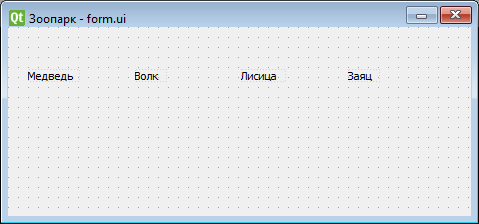
Глава 11

Перетаскивание (drag & drop):  
проект ZOO

Проект ZOO знакомит с различными аспектами режима перетаскивания drag & drop (запуск режима drag & drop и различные варианты его завершения, действия при перетаскивании на недопустимый приемник, дополнительное выделение источника и приемника, настройка вида курсора в режиме пе­ретаскивания).

11.1. Перетаскивание меток по форме

После создания формы типа QWidget разместите в ней четыре метки (label – label\_4) и настройте свойства формы и добавленных меток (листинг 11.1). Положение меток настройте в соответствии с рис. 11.1. Добавьте в текстовом редакторе в форму кастомный класс DragDropLabel (наследник QLabel). Создайте класс DragDropLabel, определите в нем событие mousePressEvent (листинг 11.2). Кроме того, опреде­лите обработчики событий dragEnterEvent и dropEvent класса Form (листинг 11.3). В конструктор формы добавьте оператор фиксации размера.



**Рис. 11.1.** Вид формы Form для проекта ZOO на начальном этапе разработки

**Листинг 11.1. Настройка свойств формы Form и ее компонентов**

Form: windowTitle = **Зоопарк**, acceptDrops = **True**

label: text = **Медведь**

label\_2: text = **Волк**

label\_3: text = **Лиса**

label\_4: text = **Заяц**

**Листинг 11.2. События mousePressEvent**

**void DragDropLabel::mousePressEvent(QMouseEvent \*event)**

**{**

**if (event->buttons() == Qt::LeftButton)**

**{**

**QDrag \*drag = new QDrag(this);**

**drag->setMimeData(new QMimeData());**

**drag->exec(Qt::MoveAction);**

**}**

**}**

**Листинг 11.3. События dragEnterEvent и dropEvent**

**void Form::dragEnterEvent(QDragEnterEvent \*event)**

**{**

**event->accept();**

**}**

**void Form::dropEvent(QDropEvent \*event)**

**{**

**DragDropLabel \*src = (DragDropLabel\*)event->source();**

**src->move(event->pos());**

**}**

**Результат.** Метки с названиями зверей можно перетаскивать с помощью левой кнопки мыши. При перетаскивании метки-*источника* (source) она остается на месте, однако вид курсора мыши изменя­ется, что является признаком режима перетаскивания. В качестве *приемника* (target) пока определена лишь сама форма: при отпускании над ней кнопки мыши происходит перемещение метки зверя на указанную позицию (см. комментарии 1 и 2).

***ПРИМЕЧАНИЕ***

В данном проекте в текстах обработчиков, связанных с перетаскиванием, будем использовать имена src и trg для объектов-источников и объектов-приемников соответственно. Для переопределенных классов вместо trg будет использоваться this.

**Недочет.** Разрешается перетащить метку не только на свободную часть формы, но и на другую метку; при таком перетаскивании происходит наложение двух меток.

**Исправление.** Определите обработчик события dragMoveEvent класса Form (листинг 11.4).

**Листинг 11.4. Событие dragMoveEvent**

**void Form::dragMoveEvent(QDragMoveEvent \*event)**

**{**

**auto trg = (DragDropLabel\*)this->childAt(event->pos());**

**if (!trg)**

**event->accept();**

**else**

**event->ignore();**

**}**

**Результат.** В начальный момент перетаскивания курсор мыши остается разрешающим. Перетаскива­ние одной метки на другую по-прежнему запрещено (см. комментарий 3).

Комментарии

1. Для активизации режима перетаскивания класс QDrag имеет метод exec, параметр которого типа Qt.DropActions указывает разрешенный результат (эффект) успешного перетаскивания. Преду­смотрены пять основных эффекта: CopyAction, MoveAction, LinkAction, TargetMoveAction и IgnoreAction (их названия связаны с вариантами действий при перетаскивании). Если в ходе пе­ретаскивания может возникнуть один из нескольких эффектов (например, CopyAction или MoveAction), то нужно укзать все эти эффекты, объединяя их операцией | (побитовое ИЛИ), напри­мер, Qt::CopyAction | Qt::MoveAction. Во втором варианте вызова функции exec существует параметр – действие по умолчанию, которое будет выполняться при указании нескольких эффек­тов автоматически.

