Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások fejlesztése I

4. előadás

Elemi grafika és egérkezelés

Giachetta Roberto

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

Elemi grafika és egérkezelés

Rajzolás grafikus felületen

- Qt-ban a grafikus felhasználói felület tartalmát tetszőlegesen „rajzolhatjuk”, azaz primitív 2D-s alakzatokat (vonal, téglalap, ellipszis, …) helyezhetünk fel rá, ezáltal teljesen egyedi megjelenítést adhatunk neki
  - pl.:

```
void MyWidget::paintEvent(QPaintEvent*)
{
   QPainter painter(this); // rajzoló objektum
   painter.setPen(Qt::blue); // toll
   painter.drawRect(rect());
      // kék keret kirajzolása
   painter.drawText(rect(), Qt::AlignCenter,
      "Hello World!"); // szöveg kirajzolása
}
```

Elemi grafika és egérkezelés

Rajzolási felület

- A rajzolást egy megadott felületen végezzük
  - mindenre rajzolhatunk, ami a **`QPaintDevice`** leszármazottja, így tetszőleges grafikus vezérlő (**`QWidget`**), kép (**`QPixmap`**) és a nyomtató (**`QPrinter`**)
  - magát a kirajzolást az osztály **`paintEvent(QPaintEvent*)`** metódusa végzi, ezt felüldefiniálva adjuk meg az egyedi rajzolást
    - automatikusan fut le, amikor a rendszer frissíti a megjelenítést
    - az **`update()`** eseménykezelőn keresztül manuálisan is lehet futtatni (pl. időzítővel történő frissítés esetén szükséges)

Elemi grafika és egérkezelés

Rajzolási eszközök

- A rajzolásért egy rajzoló objektum felel, amely a **`QPainter`** típus példánya
  - a konstruktornak átadjuk a rajzfelületet (általában az aktuális vezérlő), pl.:

```
QPainter painter(this);
    // a rajzolási felület ez a vezérlő lesz
```

  - beállítjuk a rajzolási tulajdonságokat (szín, vonaltípus, betűtípus, ...) a **`set<paraméter>(<érték>)`** metódusokkal, (hatása a következő beállításig tart), pl.:

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

Elemi grafika és egérkezelés

Rajzolási eszközök

- A rajzolást a **draw<*alakzat/szöveg/kép*>(<*elhelyezkedés, …*>)** műveletekkel végezhetjük, alakzatoknál kitöltést a **fill<*alakzat*>(…)** művelettel adhatunk meg, pl.:

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

  - a műveletek sorrendben futnak le, egymásra rajzolnak
  - a rajzolás az alakzat bal felső sarkától indul (kivéve szöveg)

Elemi grafika és egérkezelés

Ecsetek és tollak

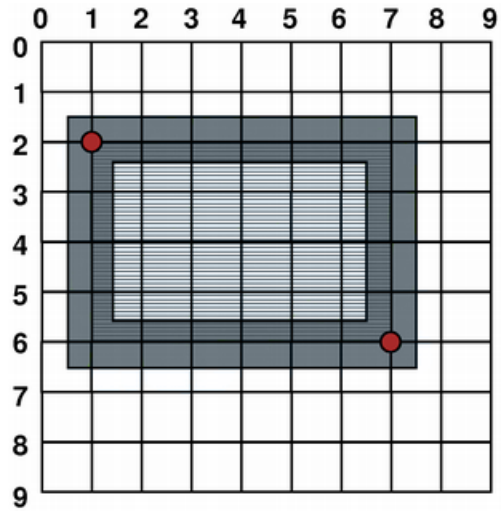
- A rajzolási beállítások elkülönülnek annak megfelelően, hogy az alakzat keretét, vagy kitöltését rajzoljuk
  - a keretet, szöveget *toll* (**`QPen`**) segítségével készítjük, amely lehet egyszínű, de tartalmazhat szaggatásokat, nyilakat, …
  - a kitöltést *ecset* (**`QBrush`**) segítségével készítjük, amely lehet egyszínű, adott mintájú, textúrájú, …
  - pl.:

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

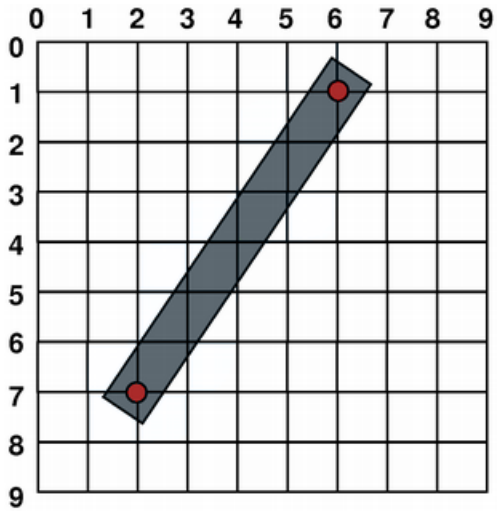
Elemi grafika és egérkezelés

Rajzolási koordináták

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



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



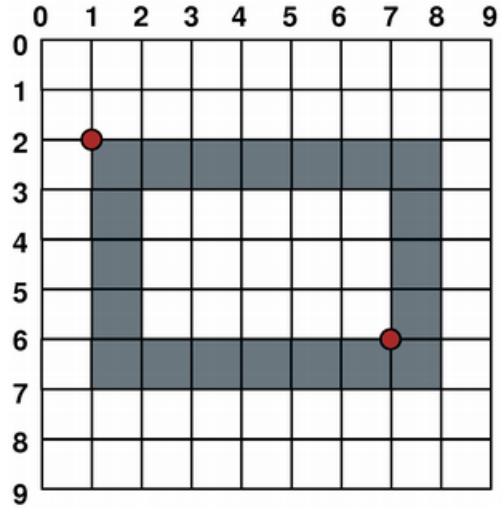
`QLine(2,7,6,1)`

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

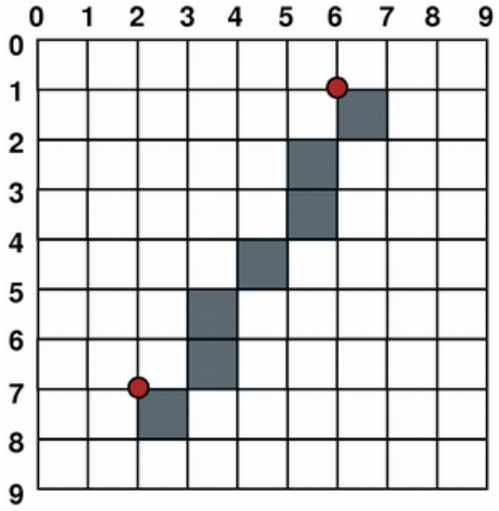
Elemi grafika és egérkezelés

Rajzolási koordináták

- A rajzolási műveletek az alakzatot a megfelelő képpontok koordinátáira igazítják




