
anim->setTargetObject(&rectangle);
    // célobjektum beállítása
anim->setPropertyName("geometry");
    // céltulajdonság
anim->setDuration(10000);
    // a hossza 10 másodperc
anim->setStartValue(QRect(0, 0, 100, 30));
    // kezdő érték
anim->setEndValue(QRect(250, 250, 100, 30));
    // befejező érték
anim->start(); // animáció végrehajtása
```

Animációk megvalósítása

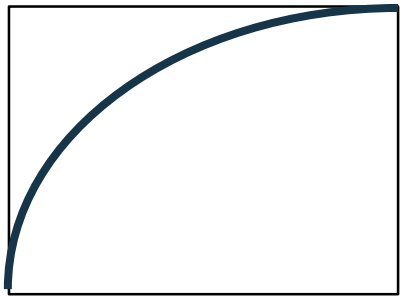
Animációk futtatása, csoportosítása

- Animációkat lehet indítani (**`start()`**), leállítani (**`stop()`**), illetve szüneteltetni (**`setPaused(<`*`igaz/hamis`*`>)`**)
- Animációk futtathatóak ciklusban tetszőleges sokszor (**`loopCount`**)
  - az alapértelmezett érték 1, -1-es érték esetén végtelen sokszor futnak, amíg le nem állítják őket
- Animációk csoportosíthatóak kétféleképpen:
  - szekvenciálisan (**`QSequentialAnimationGroup`**), ekkor egymás után kerülnek lefutásra
  - párhuzamosan (**`QParallelAnimationGroup`**), ekkor egy időben futnak le

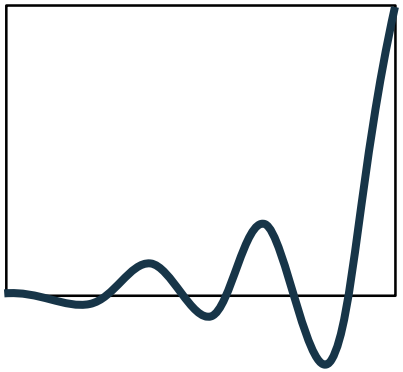
Animációk megvalósítása

Átmenetgörbék

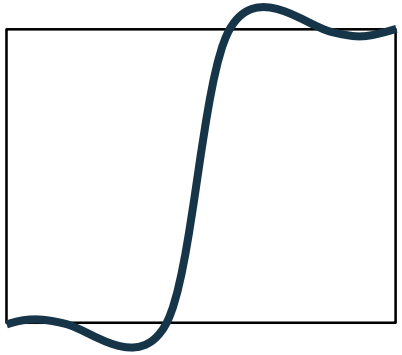
- Az animációknál az átmenet alapértelmezés szerint lineáris, ám ez módosítható átmenetgörbe (**`easingCurve`**) megadásával
  - az átmenet lehet egyedi függvény, vagy előre definiált, pl. szinusz (**`Sine`**), exponenciális (**`Expo`**), ugráló (**`Bounce`**)
  - használhatunk bemenő (**`In`**), kimenő (**`Out`**), vagy kombinált (**`InOut`**, **`OutIn`**) átmenetet



`OutCirc`



`InElastic`



`InOutBack`

Animációk megvalósítása

Kulcspontok és átmenetgörbék

- pl.:
  ```
  anim->setEasingCurve(QEasingCurve::InSine);
  ```
- egyedi átmenetek is készíthetőek (**`QEasingCurve`**)

- Az animációnak megadhatóak kulcspontok (**`keyValueAt`**), így nem a kezdőpont és a végpont között lesz lineáris az átmenet, hanem a köztes értékek között
  - a köztes pontok a [0, 1] intervallumban helyezkedhetnek el, a 0 a kezdőpont, 1 a végpont
  - pl.:
    ```
    anim->setKeyValueAt(0.3, QRect(0,0,100,100));
    anim->setKeyValueAt(0.5, QRect(10,10,100,100));
    anim->setKeyValueAt(0.7, QRect(0,0,100,100));
    ```

Animációk megvalósítása

Grafikus elemek animálása

- Az egyszerű grafikus elemek nem leszármazottai a **`QObject`**-nek, ezért nem animálhatóak, ha animálni szeretnénk őket két lehetőségünk van:
  - a **`QGraphicsObject`** osztályból származtatni, és a vizuális megjelenítést definiálni
  - a **`QObject`** és a megfelelő grafikus elem osztályból származtatni és a tulajdonságot definiálni
  - pl.:

```
class AnimableLine : QObject, QGraphicsLineItem
{ // animálható vonal
   Q_PROPERTY(QPointF pos READ pos WRITE setPos)
   …
};
```

Animációk megvalósítása

Példa

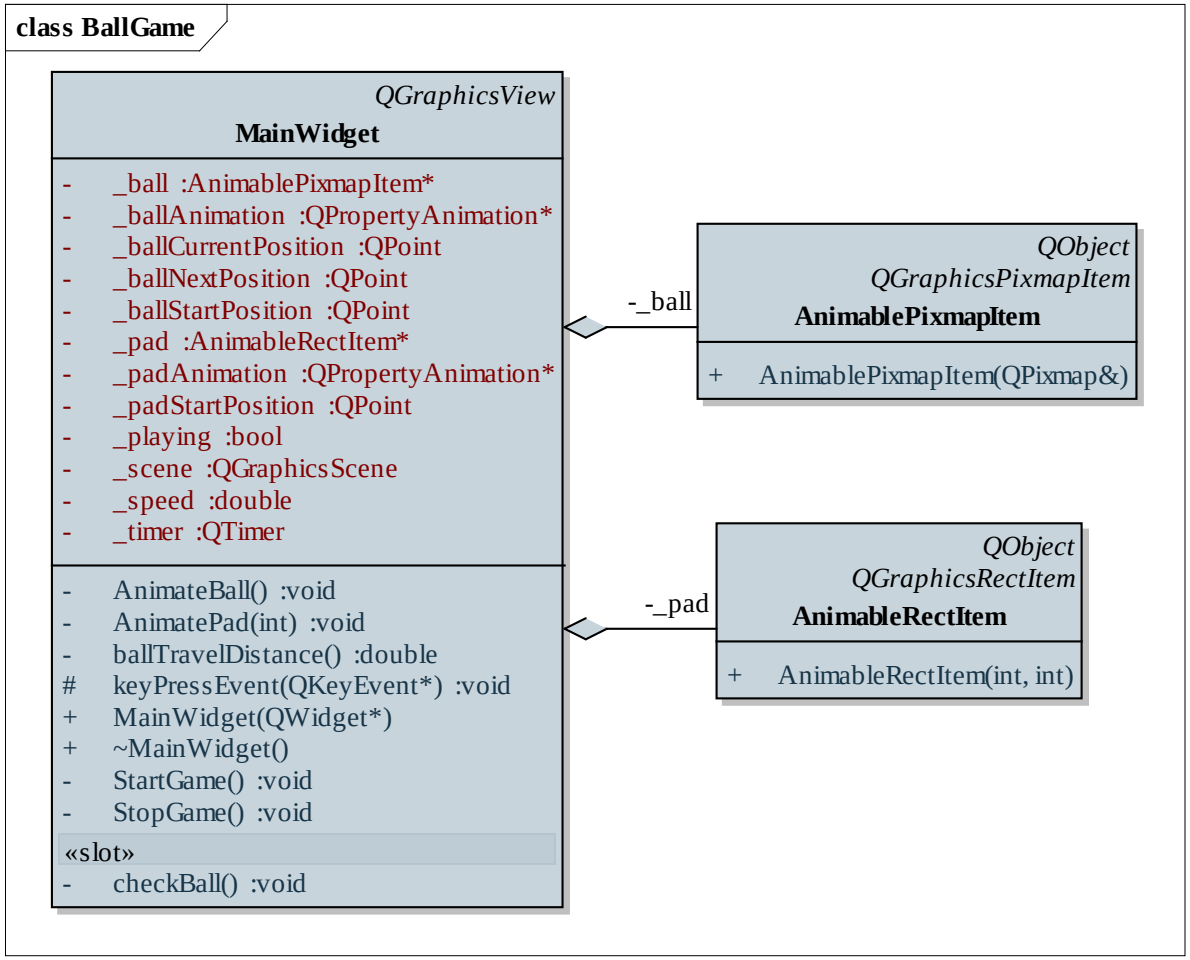
Feladat: Készítsünk egy fallabda játékot, ahol egy pattogó labdát kell egy vízszintesen mozgó ütővel visszaütni.

- a labda visszapattan minden falról, kivéve alulról, mert ha azt eléri a játékos vesztett, a labdát képpel jelenítjük meg, és mindig gyorsabb lesz, ha az ütőnek ütközik
- az ütőt billentyűvel tudjuk mozgatni balra és jobbra
- a labda és az ütő mozgatását animáció segítségével oldjuk meg, de mivel grafikus felületen vagyunk, szükséges két egyedi animálható osztály felvétele (**`AnimablePixmapItem`**, **`AnimableRectItem`**)
- az animációnál ügyelünk, hogy a sebesség ne változzon, ezért figyelembe vesszük a labda által megtett távolságot

Animációk megvalósítása

Példa

Tervezés:



Animációk megvalósítása

Példa

Megvalósítás (`mainwidget.h`*):*

