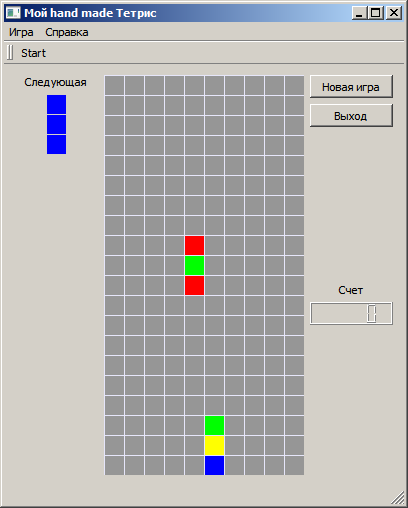
**Практика 5. Тетрис**

# Продолжаем использовать дизайнер. Продолжаем использовать класс QAction. Система динамических свойствQt. Класс QLCDNumber. Асинхронное соединение сигнал/слот. Класс QVector. События клавиатуры. Таймер.

## Задача

Создадим простейшее приложение-игру «Тетрис». Разновидностей тетрисов в мире существует много. Я предлагаю самый простой с точки зрения тех алгоритмов, которые Вам придется реализовать:

* В стакан «падают» вертикальные «тройки» разноцветных квадратиков
* Двигать столбики можно с помощью клавиатуры влево-вправо
* «Перемещать» квадратики в фигурке можно с помощью клавиатуры вниз/вверх
* «Ронять» столбик можно с помощью пробела
* «Сбрасываются» (рекурсивно) вертикальные и горизонтальные совокупности трех и более квадратиков одинакового цвета



## Шаблон приложения

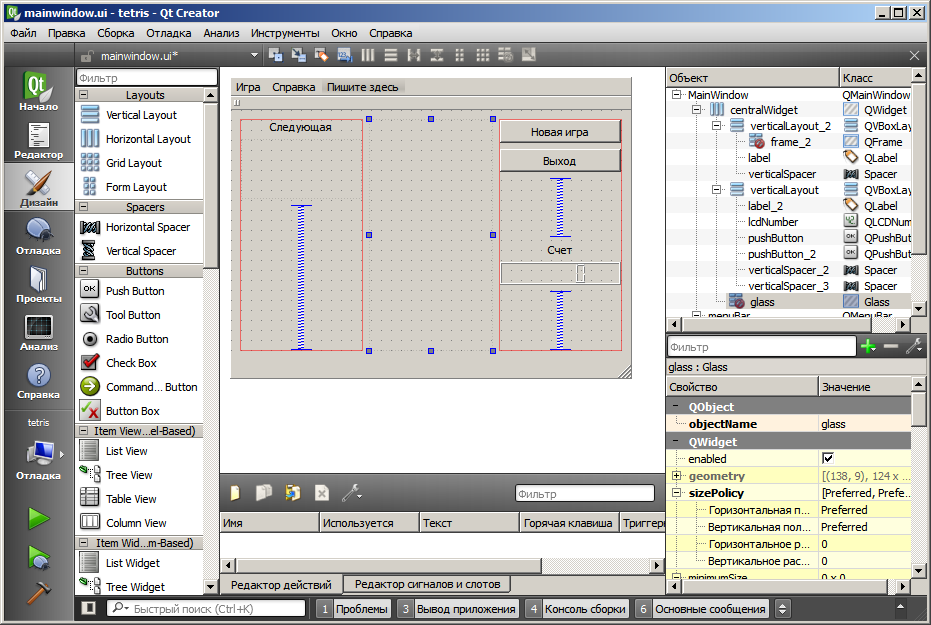
С помощью QtCreator-а создайте каркас приложения на базе предоставляемой дизайнером формы. Начинаем наполнять заготовку содержимым.