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



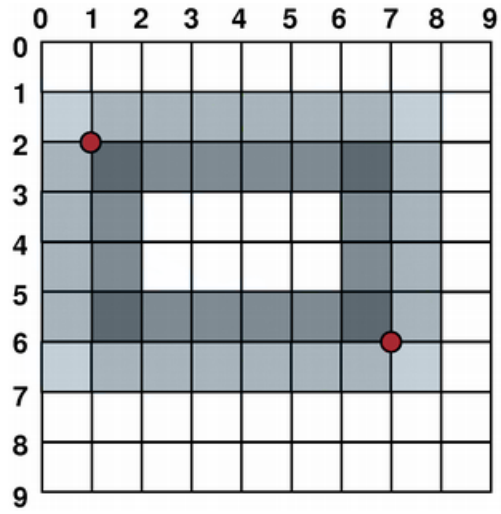

```
drawLine(2,7,6,1);
```

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

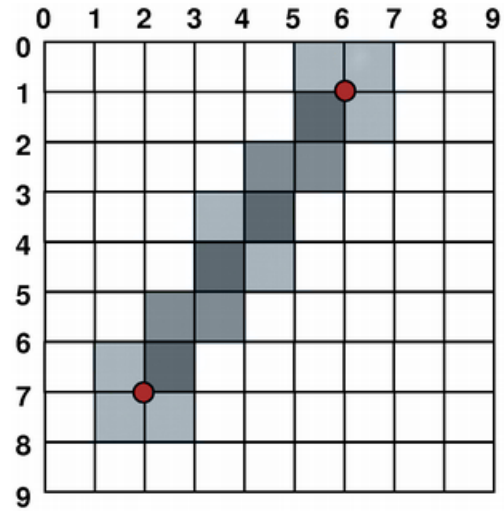
Elemi grafika és egérkezelés

Rajzolási koordináták

- Lehetőségünk elsimítást alkalmazni a rajzoláskor, ekkor minden esetben a logikai koordinátán helyezkedik el a rajz



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



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

- ehhez a rajzoló **`setRenderHint(QPainter::Antialiasing)`** üzemmódját kell beállítanunk

Elemi grafika és egérkezelés

Példa

Feladat: Készítsünk egy alkalmazást, amelyben egy célkeresztet helyezünk az ablak közepére. A célkeresztet két vonallal és egy körrel jelenítjük, szaggatott-pöttyözött piros színnel, míg a hátteret pöttyös zöld ecsettel festjük meg.

- felüldefiniáljuk az ablak **`paintEvent`** metódusát, létrehozunk benne egy rajzobjektumot (**`painter`**)
- először kitöltjük a hátteret a **`fillRect`** utasítással, majd meghúzzuk a függőleges és vízszintes vonalakat (**`drawLine`**), végül a közepére állítunk egy ellipszist (**`drawEllipse`**)
- a rajzolások közben megfelelően állítjuk a tollat és az ecsetet (az ecsetet kikapcsoljuk az ellipszis rajzolása előtt)

Elemi grafika és egérkezelés

Példa

Megvalósítás (`crosshairwidget.cpp`*):*

```
void CrosshairWidget::paintEvent(QPaintEvent *)
{
   QPainter painter(this); // rajzoló objektum
   painter.setRenderHint(QPainter::Antialiasing);
      // élsimítás használata

   QPen dashDotRedPen(QBrush(QColor(255, 0, 0)),
      2, Qt::DashDotLine);
      // pontozott-szaggatott vonalú piros toll
   QPen solidRedPen(QBrush(QColor(255, 0, 0)), 3);
      // sima piros toll
   QBrush greenBrush(QColor(0, 255, 0),
      Qt::Dense1Pattern); // pöttyös zöld ecset
```

Elemi grafika és egérkezelés

Példa

Megvalósítás (`crosshairwidget.cpp`*):*

```
    painter.setBrush(greenBrush); // ecset állítás
    painter.fillRect(0, 0, width(), height());
       // háttér kitöltése

    painter.setPen(dashDotRedPen); // toll állítás
    painter.drawLine(0, height() / 2, width(),
       height() / 2); // vonalak kirajzolása
    painter.drawLine(width() / 2, 0, width() / 2,
       height());
    painter.setPen(solidRedPen); // toll állítás
    painter.drawEllipse(width() / 2 - 30, height()
       / 2 - 30, 60, 60); // kör kirajzolása
}
```

