Практика 2.

# Cобытия мыши. Tooltips. Начинаем рисовать. Класс QPainter. События изменения размеров окна. События таймера.

## Подготовка приложения

Для выполнения первой части этой лабораторной работы можно воспользоваться приложением, разработанным в лабораторной работе 1.

## События

### События мыши.

### Обработчик нажатия мыши

Для обработки события о нажатии кнопки мыши в нашем окне перегружаем виртуальный метод:

void QWidget::mousePressEvent ( [QMouseEvent](file:///D:/C++/QT/VORONEG/PRACTIKA%202/qmouseevent.html) \* *event* ) [virtual protected]

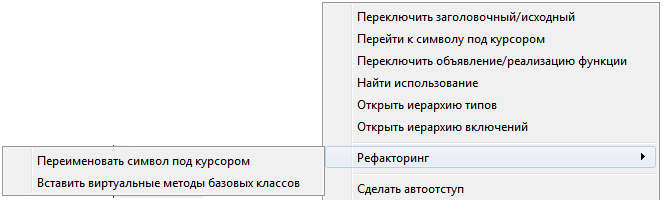
Все обработчики событий мыши принимают в качестве параметра указатель на объект [QMouseEvent](file:///D:\Marina\Qt\qmouseevent.html) (следует подключить заголовочный файл <QMouseEvent>), в котором запакована информация о событии.

|  |  |
| --- | --- |
| Информация | Как информацию получить (методы) |
| Какая из кнопок мыши нажата.  (Qt::NoButton, Qt::LeftButton, Qt::RightButton, Qt::MidButton…) | **Qt::MouseButton** QMouseEvent::**button**() const;  **Qt::MouseButtons** QMouseEvent::**buttons**() const; |
| Координаты курсора мыши относительно левого верхнего угла окна (клиентские координаты). | **int** QMouseEvent::**x**()const; **int** QMouseEvent::**y**()const; **QPoint** QMouseEvent::**pos** () const; |
| Координаты курсора мыши относительно левого верхнего угла экрана (экранные координаты). | **int** QMouseEvent::**globalX**()const; **int** QMouseEvent::**globalY**()const; **QPoint** QMouseEvent::**globalPos** () const; |
| Событие происходит при нажатой системной клавише | **Qt::KeyboardModifiers** QInputEvent::**modifiers** () const;  -устанавливает биты в соответствии с нажатыми системными клавишами. Для проверки возвращаемого значения используются именованные константы: Qt::ShiftModifier, Qt::ControlModifier, Qt::ControlModifier… |

Замечание:

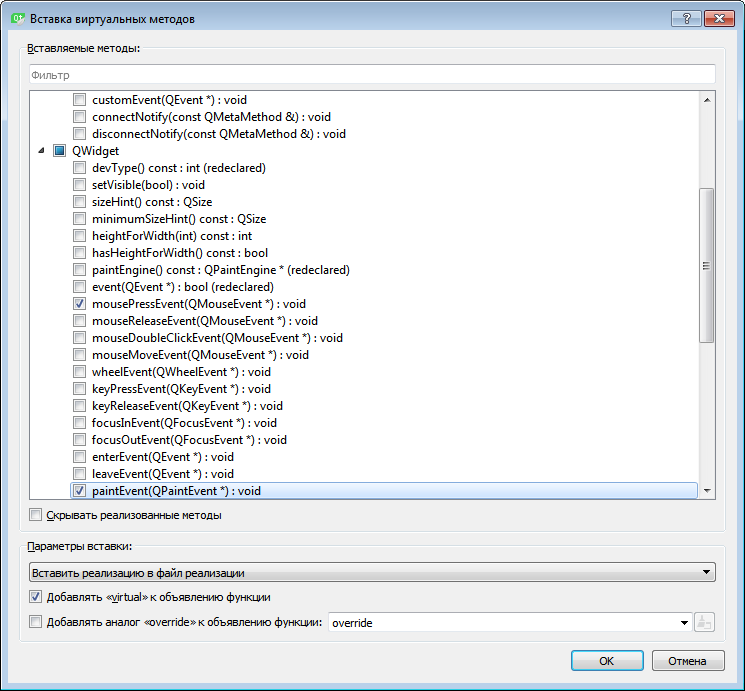
Для того, чтобы автоматически добавить объявления нужных виртуальных методов в класс виджета, можно выполнить следующую последовательность действий:

