Elemi grafika

Rajzoló objektum, egér- és billentyű követés

Rajzolási felület

- A Qt-ban a grafikus felhasználói felületnek egyedi megjelenítést adhatunk a tartalmának „megrajzolásával”.
  - Mindenre rajzolhatunk, ami a **`QPaintDevice`** leszármazottja, így az összes grafikus vezérlőre (**`QWidget`**), képre (**`QPixmap`**) , de akár nyomtató vezérlőre (**`QPrinter`**) is.
  - A kirajzolást a vezérlők felülírható **`paintEvent(QPaintEvent*)`** metódusa végzi, amely akkor fut le
    - amikor a rendszer a megjelenítést frissíti, vagy
    - amikor az **`update()`** eseménykezelőt közvetlenül meghívjuk (pl. időzítővel történő frissítés esetén).

Rajzoló eszköz

- A rajzolás a **`QPainter`** típusú rajzoló objektummal végezzük.
  - A konstruktorának átadjuk a rajzfelületet (általában az aktuális vezérlő)

    pl. **`QPainter painter(this).`**
  - Beállítjuk a rajzolási tulajdonságokat (szín, háttérszín, vonaltípus, betűtípus, …) setter metódusokkal, amelyek hatása a következő beállításig tart.

```
painter.setBackground(<kitöltés>);   // háttérszín
painter.setFont(<betűtípus>);        // szöveg betűtípusa
painter.setOpacity(<mérték>);        // átlátszóság
```

Alakzatok rajzolása

- A rajzoló eszköz **`draw`**-szerű műveleteivel rajzolhatunk alakzatot, szöveget, képet.
- A rajzoló a műveletek sorrendben futnak le, és egymásra rajzolnak.
- Egy alakzat lehet: pont, vonal, törtvonal, téglalap (lekerekített sarokkal), ellipszis, körcikk, körszelet, körcikk, stb. Bizonyos alakzatoknál (pont és vonal esetében nem) az alakzat az aktuálisan beállított ecsetszínnel lesz kitöltve.

```
painter.drawRect(10, 30, 50, 30);
   // 50x30-as téglalap kirajzolása a (10,30) koordinátába
painter.fillRect(20, 40, 50, 30);
   // keret nélküli téglalap kirajzolása
painter.drawText(20, 50, "Hello");
   // szöveg a (20,50) koordinátába
```

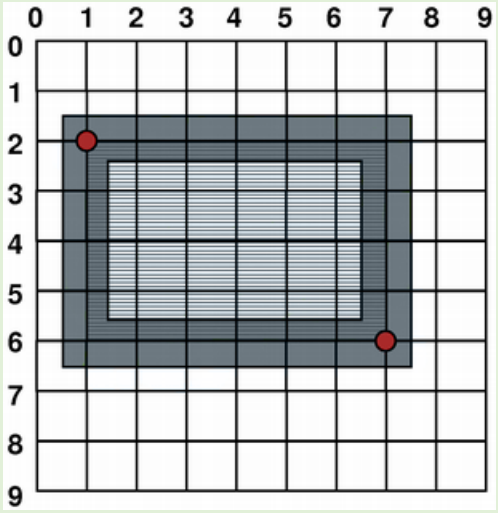
Tollak, ecsetek

- A keretet, szöveget toll (**QPen**) segítségével készítjük, amely lehet egyszínű, de tartalmazhat szaggatásokat (dash, dot, dashdot, …), nyilakat, beállíthatjuk a vonalvégeket (flat, square, round), a vonal találkozásokat (miter, bevel, round)
- A kitöltést ecset (**QBrush**) segítségével készítjük, amely lehet egyszínű, adott mintájú, textúrájú, …

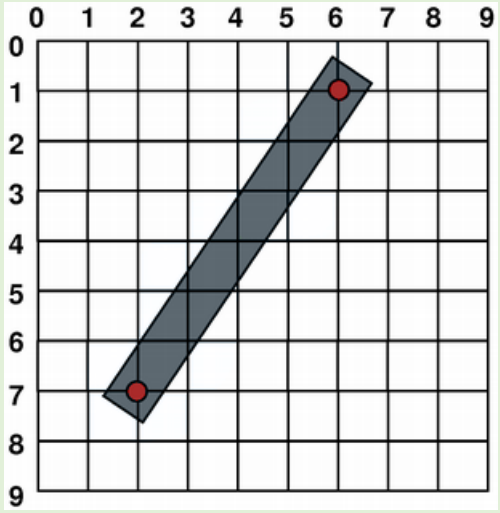
```
painter.setPen(Qt::darkGreen);
    // 1 vastag sötétzöld toll
painter.setPen(QPen(QColor(Qt::blue), 4, Qt::DotLine));
    // 4 vastag pöttyös kék toll
painter.setBrush(QBrush(QColor(250, 53, 38),   Qt::CrossPattern));
    // rácsos vöröses ecset
```

Rajzolás logikai koordinátái

- A rajzolást úgynevezett „logikai” koordináták segítségével végezzük, ezek határozzák meg az alakzat sarokpontjait.