Elemi grafika és egérkezelés

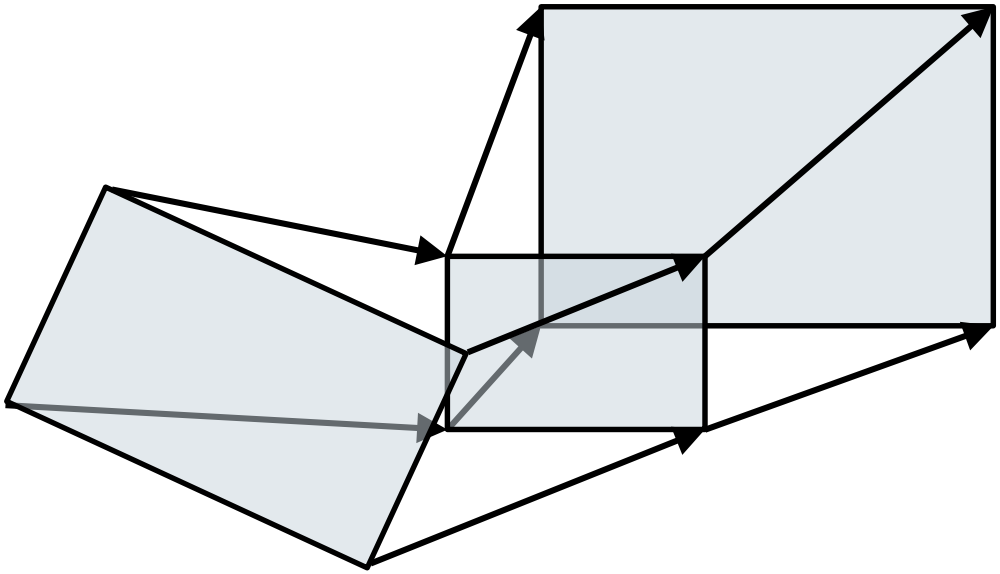
Transzformációk

- Alapból a rajzoló objektum a megadott 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>)`**)
- Az így keletkezett ablak (**`window`**) koordináták és a fizikai (**`viewPort`**) koordináták között újabb megfeleltetést létesíthetünk, más transzformációkkal (azaz két lépcsős a transzformáció)

Elemi grafika és egérkezelés

Transzformációk

- minden leképezés transzformációs mátrixok alkalmazásával történik



logikai koordináták → *affin transzformációk* → ablak koordináták → *ablak-fizikai leképezés* → fizikai koordináták

Elemi grafika és egérkezelés

További rajzolási lehetőségek

- A hátteret külön állíthatjuk (**`background`**), ekkor a teljes rajzfelület változik, a rárajzolt tartalom törölhető is (**`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`**)

Elemi grafika és egérkezelés

Példa

Feladat: Készítsünk egy analóg órát, amely mutatja az aktuális időt.

- az aktuális idő mutatásához időzítőt használunk és mindig lekérdezzük az aktuális időt (**`QTime::currentTime()`**)
- az óra és perc mutatókat háromszögből rajzoljuk ki (**`drawConvexPolygon`**, némi áttetszéssel), és a megfelelőhelyre forgatjuk (**`rotate`**), hasonlóan forgatjuk a többi jelölőt és mutatót, de azok már vonalak lesznek
- az egyszerűbb forgatás és helyezés érdekében eltoljuk (**`translate`**) és méretezzük (**`scale`**) a koordinátarendszert, hogy az ablak közepén legyen az origó

Elemi grafika és egérkezelés

Példa

Megvalósítás (`analogclockwidget.cpp`*):*

```
AnalogClockWidget::AnalogClockWidget(QWidget
    *parent) : QWidget(parent)
{
    …
    QTimer *timer = new QTimer(this); // időzítő
    connect(timer, SIGNAL(timeout()), this,
        SLOT(update()));
        // az időzítő meghívja az update-t, ami a
        // paintEvent-t
    timer->start(1000);
        // azonnal elindítjuk 1 másodperces
        // késleltetéssel
}
```

Elemi grafika és egérkezelés

Példa

Megvalósítás (`analogclockwidget.cpp`*):*

```
void AnalogClockWidget::paintEvent(QPaintEvent *){
…
   QTime time = QTime::currentTime();
      // idő
   painter.save(); // tulajdonságok elmentése
   painter.setPen(Qt::NoPen); // nincs toll
   painter.setBrush(hourColor); // ecset színe
   painter.rotate(30.0 * ((time.hour() +
      time.minute() / 60.0))); // mutató forgatása
   painter.drawConvexPolygon(hourTriangle, 3);
      // poligon kirajzolása
   painter.restore(); // rajzolás visszaállítása
…
```