## Пользовательский интерфейс

Формируем с помощью дизайнера пользовательский интерфейс:

* Меню, в котором несколько выпадающих меню («Файл», «Свойства», «Справка»…)
* В каждом выпадающем меню – соответствующие пункты (“Начать новую игру”,…)
* Toolbar – на котором кнопками дублируются некоторыепункты меню
* Новую игру можно запускать посредством:
  + выбора пункта меню
  + кнопки на tool bar-е
  + с помощью pushButton
* собственно «стакан» - пользовательский виджет (Widget)
* элемент управления(QLCDNumber), в котором отображается текущий счет (возможно, следует сопроводить пояснительным текстом- label)
* пользовательский виджет (Frame), в котором отображается следующая фигурка
* другие элементы управления

В зависимости от Ваших предпочтений и вкуса (и основываясь на уже полученных и усвоенных знаниях) должно получиться что-то вроде:



В моем случае клиентская область окна (centralwidget)представляет собой горизонтальную компоновку из трех колонок, а крайние колонки в свою очередь представляют собой вертикальные компоновки.

Замечание (важное!)

* + у «стакана» можно в свойствах установить **FocusPolicy**–**StrongFocus**. Это означает, что «стакан» будет получать фокус ввода не только при нажатии клавиш на клавиатуре, но и при перемещении по виджетам с помощью табуляции
  + для удобства рекомендую давать создаваемым с помощью дизайнера виджетам (не всем подряд, а только тем, к которым возможно придется обращаться в коде) осмысленные имена (например, pusButtonNextGame), иначе дизайнер именует их в соответствии с умолчанием: pusButton, pusButton\_2…
  + Следует учесть, что при создании связей между элементами пользовательского интерфейса одно и то же действие может быть инициировано: при выборе пункта меню, нажатии кнопки на toolbar-е + возможно, Вы предусмотрите акселератор… Поэтому полезно вспомнить про действия Qt – класс QAction

## Класс «стакан»

Пользовательские классы:

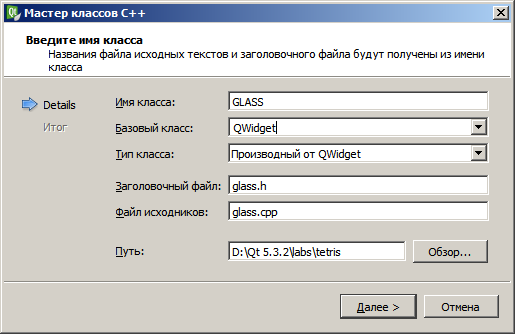
* класс игрового поля (стакана) –Glass (наследует от QWidget)
* класс «фигурка» - Figure (вспомогательный =>ненаследует от QObject)

Основным элементом приложения является пользовательскийвиджет:

* в котором отображаются уже «упавшие» цветные квадратики
* в котором «падает» текущая фигурка
* который принимает пользовательский ввод для управления падающей фигуркой (так как управлять падающими фигурками мы будем с помощью клавиатуры, то весь пользовательский ввод от клавиатуры **должен** направляться именно этому виджету)

Последовательность действий:

* С помощью QtCreator-а добавьте новый класс (Glass) и унаследуйте его от **QWidget**



* Преобразуйте виджет (Widget) в Glass

### Задаем размер стакана посредством дизайнера

Чтобы изобразить пустой стакан нужного размера, достаточно знать **размер стакана (сколько колонок и рядов) и размер клетки.**

#### Размер клетки

Вряд ли стоит позволять изменять размер клетки =>его можно задать статической целой константой в классе Glass, например,   
static const uint W=20;

Замечание: если Вы используете стандарт С++11, то константа не обязательно должна быть статической

#### Интервал, с которым будет «падать» фигурка

uint timerInterval;

#### Размер «стакана»

В классе Glass должны появиться:

* **переменные (**НЕ константы**)**, которые будут содержать текущее количество строк -**m\_rows** и количество элементов в строке- **m\_columns** в стакане (эти значения будем устанавливать посредством динамических свойств в Дизайнере)
* + **методы** get ()/set().