`QRect(1,2,6,4)`

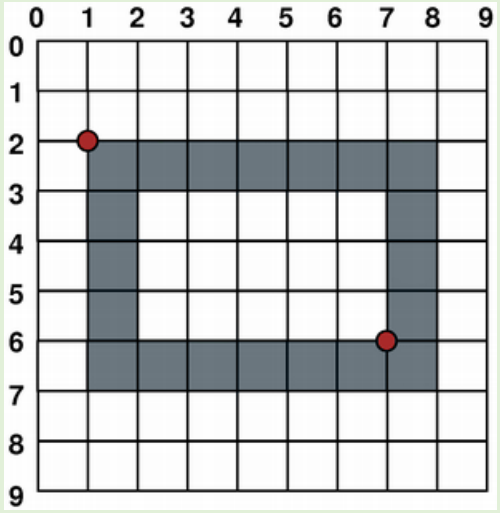


`QLine(2,7,6,1)`

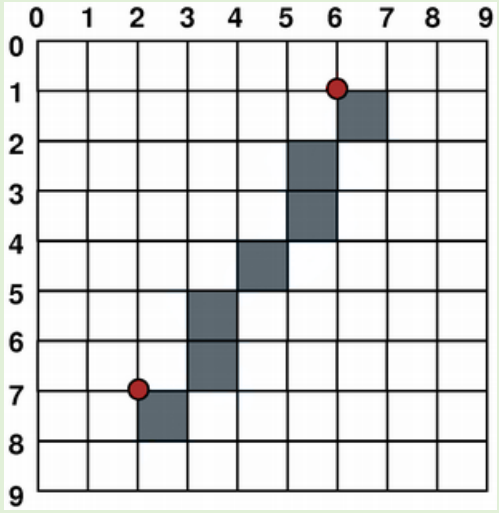
- A rendszer áttranszformálja az adatokat „fizikai” koordinátákká (*viewport*)

Rajzolás fizikai koordinátái

- A rajzolási műveletek az alakzatot a megfelelő képpontok koordinátáira (az ablak koordinátáira) igazítják.




```
drawRect(1,2,6,4);
```

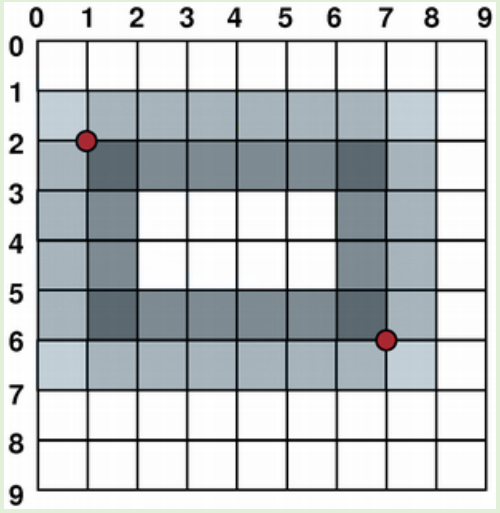



```
drawLine(2,7,6,1);
```

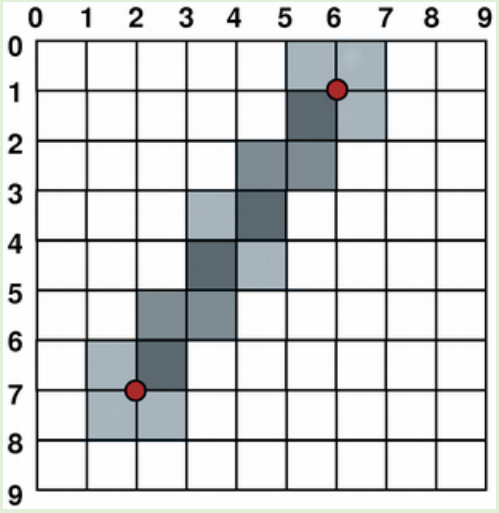
- Amennyiben a toll vastagsága páratlan, jobbra és lefelé tolódik az elhelyezés.

Simítás

- Lehet simítást alkalmazni a rajzoláskor, hogy a logikai koordinátán helyezkedjen el a rajz.




```
drawRect(1,2,6,4);
```