1. Перейдите в заголовочный файл с объявлением класса виджета
2. Установите курсор мыши на название класса виджета
3. Нажмите правую кнопку мыши для выбора контекстного меню
4. Выберите команду «Вставить виртуальные методы базовых классов»



**4**

1. Появится окно для выбора нужных виртуальных методов. Выберите те методы, которые Вам требуются.
2. Задайте параметры вставки
3. Нажмите «ОК»
4. В заголовочный файл будут вставлены объявления, а в файл реализации- определения методов



**5**

**4**

**4**

# Первая часть работы

**Задание 1.0**

Проверьте, что при **любом** нажатии на **любую** кнопку мыши вызывается Ваша заглушка. (Выведите в обработчике отладочную информацию.)

### При нажатии правой кнопки выводим текущие координаты курсора с помощью всплывающей подсказки –tooltip

**Задание 1.1**

В строке подсказки (tooltip) выведите текущие координаты курсора мыши, полученные при нажатии **правой** кнопки мыши (находящейся над виджетом).

Порядок действий:

2. Если нажата правая кнопка, из полученного параметра извлекаем координаты мыши.

Замечание:

* метод **pos()** возвращает относительные (клиентские) кординаты,
* метод **globalPos()** – абсолютные(экранные).

1. Формируем строку подсказки вида «x=<значение>, y=<значение>». В эту строчку логичнее выводить относительные (**клиентские**) координаты.  
   Подсказки:
   1. в новых версиях Qt не рекомендуется использовать QString::sprintf();
   2. 1 способ: для формирования результирующей строки для вывода можно воспользоваться классом QTextStream:  
      QString str;  
      QTextStream tstream(&str);  
      tstream<<"x="<<…
   3. 2 способ:

* для перевода целого значения в строковое представление можно пользоваться экземплярным методом :setNum()

**QString** & QString::**setNum**(**int** *n*, **int** *base* = 10);

или статическим методом number():

**QString** QString::**number**(**[uint](qtglobal.html" \l "uint-typedef)** *n*, **int** *base* = 10); [static]

QString strX=QString(). setNum(<координата x>)

QString strY=QString::number(<координата y>).

* для конкатенации строк можно использовать перегруженные операторы “+” и “+=”класса QString

Рекомендация: Попробуйте сформировать результирующую строку разными способами

1. Не забудьте #include<QToolTip>
2. Вывод подсказки осуществляется методами:

void QToolTip::showText ( const [QPoint](file:///C:/Users/user/AppData/Roaming/Microsoft/Word/qpoint.html) & *pos*, const [QString](file:///C:/Users/user/AppData/Roaming/Microsoft/Word/qstring.html) & *text*, [QWidget](file:///C:/Users/user/AppData/Roaming/Microsoft/Word/qwidget.html) \* *w=0* ) [static]  
и  
void QToolTip::showText ( const [QPoint](file:///C:/Users/user/AppData/Roaming/Microsoft/Word/qpoint.html) & *pos*, const [QString](file:///C:/Users/user/AppData/Roaming/Microsoft/Word/qstring.html) & *text*, [QWidget](file:///C:/Users/user/AppData/Roaming/Microsoft/Word/qwidget.html) \* *w*, const [QRect](file:///C:/Users/user/AppData/Roaming/Microsoft/Word/qrect.html)& *rect* ) [static]  
, где pos должна быть задана в **экранных** координатах

### А теперь будем выводить tooltip при нажатии правой кнопки мыши, если только нажата клавиша Ctrl

Подсказка: для проверки используем метод QInputEvent::modifiers()

**Qt::KeyboardModifiers** QInputEvent::**modifiers**() const;

**Задание 1.2**

Выведите в tooltip координаты мыши, полученные при нажатии правой кнопки мыши, только в том случае, если нажата клавиша-модификатор Ctrl (Qt::ControlModifier).