И то, и другое можно создать «вручную», а можно делегировать генерацию переменных и методов визарду. Так как создание «вручную» - очевидно => используем возможности визарда.

**Важно**! Для задания свойств посредством дизайнера типы свойств должны быть из тех,   
которые поддерживает QVariant – в нашем случае **int или uint**

Порядок действий:

* В классе Glass определите свойство Q\_PROPERTY (…). Свойство должно поддерживать и чтение, и запись =>  
  Q\_PROPERTY(uint rows READ rows WRITE setRows)
* Установите курсор на Q\_PROPERTY и нажмите Alt+Enter. Вы увидите всплывающее контекстное меню - «Создание отсутствующих членов Q\_PROPERTY…».  
  Нажимаем ENTER.   
  В результате визард добавит в класс как переменные (он сформирует их исходя из заданных имен и типов: rows ->m\_rows), так и методы чтения/записи свойств, имена которых будут сформированы исходя из заданных в теле макроса. В результате получится что-то вроде:

class Glass:public QWidget

{

Q\_OBJECT

Q\_PROPERTY(unsignedintrowsREADrowsWRITEsetRows)

Q\_PROPERTY(unsignedintcolumnsREADcolumnsWRITEsetColumns)

**unsigned int m\_rows;**

public:

explicit Glass(QWidget\*parent=0);

**unsigned int rows()const**

**{**

**return m\_rows;**

**}**

signals:

public slots:

**void setRows(unsigned int arg)**

**{**

**m\_rows=arg;**

**}**

…

};

* Самостоятельно добавьте свойство для чтения/записи количества колонок

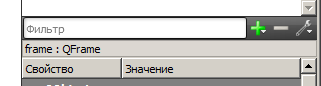
Теперь у нас появилась возможность:

* Как читать/изменять свойства в процессе выполнения посредством  
  bool [**setProperty**](file:///D:\Qt511\Задания\qobject.html#setProperty) ( constchar \* *name*, const QVariant & *value* )
* Так и задавать их в дизайнере (при этом вызов метода setProperty() генерирует дизайнер в setupUi() )

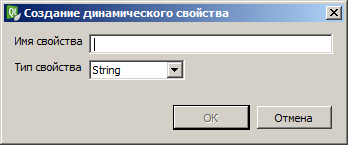
#### Задание свойств в дизайнере

Для задания динамического свойстваcпомощью дизайнера:

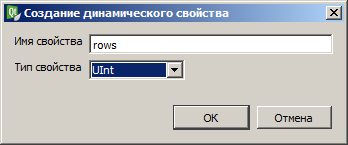
* Сделать активным виджетGlass
* Нажать «+» в



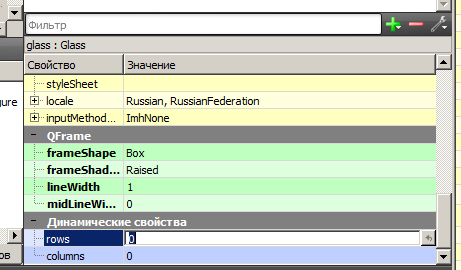
* Появится диалоговое окно



* Создаем свойства rows и columns с типом UInt



* При этом в окне свойств glassпоявится раздел «Динамические свойства»:



* Осталось только вместо нулей задать Ваши значения

В результате в setupUi() будет добавлен вызов:

…

glass->setProperty("rows",QVariant(20u)); //при этом, если в классе Glass есть свойство с именем “rows”, то будет вызван аксессор WRITE

# Данные класса Glass:

#### Признак – «Играем», например:

bool gameOn;

#### Текущий счет

uint score;

#### Размер клетки

можно задать статической константой - W

#### Текущее содержимое «стакана»

Стакан – это «двухмерный массив» клеток. Чтобы манипулировать содержимым стакана, о каждой клетке достаточно знать:

* Занята клетка или пуста
* Если занята, то - какого она цвета

=>можно цвет пустой клетки задавать специфическим цветом, например:

#define emptyCellQColor(150,150,150)//серыйцвет

или статической переменной класса

При этом нужно учесть, что размеры стакана будут задаваться динамически, =>собственно контейнер для хранения данных можно создать посредством   
**QVector<QVector<QColor>>glassArray;**

Важно!