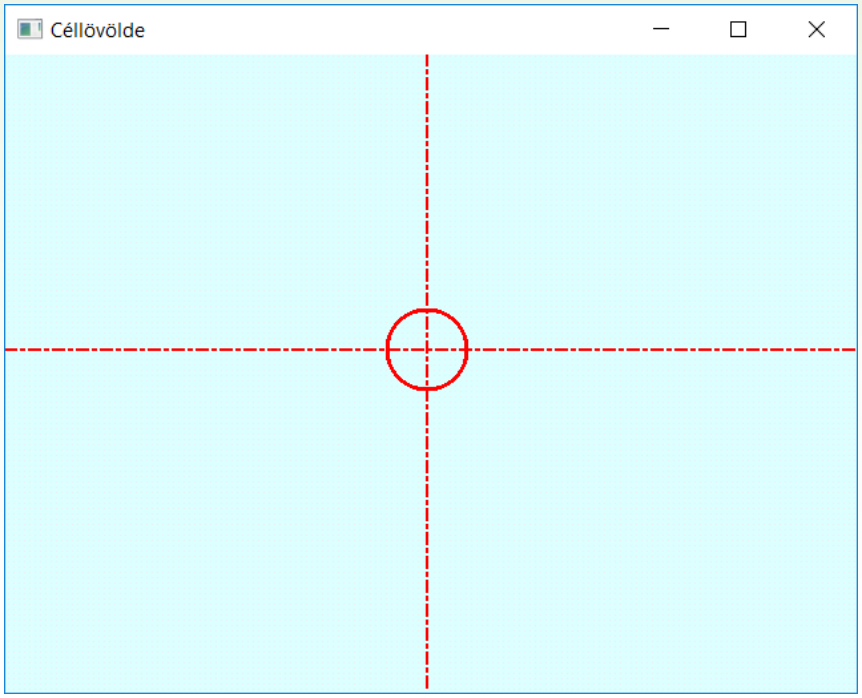



```
drawLine(2,7,6,1);
```

- Ehhez a **`setRenderHint(QPainter::Antialiasing)`** üzemmódot kell a rajzolóra beállítanunk.

1.Feladat

Készítsünk egy alkalmazást, amelyben egy célkeresztet helyezünk az ablak közepére. A célkeresztet két szaggatott-pöttyözött piros vonallal és egy piros körrel jelenítjük meg, a hátteret pöttyös világoskék ecsettel festjük ki.



1.Feladat: megoldás

- Felüldefiniáljuk egy ablak (**`Qwidget`**) **`paintEvent`** metódusát.
- Létrehozunk benne egy rajzoló objektumot (**`painter`**).
- Kitöltjük a hátteret a **`fillRect`** utasítással, majd meghúzzuk a függőleges és vízszintes vonalakat (**`drawLine`**), végül a közepére állítunk egy kört (**`drawEllipse`**).
- A rajzolások közben megfelelően állítjuk a tollat és az ecsetet (az ecsetet kikapcsoljuk az ellipszis rajzolása előtt).

1.Feladat: megvalósítás

```
void CrosshairWidget::paintEvent(QPaintEvent *)
{
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);
    QPen dashDotRedPen(QBrush(QColor(255,0,0)),2, Qt::DashDotLine);
    QPen solidRedPen(QBrush(QColor(255, 0, 0)), 3);
    QBrush blueBrush(QColor(230, 255, 255), Qt::Dense1Pattern);

    painter.setBrush(blueBrush);
    painter.fillRect(0, 0, width(), height());
    painter.setPen(dashDotRedPen);
    painter.drawLine(0, height() / 2, width(), height() / 2);
    painter.drawLine(width() / 2, 0, width() / 2, height());
    painter.setPen(solidRedPen);
    painter.drawEllipse(width()/2 - 30, height()/2 - 30, 60, 60);
}
```

pontozott-szaggatott vonalú piros toll

piros toll

pöttyös világoskék ecset

alakzatok kirajzolása

Transzformációk

- A rajzoló objektum alapértelmezés szerint az adott vezérlő koordinátarendszerében dolgozik, de lehetőségünk van ennek affin transzformálására (**`worldTransform`**).
  - forgatás (**`rotate(<szög>)`**)
  - méretezés (**`scale(<vízszintes>, <függőleges>)`**)
  - áthelyezés (**`translate(<vízszintes>, <függőleges>)`**)
  - ferdítés (**`shear(<vízszintes>, <függőleges>)`**)
- Így egy újabb szint (transzformáció) épülhet be a fizikai (**`viewPort`**) koordináták és a logikai koordináták közé (a logikai koordináták először az affin transzformációkkal ablak koordinátákká, majd fizikai koordinátákká változnak.)

Egérkezelő műveletek

- Az egérkezelés (követés, kattintás lekérdezése) bármely vezérlő területén elvégezhető.
- Egy saját vezérlő négy öröklött eseményvizsgálót írhat felül:
  - **`mousePressEvent`** : egér lenyomása
  - **`mouseReleaseEvent`** : egér felengedése
  - **`mouseMoveEvent`**: egér mozgatása
  - **`mouseDoubleClickEvent`** : dupla kattintás
- Minden eseményvizsgáló egy **`MouseEvent`** paramétert kap, amely tartalmazza az egér pozícióját lokálisan (**`pos()`**) és globálisan (**`globalX()`**, **`globalY()`**), illetve a használt gombot (**`button()`**).
- Az egérkövetés alapértelmezetten csak lenyomott gomb mellett működik, de ez átállítható állandóra a **`mouseTracking`** tulajdonság állításával.