На самом деле, указывать действие MoveAction было необязательно, так как в первом варианте вызова exec\_ это действие указано по умолчанию. Однако для наглядности мы решили дополни­тельно указать, что выполняется именно перемещение.

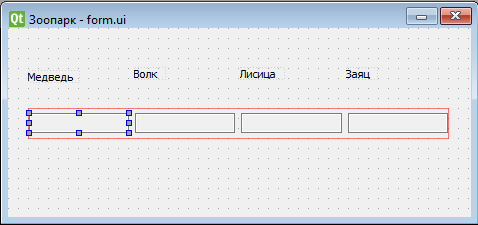
Перед запуском процесса перетаскивания объекту класса QDrag нужно указать объект mimeData. Данное поле хранит информацию о типе MIME (специальный стандарт Multipurpose Internet Mail Extensions) передаваемых данных. Использование данного стандарта обеспечивает безопасность передачи информации между приложениями и копирования внутри одного приложения. Типами MIME может быть текстовая информация (text/plain), html-страница (text/html) или, например, изображение (image/\*). В нашем случае достаточно лишь знать положение курсора (а эта инфор­мация может быть получена из событий), поэтому используется конструктор по умолчанию класса QMimeData.

1. Для перемещения метки на новое место необходимо знать позицию курсора мыши при завершении режима перетаскивания. Данная позиция содержится в свойстве pos для параметра event.
2. Установив для формы значение свойства acceptDrops равным true, мы сделали форму допусти­мым приемником. Поскольку для меток свойство acceptDrops осталось равным False (значение по умолчанию), метки не считаются допустимыми приемниками, т. е. они как бы "невидимы" в режиме перетаскивания. Поэтому, когда курсор мыши в режиме перетаскивания проходил над не­видимыми для него компонентами-метками, он считал, что находится над формой, и адресовал события dragEnterEvent и dragDropEvent именно форме. Если в этот момент источник отпус­кался, то он накладывался на расположенную под ним метку (недочет). Для исправления отмечен­ного недочета был определен обработчик события dragMoveEvent. В нем выполняется следующая проверка: если в текущей точке дочерних элементов формы не найдено (используется метод childAt, единственный параметр которого – координаты точки), то событие перемещения выпол­няется, иначе игнорируется (курсор будет иметь вид запрещающего знака).

11.2. Перетаскивание меток в поля ввода

Разместите в форме Form горизонтальный менеджер компоновки, расположите в нем четыре поля ввода типа QLineEdit (lineEdit – lineEdit\_4) и настройте позицию лейаута в соответствии с рис. 11.2. Для каждого поля ввода положите свойства readOnly равным **True** (данная настройка запрещает пользователю редактировать текст в поле ввода, однако не влияет на возможность программного из­менения текста) и styleSheet равным **background-color: #f0f0f0**.

Измените для полей ввода класс на DragDropLineEdit и определите его события dragEnterEvent и dropEvent (листинг 11.5).



**Рис. 11.2.** Вид формы Form для проекта ZOO на промежуточном этапе разработки

**Листинг 11.5. События dragEnterEvent и dropEvent класса DragDropLineEdit**

**void DragDropLineEdit::dragEnterEvent(QDragEnterEvent \*event)**

**{**

**if (text() == "")**

**event->accept();**

**else**

**event->ignore();**

**}**

**void DragDropLineEdit::dropEvent(QDropEvent \*event)**

**{**

**DragDropLabel \*src = (DragDropLabel\*)event->source();**

**setText(src->text());**

**src->setVisible(false);**

**}**

**Результат.** Теперь приемником может также служить любое *незаполненное* поле ввода ("пустая клетка"). При перетаскивании метки на незаполненноеполе ввода "зверь попадает в клетку" (текст метки отображается в поле ввода).Перетаскивание метки на уже заполненное поле ввода пока запре­щено, хотяв следующем разделе это действие станет доступным.

11.3. Взаимодействие меток при их перетаскивании друг на друга

Добавьте в конструктор формы Form следующие операторы:

**self.label.tag = 3**

**self.label\_2.tag = 2**

**self.label\_3.tag = 1**

**self.label\_4.tag = 0**

В классы DragDropLabel и DragDropLineEdit добавьте приватное описание поля tag: **int tag;** и функции для чтения и записи данного поля:

**void setTag(const int& tag) { \_tag = tag; }**

**int tag() { return \_tag; }**

Измените методы dragEnterEvent и dropEvent класса DragDropLineEdit (листинг 11.6). Определите методы dragEnterEvent и dropEvent для класса DragDropLabel (листинг 11.7). После выполнения этого действия метод dragMoveEvent класса Form больше не требуется и может быть удален.