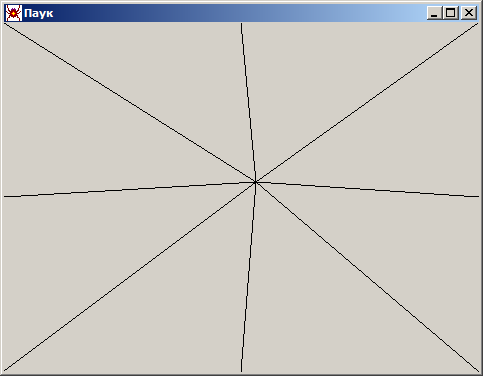
### Рисуем паутину при нажатой левой кнопке мыши

**Задание 2**

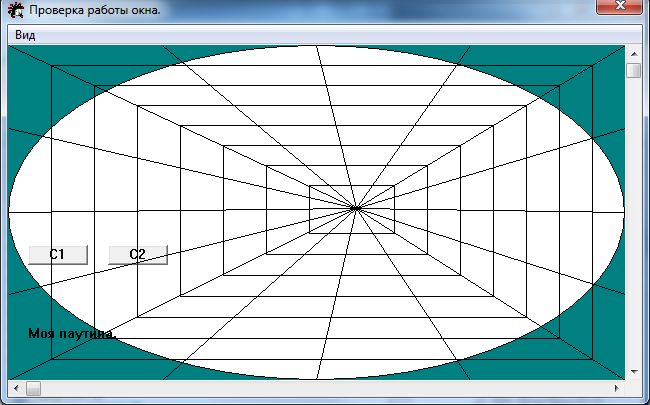
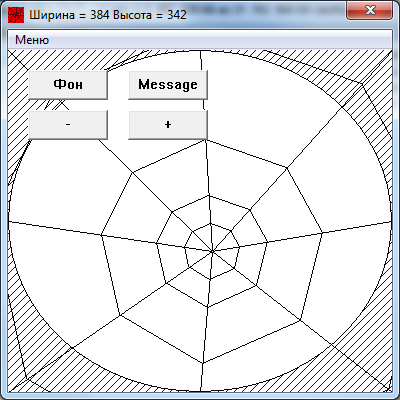
Нарисуйте на виджете паутину. Особенность паутины в том, что центр ее должен находиться в позиции курсора, а вторые концы отрезков паутины могут располагаться по краям прямоугольника рабочей области. Причем при изменении позиции курсора паутина должна перерисовываться.

Паутина отображается только при нажатой кнопке мыши. При отпускании кнопки мыши паутина исчезает.

Самая примитивная паутинка может выглядеть так:



Развивать эту тему можно бесконечно в зависимости от Ваших математических и эстетических возможностей.

Порядок действий:

1. Так как паутину нужно рисовать далеко не всегда (а только при нажатой левой кнопке мыши)=> нам нужен признак рисования => в классе Spider объявите булеву переменную, например, isDrawing. Как и где нужно проинициализировать эту переменную?
2. При нажатии левой кнопки мыши (в обработчике mousePressEvent()) флажок isDrawing нужно устанавливать
3. При отпускании левой кнопки мыши соответственно сбрасывать =>перегрузите виртуальный метод  
   void QWidget::mouseReleaseEvent([QMouseEvent](qthelp://org.qt-project.qtwidgets.502/qtgui/qmouseevent.html) \**event*) [virtual protected]
4. При перемещении мыши  
   void QWidget:: mouseMoveEvent ([QMouseEvent](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qmouseevent.html) \* *event* ) [virtual protected]  
   если требуется рисовать, вызывать перерисовку окна посредством метода  
   void QWidget::repaint () или QWidget::update().