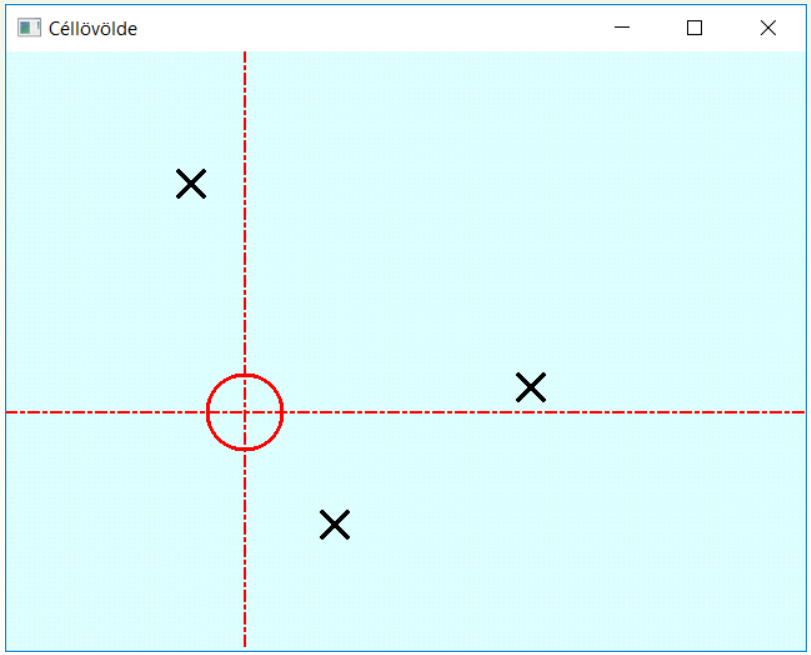
Billentyűzetkezelő műveletek

- A saját vezérlőinkben két öröklött eseményvizsgálót írhatunk felül:
  - **`keyPressEvent`** : billentyű lenyomása
  - **`keyReleaseEvent`** : billentyű felengedése
- Minden eseményvizsgáló egy **`QKeyEvent`** paramétert kap, amelyből kinyerhető az érintett billentyűt (**`key`**) beazonosító információ.

2.Feladat

A célkereszt megjelenítő programunkban kövesse az egérkurzort a célkereszt mindaddig, amíg az egérkurzor az ablak felett van.

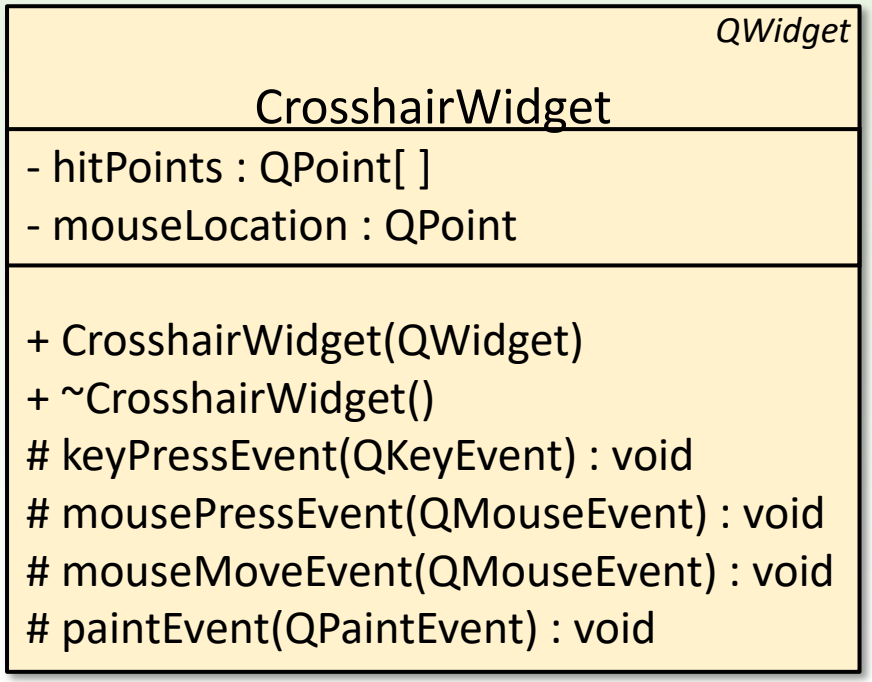
Az egérfül vagy a szóköz billentyű lenyomásával lehessen lőni úgy, hogy a lövés helyén egy fekete X jelenik meg.