**Листинг 11.6. Новый вариант событий dragEnterEvent и dropEvent класса DragDropLineEdit**

**void DragDropLineEdit::dragEnterEvent(QDragEnterEvent \*event)**

**{**

**event->accept();**

**}**

**void DragDropLineEdit::dropEvent(QDropEvent \*event)**

**{**

**DragDropLabel \*src = (DragDropLabel\*)event->source();**

**if (src->tag() >= tag())**

**{**

**setText(src->text());**

**setTag(src->tag());**

**}**

**lb->setVisible(false);**

**}**

**Листинг 11.7. Cобытия dragEnterEvent и dropEvent класса DragDropLabel**

**void DragDropLabel::dragEnterEvent(QDragEnterEvent \*event)**

**{**

**event->accept();**

**}**

**void DragDropLabel::dropEvent(QDropEvent \*event)**

**{**

**DragDropLabel \*lb = (DragDropLabel\*)event->source();**

**if (this == lb)**

**return;**

**if (lb->tag() > tag())**

**{**

**lb->move(pos());**

**setVisible(false);**

**}**

**else**

**lb->setVisible(false);**

**}**

**Результат.** При перетаскивании названия одного зверя на название другого более сильный "поедает" более слабого. То же самое происходит, если один из зверей перетаскивается в клетку, уже занятую другим зверем.

**Ошибка.** Если при перетаскивании метки отпустить ее над ней самой, то метка исчезнет. Таким образом, зверь съест самого себя.

**Исправление.** В методе **dropEvent** (листинг 11.7) перед оператором

if (src->tag() > tag())

вставьте

**if (this == src)**

**return**

Комментарии

1. С помощью свойства tag определяется относительная сила зверей. При помещении зверя в клетку информация о силе зверя сохраняется в свойстве tag клетки (т. е. компонента DragDropLineEdit). Для того чтобы сохранить возможность помещения зверя в пустую клетку, в начале программы свойства tag всех полей ввода полагаются равными 0.
2. При определеии нового значения для src в методе dropEvent можно было бы использовать свойство pos параметра event, как в методе dropEvent формы Form (листинг 11.3), однако проще воспользоваться свойством pos приемника. Кроме того, подобный способ позволяет поместить метку-источник в точности на место метки-приемника, независимо от того, в какой точке метки-приемника была отпущена кнопка мыши.

11.4. Действия в случае перетаскивания на недопустимый приемник

Измените метод mousePressEvent класса DragDropLabel (листинг 11.8).

**Листинг 11.8. Cобытие mousePressEvent класса DragDropLabel**

void DragDropLabel::mousePressEvent(QMouseEvent \*event)

{

if (event->buttons() == Qt::LeftButton)

{

QDrag \*drag = new QDrag(this);

drag->setMimeData(new QMimeData());

drag->exec(Qt::MoveAction);

**QRect rect = ((QWidget\*)parent())->rect();**

**QPoint pos = ((QWidget\*)parent())->mapFromGlobal(QCursor::pos());**

**if (!rect.contains(pos))**

**setVisible(false);**

}

}

**Результат.** Если перетаскивание метки-зверя завершается за пределами формы (в этом случае курсор имеет вид запрещающего знака), то зверь "убегает" из зоопарка и его метка исчезает.

Комментарий

В измененном методе вычисляется рабочий прямоугольник формы (родительского виджета для метки) и текущие координаты курсора (параметр event->pos() не подходит, так как возвращает координаты курсора в начале перетаскивания). Если прямоугольник не содержит такой позиции, то свойство isVisible метки меняется на false (животное "сбегает" из зоопарка).

11.5. Дополнительное выделение источника и приемника в ходе перетаскивания

Измените методы mousePressEvent, dragEnterEvent и определите метод dragLeaveEvent класса DragDropLabel (листинг 11.9). Измените методы dragEnterEvent, dropEvent и определите метод dragLeaveEvent класса DragDropLineEdit (листинг 11.10).