Замечания:

* по умолчанию виртуальный метод mouseMoveEvent вызывается только в том случае, когда пользователь двигает мышь при **нажатой** кнопке.
* Собственно рисование паутины будет осуществляться в перегруженном обработчике события перерисовки окна - paintEvent()
* !!! Обратите внимание!!! Qt «захватывает мышь», если нажимается любая кнопка мыши. Это означает, что все события мыши (события о перемещении мыши) будут направляться окну (независимо от того, вышел курсор за пределы окна или нет). Мышь будет «захвачена» до того момента, пока не будет отжата последняя кнопка. И даже вид курсора меняться не будет!

**Важное замечание:**

### Метод

### [Qt::MouseButton](qthelp://org.qt-project.qtcore.551/qtcore/qt.html#MouseButton-enum) QMouseEvent::button() const – возвращает кнопку, вызвавшую событие. Для события mouseMoveEvent всегда возвращается значение [Qt::NoButton](qthelp://org.qt-project.qtcore.551/qtcore/qt.html#MouseButton-enum), поэтому не следует использовать его в обработчике события mouseMoveEvent.

## Начинаем рисовать

### Перегрузите виртуальный метод QWidget::paintEvent()

void QWidget::paintEvent ( [QPaintEvent](file:///C:\\Users\\user\\AppData\\Roaming\\Microsoft\\Word\\qpaintevent.html) \* *event* ) [virtual protected]

**Замечания**:  
1) Обработчик paintEvent() **никогда не вызывается явно**. Перерисовку может инициировать сама ОС (например, когда требуется восстановить окно, временно перекрытое другим окном…) или опосредованно программист может инициировать перерисовку, вызвав метод  
void QWidget::**repaint** () [slot] – вызывает немедленную перерисовку  
или void QWidget::**update** () [slot]– поставить событие о перерисовке в очередь => оптимизации =>может обработать несколько событий о перерисовке за один вызов paintEvent()

**2**) В самом методе paintEvent() не следует вызывать QWidget::repaint () или QWidget::update (), т.к. это приведет к рекурсии.

# Прежде, чем собственно что-либо нарисовать

Для рисования нужно подготовиться:

* создать объект, которому Вы будете делегировать все операции вывода.

Класс QPainter инкапсулирует системный контекст устройства и осуществляет вывод текста, векторной и растровой графики на любое стандартное устройство вывода (дисплей, принтер, плоттер…), а также содержит средства для масштабирования, задания системы координат, хранит инструменты для рисования…

#include<QPainter>

Замечание:

для подавления «мерцания» при перерисовке в Qt используется двойная буферизация (изображение формируется в памяти, а потом переносится на экран).

При создании объекта QPainter в конструктор в качестве параметра передается QPaintDevice\* - указатель на соответствующий контекст устройства (это объект, который содержит информацию о том, куда мы собираемся осуществлять вывод: в окно, на принтер…).

* Обычно объекты QPainter создаются (локально) в **обработчике события о перерисовке**. Там же происходит рисование.

# Рисование линий

Напоминание: рисовать стоит только в том случае, когда установлен флаг IsDrawing

* Создаем объект QPainter. В качестве параметра конструктора указываем наше окно, на котором осуществляется рисование ( вспомните, что QWidget является потомком QPaintDevice).
* Получаем размеры клиентской области  
  QRect QWidget::rect() const
* Получаем текущее положение курсора

1 способ

Можно завести в классе Spider переменную типа QPoint и фиксировать текущее положение курсора в обработчиках события QMouseEvent: mousePressEvent и mouseMoveEvent.