2.Feladat: megoldás

- Beállítjuk állandóra (**`setMouseTracking(true)`**) az egérkövetést a konstruktorban.
- Külön adattagban tároljuk az aktuális kurzor-pozíciót (**`mouseLocation`**). Ezt az egérkövetés felüldefiniált eseményvizsgálója frissíti folyamatosan.
- Külön sorozatban (**`hitPoints`**) tároljuk az eddigi találatokat (lövések célkereszt-pozícióit). Ehhez felüldefiniáljuk az egér/billentyű lenyomás eseményvizsgálóit úgy, hogy azok minden kattintásnál elmentsék az újabb lövést a korábbiak mellé.
- Mindhárom előbb említett eseményvizsgáló frissítse a kijelzőt (**`update()`**).
- Kirajzolásnál (**`paintEvent`**) az elmentett pontokat is kirajzoljuk.

2.Feladat: tervezés



2.Feladat: megvalósítás

```
CrosshairWidget::CrosshairWidget(QWidget *parent): QWidget(parent)
{
    setWindowTitle(tr("Céllövölde"));
    setMouseTracking(true); // állandóan követjük az egeret
}

void CrosshairWidget::paintEvent(QPaintEvent *)
{
    …
    painter.setPen(QPen(Qt::black, 4)); // fekete, 4 vastag toll
    foreach(QPoint point, hitPoints) {
        painter.drawLine(point.x() - 10, point.y() - 10,
                         point.x() + 10, point.y() + 10);
    }
    painter.setPen(dashDotRedPen);
    painter.drawLine(0, mouseLocation.y(), width(), mouseLocation.y());
    painter.drawLine(mouseLocation.x(), 0, mouseLocation.x(), height());
    painter.setPen(solidRedPen);
    painter.drawEllipse(mouseLocation.x()-30, mouseLocation.y()-30, 60, 60);
```

egérkövetés

eddigi találatok kirajzolása

célkereszt az egérkurzor alatt

2.Feladat: megvalósítás

```
void CrosshairWidget::mouseMoveEvent(QMouseEvent*event)
{
     mouseLocation = event->pos();
     update(); // képernyő frissítése
}
void CrosshairWidget::mousePressEvent(QMouseEvent *event)
{
     hitPoints.append(event->pos()); // új pont felvétele
     update();                       // képernyő frissítése
}
void CrosshairWidget::keyPressEvent(QKeyEvent *event)
{
     if (event->key() == Qt::Key_Space) { // ha szóközt ütöttünk
        hitPoints.append(mouseLocation);  // új pont felvétele
        update();                         // képernyő frissítése
     }
}
```

egérpozíció lekérése

új találat rögzítése

új találat rögzítése

Kurzorkezelés

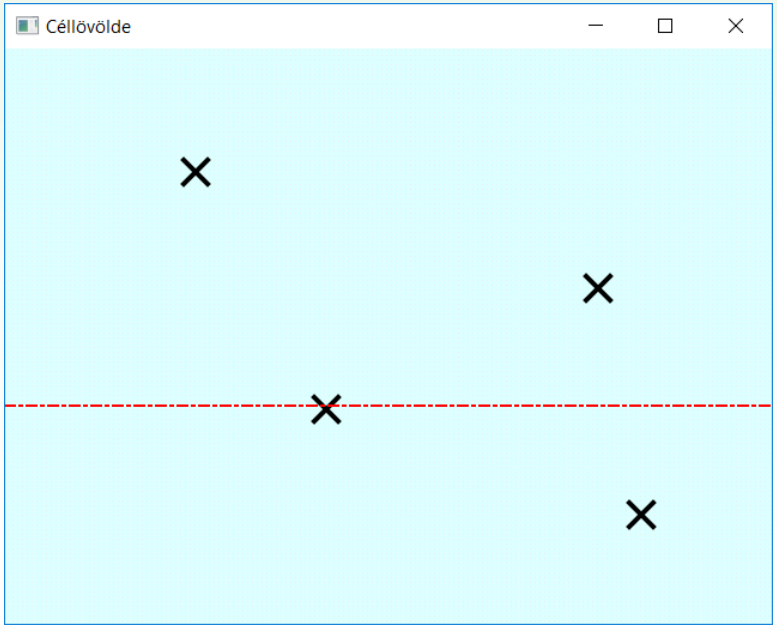
- Az egérkezelő műveletektől függetlenül is állíthatjuk az egérpozíciót a kurzorkezelés (**`QCursor`**) segítségével.
  - A kurzor mindig az egérpozícióval egybeeső helyen van, amely lekérdezhető, és beállítható (**`QCursor::pos()`**).
  - A kurzornak módosítható a kinézete (pl. nyíl, kéz, homokóra, …), vagy beállítható tetszőleges kép.