**Листинг 11.9. Методы класса DragDropLabel**

void DragDropLabel::*mousePressEvent*(QMouseEvent \*event)

{

if (event->buttons() == Qt::LeftButton)

{

**setStyleSheet("color: blue;");**

QDrag \*drag = new QDrag(this);

drag->setMimeData(new QMimeData());

drag->exec(Qt::MoveAction);

QRect rect = ((QWidget\*)parent())->rect();

QPoint pos = ((QWidget\*)parent())->mapFromGlobal(QCursor::pos());

if (!rect.contains(pos))

*setVisible*(false);

**setStyleSheet("color: black;");**

}

}

void DragDropLabel::*dragEnterEvent*(QDragEnterEvent \*event)

{

event->accept();

**setStyleSheet(styleSheet().append("background-color: yellow;"));**

}

void DragDropLabel::*dragLeaveEvent*(QDragLeaveEvent\*)

{

**setStyleSheet(styleSheet().append("background-color: #f0f0f0;"));**

}

**Листинг 11.10. Методы класса DragDropLineEdit**

void DragDropLineEdit::*dragEnterEvent*(QDragEnterEvent \*event)

{

event->accept();

**setStyleSheet("background-color: yellow;");**

}

void DragDropLineEdit::*dropEvent*(QDropEvent \*event)

{

DragDropLabel \*src = (DragDropLabel\*)event->source();

if (src->tag() >= tag())

{

setText(src->text());

setTag(src->tag());

}

src->*setVisible*(false);

**setStyleSheet(styleSheet().append("background-color: #f0f0f0;"));**

}

void DragDropLineEdit::*dragLeaveEvent*(QDragLeaveEvent\*)

{

**setStyleSheet("background-color: #f0f0f0;");**

}

**Результат.** В режиме перетаскивания цвет текста метки-источника изменяется на синий, а текущий компонент-приемник (метка или поле ввода) изображается на желтом фоне (см. комментарий 1).

Комментарий

Для дополнительного выделения источника перетаскивания достаточно настроить соответствующим образом его свойства до вызова метода exec, а после завершения этого метода (т. е. после выхода из режима перетаскивания) восстановить измененные свойства. Для выделения текущего приемника следует изменять его свойства в обработчике события dragEnterEvent, а восстанавливать — в обработчике события dragLeaveEvent, которое возникает в тот момент, когда курсор мыши покидает текущий приемник. Заметим, что в случае недопустимого приемника событие dragLeaveEvent возникает также в ситуации, когда перетаскивание завершается над подобным приемником. Если же перетаскивание завершается над допустимым приемником, то событие dragLeaveEvent не возникает, и для восстановления свойств приемника надо использовать обработчик события dropEvent.

Объединить обработчики в один (и для текстового поля, и для метки), несмотря на их похожесть, все же нельзя. Это связано с тем, что при изменении таблицы стилей она переписывается: такое поведение может привести к тому, что смена фонового цвета восстановит цвет текста по умолчанию. Поэтому в случае метки выполняется конкатенация таблиц стилей (в таком случае для каждого свойства будет использоваться последнее значение).

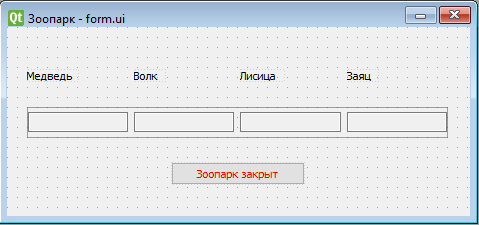
11.6. Информация о текущем состоянии программы. Кнопки с изображениями

Разместите в форме Form кнопку pushButton, в качестве ее свойств text и styleSheet укажите **Зоопарк закрыт** и **background-color: red** соответственно, а также настройте положение кнопки в соответствии с рис. 11.3.

Для большей наглядности добавим слева от надписи кнопки небольшое изображение, вид которого (как и текст кнопки) будет зависеть от текущего состояния программы. Добавьте в конструктор формы оператор

**ui->pushButton->setIcon(style()->standardIcon(QStyle::SP\_MessageBoxCritical));**

Добавьте сигнал buttonChanged и измените метод mousePressEvent в классе DragDropLabel, добавив к нему новые операторы (листинг 11.11). Соедините сигнал buttonChanged для каждой из четырех меток label-label\_4 в конструкторе формы Form со слотом setButton (листинг 11.12).