В качестве параметра в этих обработчиках используется QMouseEvent\* event. Можно получить текущуюю позицию курсора с помощью метода:

QPoint QMouseEvent::pos() const;

2 способ

А можно получить текущее положение курсора в событии QPaintEvent (**в координатах экрана**) посредством статического метода класса QCursor  
QPoint QCursor::pos() [static];

А рисовать будем **в клиентских координатах.** Для преобразования экранных координат в в оконные можно использовать:  
QPoint QWidget::**mapFromGlobal** (const QPoint &pos) const;

* Рисуем паутину, используя текущую точку курсора и размеры клиентской области
* Подсказка: для рисования отрезков можно использовать методы  
  QPainter::drawLine()  
  QPainter::drawPolyline()  
  или QPainter::drawLines()
* Подсказка: у класса QRect есть полезные методы для получения левого верхнего угла, правого нижнего (topLeft()), …
* Цвет паутины тоже можно задавать по желанию. Для этого можно настроить пере **QPen**.

## Курсор

По умолчанию курсор имеет форму стрелки. Изменить вид курсора можно посредством:  
void QWidget::**setCursor** ( const QCursor& );

восстановить умолчание – посредством:  
void QWidget::**unsetCursor** ();

Замечание:   
курсор принимает заданную форму только в пределах **клиентской** области того виджета, для которого был вызван метод. Это означает, что для нашего главного окна приложения в «системных» областях окна (меню, рамка…) курсор будет иметь форму стрелки.

Qt предоставляет некоторые наиболее используемые курсоры посредством перечисления enum **Qt::CursorShape**. Любую из приведенных в справке констант можно использовать в качестве параметра функции setCursor(). Примеры констант: Qt::CrossCursor, Qt::WaitCursor, Qt::PointingHandCursor,…

**Задание 3**

Установите в конструкторе какой-либо из стандартных курсоров на время работы программы, а

**на время рисования** изменяйте вид курсора (на паучка).

Подсказка:

Не очень хорошо формировать собственный курсор каждый раз, как только он потребуется при нажатии левой кнопки мыши или в событии рисования.

Лучше заготовить специфический курсор на все время работы программы. Для этого можно

в классе Spider завести переменную типа QCursor или QCursor\*, например  
QCursor\* spiCursor;  
а в конструкторе виджета сформировать курсор посредством картинки.

## Не выпускаем«паука» за пределы окна во время рисования

Попробуйте «потянуть» паутину за пределы окна. Так как Qt автоматически «захватывает» мышь на время нажатия любой ее кнопки, все события мыши поступают виджету, даже если курсор вышел за его пределы. Такой эффект не всегда соответствует логике приложения, поэтому два варианта решения:

В обработчике  
void Spider::mouseMoveEvent(QMouseEvent\*event)  
можно проверить, что мышь вышла за пределы окна с помощью метода  
bool QRect::**contains** ( const [QPoint](file:///C:/Users/user/AppData/Roaming/Microsoft/Word/qpoint.html) & *point*, bool *proper* = false ) const

1. способ

* можно вернуть оригинальный курсор, отказавшись от курсора-паучка. Для этого можно использовать void QWidget::**unsetCursor** ();
* После того, как курсор «вернется» в клиентскую область, он должен снова «превратиться» в паучка.

1. способ

А можно «возвращать» курсор в пределы виджета посредством  
void QCursor::setPos(int x,int y)[static] или  
void QCursor::setPos(const QPoint &p)[static]

Замечание 1: метод QCursor::setPos () ожидает позицию в **глобальных** координатах. Для преобразования клиентских координат в экранные можно воспользоваться методом **mapToGlobal**()

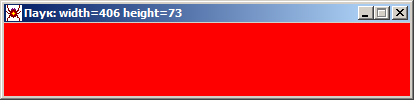
QPoint QWidget::mapToGlobal (const QPoint &pos) const;

Замечание 2: в ОС Windows есть системная Win32 API функция ClipCursor(), которая «не выпускает» курсор за пределы окна, но в Qt ее аналога нет.