* каким образом будет проинициализирован объект glassArray при создании объекта Glass?
* как (когда) можно задать размеры и проинициализировать все элементы?

#### В дальнейшем нам понадобятся другие данные =>

Будем добавлять их по мере необходимости

# Проблема:

исходные значения размеров стакана (m\_rows, m\_columns) формируются при выполнении

MainWindow::MainWindow(QWidget\*parent):

QMainWindow(parent),

ui(newUi::MainWindow)

{

ui->**setupUi**(this);

}

Объект Glass тоже создается при выполнении этой функции =>на момент вызова конструктора Glass - m\_rows и m\_columns ГАРАНТИРОВАННО еще не сформированы, а контейнер glassArray ПУСТОЙ (он содержит «совершенно пустой вектор совершенно пустых векторов»)!

=>нужно обеспечить формирование glassArray (m\_rows \*m\_columns) непосредственно после вызова   
ui->**setupUi**(this), но перед тем, как стакан m\_rows \*m\_columns будет первый раз отрисован:

# Слот для инициализации размеров стакана:

1. Выделяем действие по формированию glassArray в слот, например:  
   void Glass::slotGlassInit()  
   , в котором посредством QVector::resize() создаем вектор из m\_rows векторов, в каждом из которых m\_columns элементов
2. Дальше нужно предусмотреть «очистку» стакана, которая понадобится нам и в тех случаях, когда пользователь будет начинать новую игру =>логично очистку стакана выделить во вспомогательный метод, например:   
   void сlearGlass();
3. Дальше следует вычислить (в зависимости от количества строк и столбцов в стакане)размеры Glass. Высчитываем размер в пикселях:  
   QSize s(вычисляем размеры);
4. Задаем фиксированный размер стакана с помощью setFixedSize(s);

Вспомогательный метод void сlearGlass();   
В этом методе просто заполняем все элементы стакана значением «пусто» - emptyCell.   
Для заполнения удобно пользоваться методом QVector::fill().

Здесь же можно реализовать другие действия по «очистке» стакана, например, обнулить счет, задать начальный интервал таймера…

# Конструктор Glass

В нашей реализации нужно помнить о том, что размеры стакана (в клетках) будут задаваться **посредством динамических свойств** => на момент вызова конструктора стакана они еще **не будут установлены!** В конструкторе:

1. Обеспечить инициализацию признака «идет игра» (в дальнейшем счет, уровень…)
2. **Но!** Как будет проинициализирован glassArray???На момент вызова конструктора еще не сформированы с помощью динамических свойств размеры стакана => нужно гарантировать, что все данные, которые «завязаны» на размеры, обязательно будут сформированына момент отображения главного окна приложения =>то есть сразу после завершения конструктора MainWindow (а главное, после выполнения метода setupUi()). Для этого в конструкторе Glassустанавливаем асинхронное соединение (см. следующий раздел) и эмитируем самому себе «отложенный сигнал»  
   Замечание: все не проинициализированные значения для безопасности можно просто обнулить
3. Если Вы не задали в свойствах виджета в дизайнере, можно вызвать setFocusPolicy(Qt::StrongFocus)
4. Возможно, понадобятся еще какие-то инициализирующие действия… Будем добавлять их по мере необходимости

### Асинхронное соединение

Подсказки:

* Введите в класс Glass сигнал, например   
  void signalGlassInit();
* В конструкторе Glass установите между signalGlassInit () и уже реализованным слотом slotGlassInit() **асинхронное** соединение (в качестве последнего параметра метода connect() следует указать тип соединения **- Qt::QueuedConnection**), чтобы слот был вызван гарантированно после завершения инициализации (после setUpUi())
* В конструкторе Glassсэмитируйте сигнал signalGlassInit()

Проверьте функциональность отложенного слота – при запуске приложения размеры стакана должны подстроиться под Ваши значения.