**Рис. 11.3.** Окончательный вид формы Form для проекта ZOO

**Листинг 11.11. Добавление к методу mousePressEvent класса DragDropLabel**

**QString s = "";**

**for (auto child = parent()->children().begin()**

**child < parent()->children().begin() + 4; child++)**

**{**

**if (((DragDropLabel\*)(\*child))->isVisible())**

**return;**

**s += ((DragDropLabel\*)(\*child))->text();**

**}**

**if (s == "")**

**return;**

**emit buttonChanged();**

**Листинг 11.12. Слот setButton**

**void Form::setButton()**

**{**

**ui->pushButton->setText("Зоопарк открыт");**

**ui->pushButton->setStyleSheet("color: green;");**

**ui->pushButton->setIcon(style()->*standardIcon*(QStyle::SP\_DialogApplyButton));**

**}**

**Результат.** Если все метки исчезли и при этом хотя бы одно поле ввода оказалось заполненным, то на кнопке выводится текст **Зоопарк открыт**, а цвет оформления кнопки меняется с красного на зеленый.

***ПРИМЕЧАНИЕ***

Для перебора всех меток в новом варианте метода mousePressEvent использовалось свойство-коллекция свойство-коллекция children. В данном случае мы пользуемся тем, что метки были первыми четырьмя дочерними элементами.

**Недочет.** Перетаскивание метки на кнопку pushButton приводит к тому, что метка "проваливается" под кнопку и становится недоступной для последующего перетаскивания, поскольку ее не удается зацепить мышью.

**Исправление.** Положите свойство acceptDrops кнопки pushButton равным True. Добавьте в текстовом редакторе кастомный класс DragDropPushButton (наследник QPushButton), измените тип pushButton на DragDropPushButton и определите в данном классе обработчик события dragEnterEvent (листинг 11.13).

**Листинг 11.13. Обработчик события dragEnterEvent класса DragDropPushButton**

**void DragDropPushButton::*dragEnterEvent*(QDragEnterEvent \*event)**

**{**

**((Form\*)parent())->*dragEnterEvent*(event);**

**}**

**Результат.** Теперь при попытке перетаскивания метки на кнопку pushButton ничего не происходит: метка остается на прежнем месте (см. комментарий).

Комментарий

Отмеченный недочет обусловлен тем, что по умолчанию свойство acceptDrops кнопки равно **False**, и поэтому кнопка невидима для режима перетаскивания. Заметим, что простого изменения значения свойства acceptDrops на **True** недостаточно: в этом случае кнопка будет вести себя как *недопустимый* приемник, и поэтому метка, отпущенная над кнопкой, будет исчезать с формы, поскольку именно такое действие выполняется в нашей программе при попытке отпустить источник над недопустимым приемником (см. разд. 11.4). Связывание события dragEnterEvent кнопки с обработчиком dragEnterEvent формы Form решает проблему, так как в этом случае кнопка становится допустимым приемником с эффектом MoveAction, хотя при отпускании над ней источника она ничего не делает (последнее обстоятельство связано с тем, что для кнопки не определен обработчик события dropEvent).

11.7. Восстановление исходного состояния

Определите метод reload и слот on\_pushButton\_clicked формы Form (листинг 11.14).

**Листинг 11.14. Методы reload и on\_pushButton\_clicked**

**void Form::reload()**

**{**

**for (auto child = children().begin(); child < children().begin() + 4; child++)**

**{**

**std::string s = (\*child)->objectName().toStdString().substr(5);**

**auto \*c = ui->horizontalLayoutWidget->findChild<DragDropLineEdit \*>**

**(QString::fromStdString("lineEdit" + s));**

**QPoint pos = ui->horizontalLayoutWidget->mapToParent(c->pos());**

**((DragDropLabel\*)(\*child))->move(pos.x(), pos.y() / 2);**

**}**

**ui->pushButton->setFocus();**

**}**

void Form::on\_pushButton\_clicked()

{

**reload();**

**for (auto child = children().begin(); child < children().begin() + 4; child++)**

**{**

**((DragDropLabel\*)(\*child))->*setVisible*(true);**

**std::string s = (\*child)->objectName().toStdString().substr(5);**

**auto \*c = ui->horizontalLayoutWidget->findChild<DragDropLineEdit\*>**

**(QString::fromStdString("lineEdit" + s));**

**c->setText("");**

**c->setTag(0);**

**}**

**ui->pushButton->setText("Зоопарк закрыт");**

**ui->pushButton->setStyleSheet("color: red;");**

**ui->pushButton->setIcon(**

**style()->*standardIcon*(QStyle::SP\_MessageBoxCritical));**

}

**Результат.** Исходное положение меток-зверей теперь определяется программно, а именно в методе формы reload (метки располагаются над соответствующими полями ввода и выравниваются по их левой границе). В дальнейшем при нажатии кнопки pushButton исходное положение зверей восстанавливается, а клетки освобождаются.