**Задание 4**

Отслеживайте положение курсора относительно виджета и не позволяйте паучку «покидать» пределы рабочей области виджета.

## Отслеживаем изменения размеров окна



Чтобы реагировать на изменение размеров окна – перегружаем виртуальный обработчик:

voidQWidget::**resizeEvent**([QResizeEvent](qthelp://org.qt-project.qtwidgets.532/qtgui/qresizeevent.html) \**event*) [virtual protected]

**Задание 5**

При изменении размера требуется формировать в заголовке окна строку, содержащую информацию о текущей ширине и высоте виджета.

Было бы неплохо, если приложение также, как и лаб1, было локализовано.

# Вторая часть работы

## Рисование прямоугольника с использованием мыши.

Для выполнения этого задания создайте каркас приложения так, как описано в лабораторной работе 1. Создаем НОВОЕ приложение.

**Задание 6:**

Нарисуйте на виджете прямоугольник следующим образом:

1. при нажатии **правой** кнопки мыши определяются координаты одной из вершин.
2. При перемещении мыши (при нажатой кнопке) определяются координаты противоположной вершины и выполняется отрисовка текущего прямоугольника.
3. Рисование прямоугольника завершается, когда кнопка мыши отжимается.

A picture containing icon

Description automatically generated

1. Рисование начинается при нажатии кнопки мыши над виджетом.
2. При нажатии правой кнопки мыши (событие *mousePressEvent*(QMouseEvent \*)):
   1. устанавливается признак «рисуем прямоугольник»: *isDrawing=true*;
   2. фиксируется одна из вершин прямоугольника startPoint
   3. координаты противоположной вершины curPoint первоначально имеют то же значение.
3. при перемещении (событие *mouseMoveEvent*(QMouseEvent \*)) мыши (**только** если происходит рисование):
   1. получаем **текущую** координату курсора – curPoint
   2. Сформируйте прямоугольник **QRect** по заданным координатам вершин.
   3. запомните свойства этого прямоугольника в классе виджета в переменной m\_currentRect.
   4. выполняем перерисовку виджета
   5. При **отпускании** мыши (**только** если происходит рисование) – «рисование текущего прямоугольника закончено нужно сбросить признак «рисуем прямоугольник»

Замечание: Вы наверняка будете огорчены, что нарисованный Вами прямоугольник исчезнет при попытке нарисовать с помощью мыши следующий прямоугольник. Причину этого явления, надеюсь, Вы сможете установить. Чтобы побороть этот эффект можно выполнить задание 1 из раздела дополнительных задач.

## Выделение прямоугольника с помощью мыши

Нарисованный с помощью **правой** кнопки мыши прямоугольник сохранен в переменной m\_currentRect. Следующей задачей будет научиться выделять этот прямоугольник с помощью **левой** кнопки мыши.

Shape

Description automatically generated

Выделять прямоугольник надо будет только в том случае, если нажата **левая** кнопка мыши и курсор мыши находится над прямоугольником, который хотим выделить. Если координаты курсора не попадают в прямоугольник, то выделение не происходит.

* + 1. Нам потребуется признак того, что прямоугольник надо выделить. В классе Spider объявите булеву переменную, например, isSelected. Не забудьте ее проинициализировать.
    2. При перерисовке прямоугольника можно использовать не сплошную, а штриховую линию. Этого можно добиться, задав нужный стиль пера.

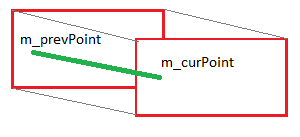
**enum Qt::PenStyle**

|  |  |
| --- | --- |
| **Constant** | **Value** |
| Qt::NoPen | 0 |
| Qt::SolidLine | 1 |
| Qt::DashLine | 2 |
| Qt::DotLine | 3 |
| Qt::DashDotLine | 4 |
| Qt::DashDotDotLine | 5 |
| Qt::CustomDashLine | 6 |

## Перетаскивание выделенного прямоугольника с помощью мыши

Перетаскивание выделенного прямоугольника выполняется при нажатой **левой** кнопке мыши.