# Отрисовка стакана

Перегружаем виртуальный метод **paintEvent()**, в котором будет прорисовываться текущее содержимое стакана + падающая фигурка (причем, фигурку следует рисовать только если «Идет игра»).

voidGlass::*paintEvent*(QPaintEvent\*event)

{

QPainter painter(this);

//стакан рисуем всегда!

//каждую клетку своим цветом, а для контура клетки можно задать любое перо =>  
получится эффект сетки

//Подсказки: для рисования клетки можно использовать методы QPainter:  
fillRect(),drawRect(), SetBrush()…

if(gameOn) //а фигурку будем рисовать только, если «идет игра»

{

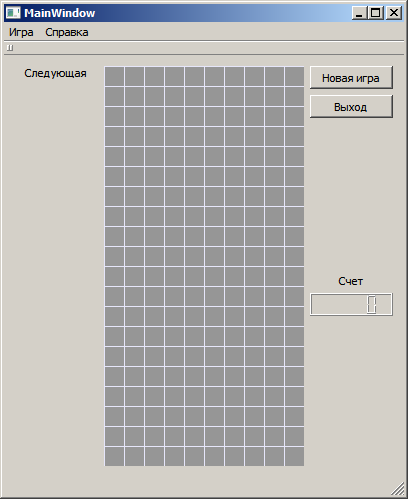
//здесь будем рисовать падающую фигурку

}

}

Подсказка: если закрашивать квадратик не целиком(fillRect() ), а «отступить» от каждого края на 1 пиксел, то получится эффект сетки и прямоугольник (drawRect() )можно специально не рисовать

Например, пока стакан пуст, он может выглядеть так:



Для заполнения внутренней части квадратика я использовала метод класса QPainter:  
fillRect()  
Вы можете это сделать другими средствами и придать стакану любой вид

## Класс Figure

Пусть фигурка всегда состоит из трех клеток, расположенных вертикально. Новая фигурка появляется всегда в центре стакана. Цвета можно циклически перемещать с помощью клавиатуры вниз/вверх. А при нажатиях влево/вправо фигурку целиком можно перемещать по горизонтали (если есть куда).

Создайте класс Figure. Поддержка механизма сигналов/слотов для наших целей особого смысла не имеет, поэтому этот класс не наследует от QObject. Класс Figure должен/может содержать следующие данные:

1. Массив из трех элементов QColor (это и есть наша фигурка)
2. Индексы верхней клетки фигуры (текущее положение фигурки в стакане) – m\_i, m\_j
3. Размер клетки – m\_W

Методы класса figure (у Вас набор методов может быть другим.Главное, обеспечить функциональность):

1. Конструктор
2. Методы изменения индексов верхней клетки фигуры
3. Методы получения индексов верхней/нижней клеток
4. Циклическое перемещение цветов –вниз/вверх, например:  
   void rotateColors(<признак – вверх/вниз>)
5. Формирование случайным образом цветов, например void MakeRandomColors();
6. Отрисовка фигуры, например:   
   void paintFigure(QPainter&painter);
7. …

## Модификация класса Glass

* Добавьте в класс два указателя:  
  Figure \*cur; //текущая (падающая) фигурка  
  Figure \*next; //образец следующей фигурки
* В конструкторе динамически создайте соответствующие объекты. Подсказка:  
  для обеих фигурок достаточно задать только ширину клетки, а индексы будем формировать в процессе выполнения => лучше задать нулевые значения
* Не забудьте уничтожить в деструкторе
* Создайте слот для начала новой игры, например, slotNewGame(). В слоте предусмотрите:  
  установку признака «Играем»;  
  очистка текущего содержимого стакана;  
  формирование текущей фигурки (цвета, начальные индексы);  
  формирование следующей фигурки (цвета – для отображении в образце, индексы – 0,0);  
  в дальнейшем здесь же будет эмитирование сигнала о перерисовке образца;  
  запуск таймера;  
  **перевод фокуса в стакан – setFocus() !!!Иначе события от клавиатуры будут поступать главному окну!**
* Соедините действие (action)о начале новой игры со слотом slotNewGame()
* Вызовите перерисовку стакана- repaint(). Примечание: без этого не отрисовывается фигурка при нажатии на кнопку на панели инструментов.