Elemi grafika és egérkezelés

Egérkezelő műveletek

- Az egérkezelés (követés, kattintás lekérdezése) bármely vezérlő területén elvégezhető, műveletek felüldefiniálásával
- 4 eseménykezelő áll rendelkezésünkre:
  - egér lenyomása (**`mousePressEvent`**) és felengedése (**`mouseReleaseEvent`**)
  - egér mozgatása (**`mouseMoveEvent`**)
  - dupla kattintás (**`mouseDoubleClickEvent`**)
- Minden eseménykezelő **`MouseEvent`** paramétert kap, amely tartalmazza az egér pozícióját lokálisan (**`pos()`**) és globálisan (**`globalX()`**, **`globalY()`**), illetve a használt gombot (**`button()`**)

Elemi grafika és egérkezelés

Egérkezelő műveletek

- Az egérkövetés alapértelmezetten csak lenyomott gomb mellett működik, de ez átállítható állandóra a **`mouseTracking`** tulajdonság állításával

- Pl.:

```
class MyWidget {
…
protected:
    void mousePressEvent(MouseEvent* event);
    void mouseReleaseEvent(MouseEvent* event);
    void mouseMoveEvent(MouseEvent* event);
    void mouseDoubleClickEvent(MouseEvent* event);
    // minden egéreseményt kezelünk
}
```

Elemi grafika és egérkezelés

Billentyűkezelő műveletek

- Az egérkövetésnek megfelelően van lehetőségünk billentyűzetkövetésre is, pontosabban billentyű lenyomásának (**`keyPressEvent`**) és felengedésének (**`keyReleaseEvent`**) kezelésére
  - a paraméter (**`QKeyEvent`**) tartalmazza a billentyűt (**`key`**)
  - pl.:

```
class MyWidget {
…
protected:
    void keyPressEvent(QKeyEvent* event);
    void keyReleaseEvent(QKeyEvent* event);
    // billentyűesemények kezelése
}
```