При перетаскивании по сути осуществляется «параллельный перенос» всех точек фигуры. При изменении положения курсора мыши «синхронно» должны изменяться и координаты прямоугольника. Поскольку перемещение должно осуществляться от одной точки до другой, пока зажата левая кнопка мыши, нам потребуется кроме текущих координат мыши иметь в своем распоряжении значения координат «предыдущей точки».



# Третья часть работы (необязательная)

## Дополнительные задания

В эту часть работы включены необязательные задания, но они могут серъезно поднять Ваш профессиональный уровень.

Задание 1 предназначено для решения задачи сохранения и отображения **всех** пямоугольников, которые Вы рисуете на экране с помощью мыши. При этом используется контейнер для уже нарисованных фигур.

В задание 2 подключается обработка таймера, который используется для анимации движения паучка. А во второй части задания для сохранения траектории движения паучка можно применить контейнер, как в дополнительном задании 1. Но можно использовать иной подход. А именно, рисовать траекторию движения «в памяти», а пото ее выводить

Задание 3 для тех, кто хочет попробовать поработать над темой генерации и обработки пользовательских событий. Это умение может пригодиться при разработке многопоточных приложений с использованием библиотеки Qt.

## Сохранение всех прямоугольников на экране

**Задание 1.1:**

Попытайтесь доработать программу таким образом, чтобы отображались все нарисованные прямоугольник, а не только самый последний.

Подсказка:

Для того, чтобы отображались ВСЕ нарисованные прямоугольники можно:

* создать контейнер для хранения прямоугольников
* при завершении формирования прямоугольника сохранять его в контейнере
* в событии paintEvent выполнить рисование всех сохраненных прямоугольников.

Icon

Description automatically generated

**Задание 1.2:**

Попытайтесь добавить возможность выделения одного из нескольких прямоугольников с помощью левой кнопки мыши.

А также возможность перетаскивания выделенного прямоугольника.

Icon

Description automatically generated

## Таймер

Встроенный таймер:

* В каждом классе, производном от QObject, есть возможность обрабатывать событие таймера.
* Для запуска таймера используется метод : int QObject::**startTimer** (int interval);

где interval - интервал запуска в миллисекундах,

метод возвращает идентификатор, необходимый для распознавания таймеров, используемых в объекте.

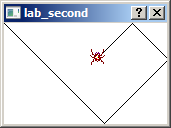
* Для остановки таймера используется метод: void QObject::**killTimer** (int id);

По истечении установленного интервала генерируется событие **QTimerEvent**,

* Событие таймера можно обработать с помощью перегруженного метода

void QObject::timerEvent ( QTimerEvent \* pEvent);

* Внутри обработчика можно узнать идентификатор таймера, инициировавшего это событие. Для этого предназначен метод int QTimerEvent::**timerId**() const; позволяет



**Задание 2.1:**

Реализуйте передвижение паучка по окну.

Начинать и останавливать движение можно по событию нажатия клавиши мыши.

Подсказки:

* При обработке события таймера паучек «передвигается» в определенном направлении на заданные значения перемещения.
* В том случае, когда паучек достигает границы, он должен «отталкиваться» от стенки и менять направление движения.

**Задание 2.2:**

Попробуйте отображать траекторию движения паучка.

Подсказка:

см. подсказку к дополнительному заданию 1.

Или можно применить иной подход:

1. В конструкторе виджета создать QPixmap в памяти: *QPixmap(int width, int height);*
2. выполнять рисование каждого отрезка пути на QPixmap;
3. в событии paintEvent выполнить рисование объекта типа QPixmap на виджете.