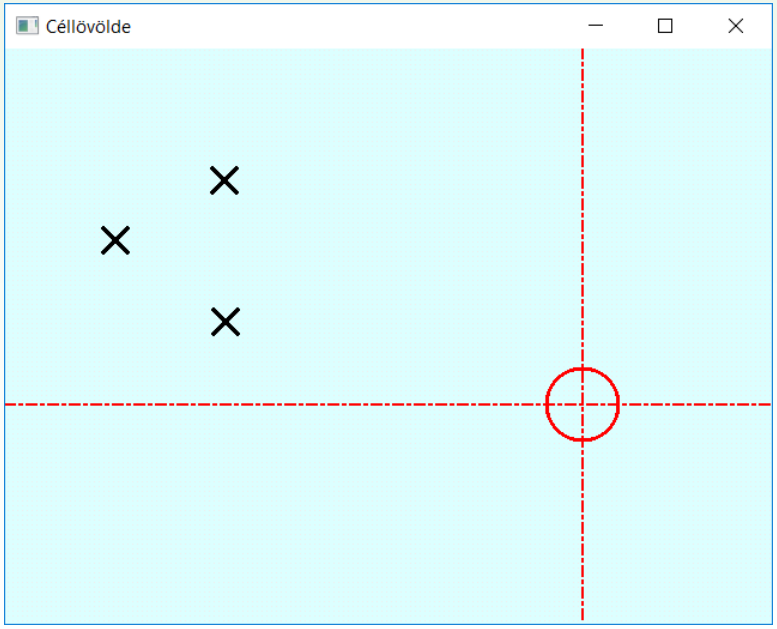
```
widget.setCursor(QCursor(Qt::CrossCursor)); // kereszt
widget.setCursor(QCursor(Qt::BusyCursor));  // homokóra
```

- A kurzortól lekért pozíció globális, de minden vezérlőnél lehetőségünk van leképezni a lokális koordinátarendszerbe a **`QWidget::mapFromGlobal(<pozíció>)`** művelettel.

3.Feladat

Módosítsuk a célkereszt megjelenítő programunkat úgy, hogy az egérkurzor maga is egy kereszt legyen, amely takarásban ugyan, de akkor is kövesse az egérkurzor mozgását, ha az az ablakon kívül van.

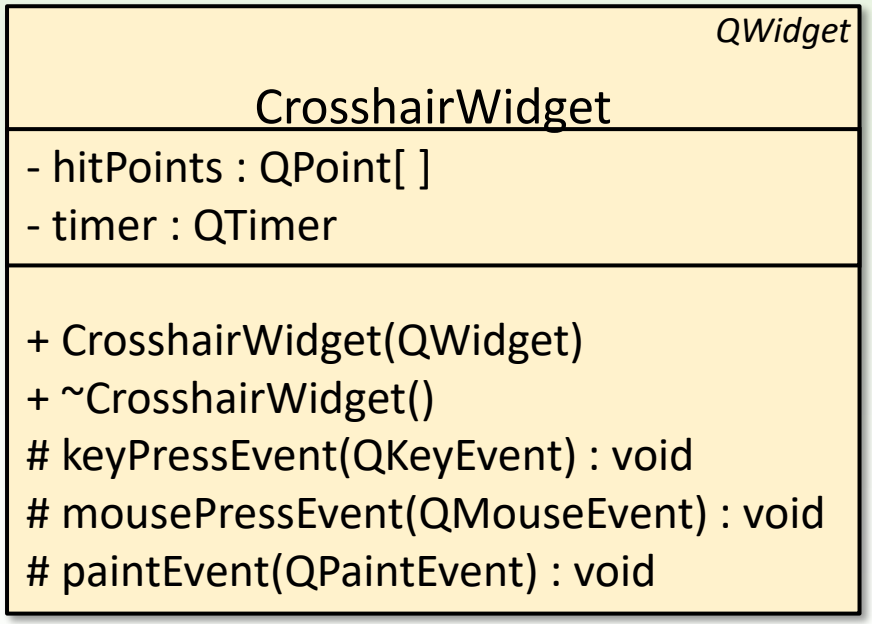




3.Feladat: megoldás

- A konstruktorban módosítjuk a kurzormegjelenést (`setCursor(Qt::CrossCursor)`).
- Mivel nincs egérkövetés, nem tudunk egéreseményre reagálva rajzolni, ezért időzítő segítségével meghatározott időközönként (0.01 másodperc) frissítjük a képernyőt, ahol mindig lekérjük az aktuális kurzorpozíciót a rajzolásnál.
- Az egér/billentyű lenyomás eseményét megtartjuk, ebben továbbra is felvesszük az újabb lövések célkereszt-pozíciót a korábbiak mellé a `hitPoints` vektorba.

3.Feladat: tervezés



3.Feladat: megvalósítás

```
CrosshairWidget::CrosshairWidget(QWidget *parent): QWidget(parent){
    setWindowTitle(tr("Céllövölde"));

    timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(update()));
    timer->start(10);

    setCursor(Qt::CrossCursor); // kurzor beállítása célkeresztre
}
```