Elemi grafika és egérkezelés

Példa

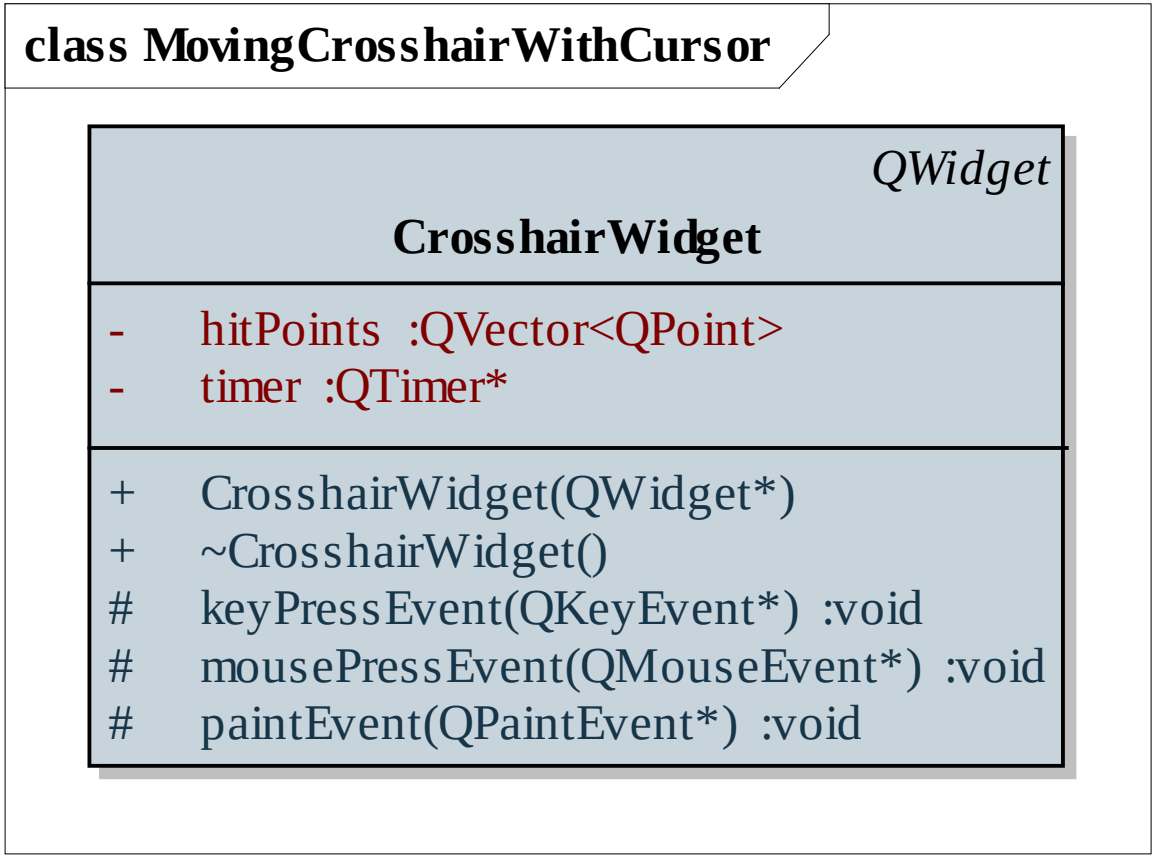
Feladat: Módosítsuk a célkereszt megjelenítő programunkat úgy, hogy kövesse az egeret, és egérgombra, illetve szóköz billentyűre lehessen lőni is, amit úgy jelenítünk meg, hogy egy fekete X-et rajzolunk a helyére.

- felüldefiniáljuk az egér/billentyű lenyomás és egér követés eseményeket és beállítjuk, hogy mindig kövesse az egeret (**`setMouseTracking(true)`**), az egérpozíciót elmentjük (**`mouseLocation`**)
- minden egérmozgásnál frissítjük a kijelzőt (**`update()`**), kattintásnál elmentjük az aktuális pozíciót egy vektorba (**`hitPoints`**)
- a kirajzolásnál az elmentett pontokat is kirajzoljuk

Elemi grafika és egérkezelés

Példa

Tervezés:



Elemi grafika és egérkezelés

Példa

Megvalósítás (`crosshairwidget.cpp`*):*

```
void CrosshairWidget::mousePressEvent(QMouseEvent
   *event){
   hitPoints.append(event->pos());
      // új pont felvétele
   update(); // képernyő frissítése
}

void CrosshairWidget::mouseMoveEvent(QMouseEvent
   *event){
   mouseLocation = event->pos();
   update();
}
```

Elemi grafika és egérkezelés

Példa

Megvalósítás (`crosshairwidget.cpp`*):*

```
void CrosshairWidget::paintEvent(QPaintEvent *){
   foreach(QPoint point, hitPoints){
      // kirajzoljuk a pontokat
      painter.drawLine(point.x() - 10, point.y()
         - 10, point.x() + 10, point.y() + 10);
      painter.drawLine(point.x() - 10, point.y()
         + 10, point.x() + 10, point.y() - 10);
   }
   …
   painter.drawEllipse(mouseLocation.x() - 30,
      mouseLocation.y() - 30, 60, 60);
      // kör kirajzolása
   …
```

Elemi grafika és egérkezelés

Kurzorkezelés

- Az egérkezelő műveletektől függetlenül is bármikor használhatjuk az egérpozíciót, 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, pl.:

```
widget.setCursor(QCursor(Qt::BusyCursor));
    // homokóra beállítása a vezérlőre
```