```
…
// segédosztályok az animációhoz:
class AnimablePixmapItem : public QObject,
   public QGraphicsPixmapItem {
    Q_OBJECT
    Q_PROPERTY(QPointF pos READ pos WRITE setPos)
        // saját tulajdonság definiálása
public:
    AnimablePixmapItem(const QPixmap &pix) :
       QObject(), QGraphicsPixmapItem(pix) {}
};
…
```

Animációk megvalósítása

Példa

Megvalósítás (`mainwidget.cpp`*):*

```
…
ball =
    new AnimablePixmapItem(QPixmap(":/ball.png"));
    // a labdát képből töltjük be
…
_ballAnimation = new QPropertyAnimation();
_ballAnimation->setTargetObject(_ball);
    // célobjektum beállítása
_ballAnimation->setPropertyName("pos");
    // céltulajdonság beállítása
…
```

Animációk megvalósítása

Példa

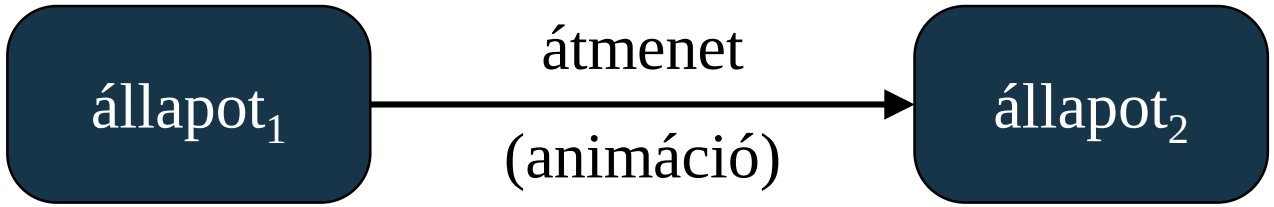
Megvalósítás (`mainwidget.cpp`*):*

```
void MainWidget::AnimateBall(){
    _ballAnimation->stop();
       // a régi animációt leállítjuk
    _ballAnimation->setDuration(5 / _speed *
       ballTravelDistance());
       // az időt a labda sebessége és a megtett
       // út alapján számoljuk
    _ballAnimation->setStartValue(_ball->pos());
       // a jelenlegi pozícióból indul
    _ballAnimation->setEndValue(_ballNextPosition);
    _ballAnimation->start();
}
…
```

Animációk megvalósítása

Állapotok kezelése

- Animációk összefoghatóak állapotkezeléssel is, amelynek segítségével állapotátmenet gráfokat definiálhatunk az alkalmazásban
  - az állapotkezelést a **`QStateMachine`** osztály biztosítja, az állapotot a **`QState`** definiálja
  - megszabhatjuk, milyen állapotok (**`addState`**) és átmenetek lehetnek (**`addTransition`**), azokhoz pedig rendelhetőek animációk (**`addAnimation`**)



Animációk megvalósítása

Állapotok kezelése

- Pl.:

```
QPushButton *button =
   new QPushButton("Animated Button");
button->show();

QStateMachine *machine = new QStateMachine;
  // állapotkezelő

QState *state1 = new QState(machine); // állapot
state1->assignProperty(button, "geometry",
   QRect(0, 0, 100, 30));
   // az állapot által kezelt tulajdonság
machine->setInitialState(state1);
   // kezdőállapot megadása
```

Animációk megvalósítása

Állapotok kezelése

```
QState *state2 = new QState(machine);
   // második állapot
state2->assignProperty(button, "geometry",
   QRect(250, 250, 100, 30));

QSignalTransition *transition1 =
   state1->addTransition(button,
                SIGNAL(clicked()), state2);
   // állapotátmenet adott eseményre
transition1->addAnimation(
   new QPropertyAnimation(button, "geometry"));
   // animáció az átmenethez
```

Animációk megvalósítása

Állapotok kezelése

```
QSignalTransition *transition2 =
   state2->addTransition(button,
      SIGNAL(clicked()), state1);
transition2->addAnimation(
   new QPropertyAnimation(button, "geometry"));

machine->start(); // állapotkezelés indítása
```