az update hívja a paintEvent()-t

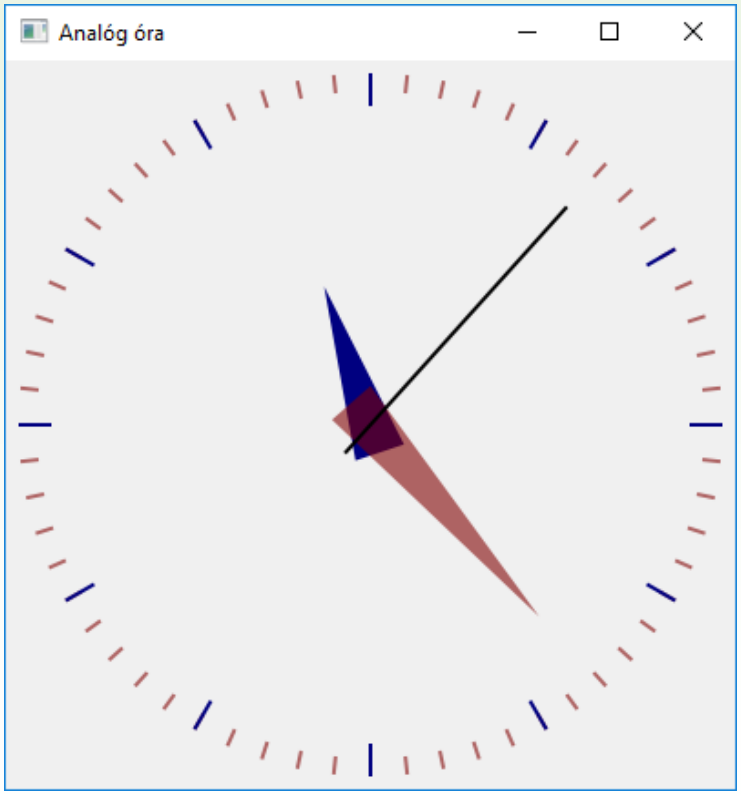
```
void CrosshairWidget::paintEvent(QPaintEvent *)
{
    …
    QPoint mouseLocation = QCursor::pos();
    mouseLocation = QWidget::mapFromGlobal(mouseLocation);
    …
```