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

Elemi grafika és egérkezelés

Példa

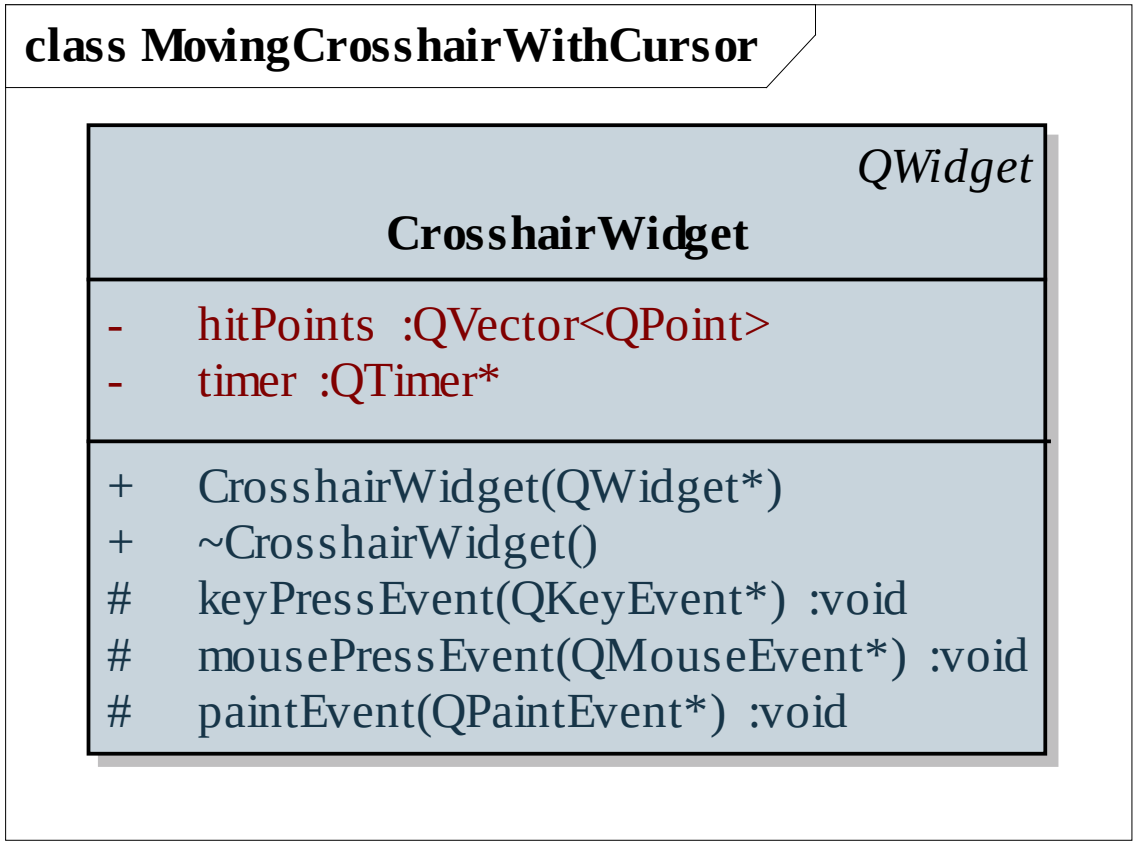
Feladat: Módosítsuk a célkereszt megjelenítő programunkat úgy, hogy a kurzorpozíció alapján jelenítse meg a célkeresztet, továbbá maga az egérkurzor is legyen egy célkereszt.

- 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, és mindig lekérjük a kurzorpozíciót a rajzolásnál
- az egér/billentyű lenyomás eseményét megtartjuk, ebben továbbra is felvesszük az új lövéseket

Elemi grafika és egérkezelés

Példa

Tervezés:



Elemi grafika és egérkezelés

Példa

Megvalósítás (`crosshairwidget.cpp`*):*

```
void CrosshairWidget::paintEvent(QPaintEvent *){
    …
    QPoint mouseLocation = QCursor::pos();
      // egérpozíció lekérdezése a képernyőn
    mouseLocation =
      QWidget::mapFromGlobal(mouseLocation);
      // egérpozíció transzformálása az ablakra
    …
```