egérpozíció lekérdezése a képernyőn

egérpozíció transzformálása az ablakra

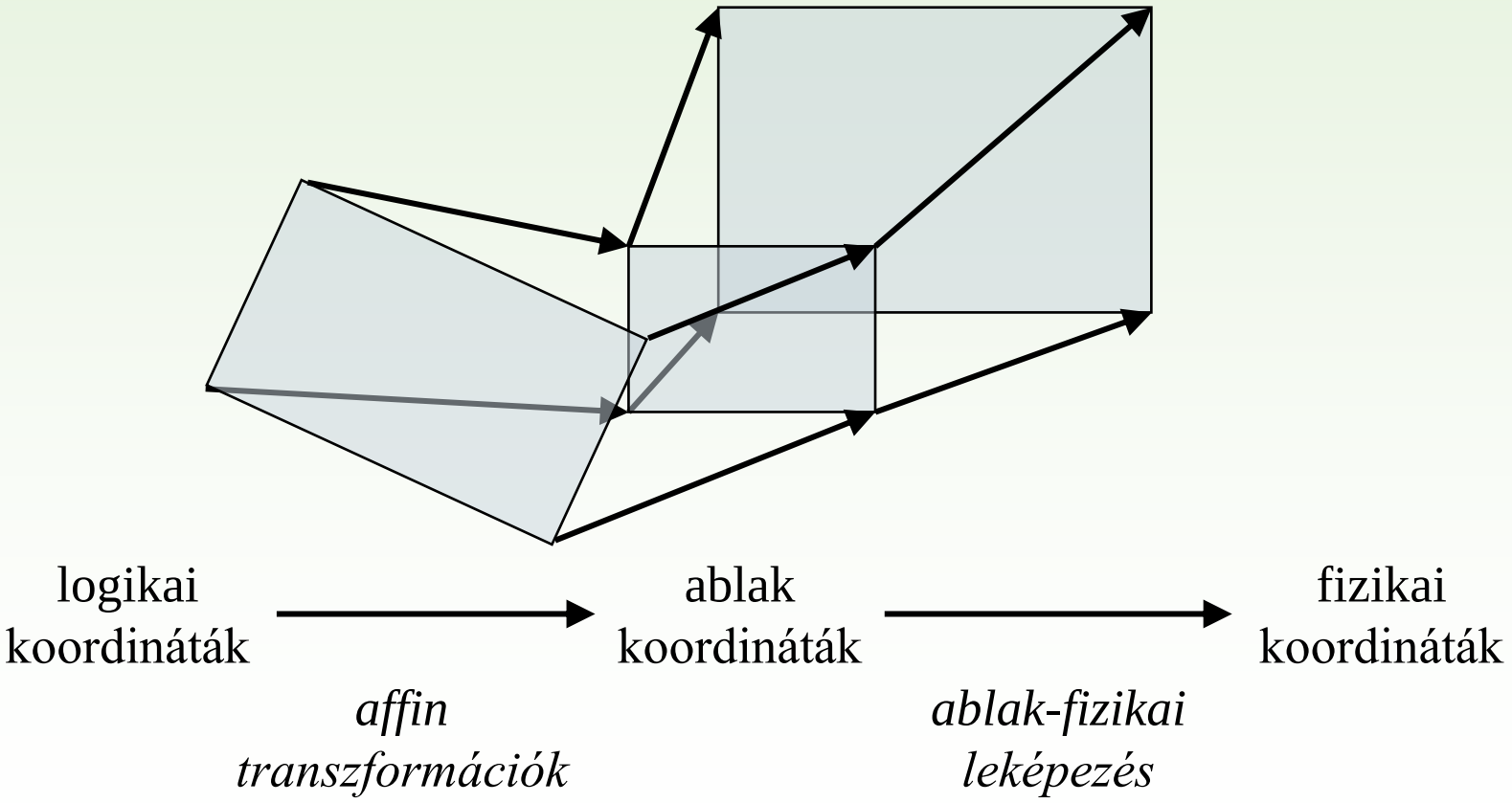
4.Feladat

Készítsünk egy analóg órát, amely mutatja az aktuális időt.



Logikai – ablak – fizikai koordináták

- Minden leképezés transzformációs mátrixok alkalmazásával történik.



További rajzolási lehetőségek

- A hátteret külön állíthatjuk (**`background`**), ekkor a teljes rajzfelület változik, a rárajzolt tartalom külön törölhető (**`erase()`**).
- Amennyiben több tulajdonság beállítását is elvégezzük a rajzolás során, lehetőségünk van korábbi beállítások visszatöltésére.
  - A **`save()`** művelettel elmenthetjük az aktuális állapotot, a **`restore()`** művelettel betölthetjük az utoljára mentettet.
- A rajzolás tartalmát megvághatjuk téglalap (**`clipRegion`**), vagy egyéni alakzat (**`clipPath`**) alapján.
- Több rajzot is összeilleszthetünk különböző műveleti sémák szerint (**`compositionMode`**).

4.Feladat: megoldás

- Az aktuális időt (**`QTime::currentTime()`**) egy időzítő segítségével kérdezgetjük le, és jelenítjük meg.
- Mind az óra rovátkáit, mind a mutatóit a kirajzolásuk után forgatjuk (**`rotate`**) a megfelelő helyre. Ezt kényelmesebbé azzal tehetjük, ha eltoljuk (**`translate`**) és átméretezzük (**`scale`**) a koordinátarendszert úgy, hogy az ablak közepén legyen az origó.
- Az óra és perc mutatókat háromszögként ábrázoljuk némi áttetszéssel (**`drawConvexPolygon`**).

4.Feladat: megvalósítás

```
AnalogClockWidget::AnalogClockWidget(QWidget *parent)
     : QWidget(parent) {
     setWindowTitle(tr("Analóg óra"));
     resize(400, 400);
     QTimer *timer = new QTimer(this);
     connect(timer, SIGNAL(timeout()), this, SLOT(update()));
     timer->start(1000);
}
```

az update hívja a paintEvent-t

```
void AnalogClockWidget::paintEvent(QPaintEvent *){
     // mutatók geometriája:
     QPoint hourTriangle[3] =
          { QPoint(7, 8), QPoint(-7, 8), QPoint(0, -40) };
     QPoint minute Triangle[3] =
          { QPoint(7, 8), QPoint(-7, 8), QPoint(0, -70)};
     // mutatók és vonások színe:
     QColor hourColor(0, 0, 128);           // RGB
     QColor minuteColor(128, 0, 0, 150);    // RGB és áttetszőség
     …
```

4.Feladat: megvalósítás

```
…
QPainter painter(this);                          // festő objektum
painter.setRenderHint(QPainter::Antialiasing); // élsimítás
painter.translate(width() / 2, height() / 2);   // transzformációk
painter.scale(height() / 200.0, height() / 200.0);
// percjelek rajzolása:
painter.setPen(minuteColor);
for (int j = 0; j < 60; ++j){
    painter.drawLine(92, 0, 96, 0);    // vonal kirajzolása
    painter.rotate(6.0);               // forgatás 6 fokkal
}
// órajelek rajzolása:
painter.setPen(hourColor);
for (int i = 0; i < 12; ++i){
    painter.drawLine(88, 0, 96, 0);
    painter.rotate(30.0);              // forgatás 30 fokkal
}
…
```

origó az ablak közepére

A vásznat forgatjuk

4.Feladat: megvalósítás

```
    …
    QTime time = QTime::currentTime(); // aktuális idõ

// óramutató rajzolása:
    painter.save();                    // rajzolási tulajdonságok elmentése
    painter.setPen(Qt::NoPen);         // nincs toll
    painter.setBrush(hourColor);       // ecset színe
    painter.rotate(30.0 * ((time.hour() + time.minute() / 60.0)));
    painter.drawConvexPolygon(hourTriangle, 3); // poligon kirajzolása
    painter.restore();                 // rajzolás visszaállítása

// percmutató rajzolása:
    …
// másodpercmutató rajzolása:
    …
}
```

A vásznat forgatjuk