## Управление падающей фигуркой посредством клавиатуры

В классе Glass перегружаем виртуальныйметод  
void QWidget::**keyPressEvent(**QKeyEvent\*event**)** [virtual protected]

, в котором будем «управлять» падающей фигуркой посредством событий клавиатуры. В этом методе следует предусмотреть:

void Glass::keyPressEvent(QKeyEvent\*event)

{

if(gameOn)

{

**//Если«идет игра»**

switch(event->key()){//коднажатойклавиши

caseQt::Key\_Left:

//если есть «куда», перемещаем фигурку влево

break;

caseQt::Key\_Right:

//…

break;

caseQt::Key\_Down:

//циклически ”переливаем” цвета в фигурке сверху вниз

break;

caseQt::Key\_Up:

//циклически ”переливаем” цвета в фигурке снизу вверх

break;

caseQt::Key\_Space:

//«роняем» фигурку

break;

default:

QWidget::keyPressEvent(event); //принажатиилюбыхдругихклавишвызываембазовуюобработкусобытия

}

else{

QWidget::*keyPressEvent*(event);//предоставляемвозможностьбазовому

//классуобработатьсобытие

}

}

### Подсказка:

caseQt::Key\_Space://«роняем» фигурку

{

//ищем «куда ронять»

//вызываем вспомогательный метод (так как нам нужно будет выполнить те же действия, когда при «падении» вниз фигурке некуда будет падать, то есть она «упрется»), например:

// void AcceptColors(inti, intj);

//в котором:

//Добавляем фигурку в стакан

//Анализируем и сбрасываем текущее содержимое стакана, вызываем например: voidCheckGlass();

//Меняем местами next и cur

//Настраиваем next и cur (у nextобнуляем индексы, а у curустанавливаем). Также генерируем новые цвета в next

//эмитируем сигнал drawPattern(next); чтобы отрисовать в образце следующую фигурку

//

## Таймер

После того, как пользователь начал игру (при выборе пункта меню, нажатии кнопки «Новая игра» или нажатии дублирующей кнопки на toolbar-е) должна появиться и с определенным интервалом начать «падать» в стакан очередная фигурка =>в соответствующем слоте предусмотреть запуск таймера. При каждом тике таймера фигурка должна смещаться на одну клетку вниз, если есть свободная клетка. Если свободной клетки нет, то должны итеративно «сбрасываться» горизонтальные и вертикальные совокупности клеток (>=3), закрашенных одинаковым цветом. После этого генерируется и начинает «падать» следующая фигурка.

Встроенный таймер:

* В каждом классе, производном от QObject, есть возможность обрабатывать событие таймера. После того, как таймер запущен посредством метода startTimer(), с указанным интервалом будет вызываться перегруженный виртуальный метод timerEvent().
* «Остановить» таймер можно посредством метода killTimer()

Последовательность действий:

1. В классе Glassобъявите переменную   
   intidTimer;  
   в ней мы сохраним возвращаемое startTimer() значение, чтобы использовать его в killTimer()
2. В слоте Glass«Начать новую игру» предусмотрите запуск таймера - startTimer().
3. Перегрузите в классе Glass виртуальный метод  
   void*timerEvent*(QTimerEvent\*event)  
   , в котором реализуйте следующую логику:
   1. Если фигурке “есть куда падать”, перемещаем на клетку вниз и перерисовываем стакан
   2. Если фигурка «уперлась»,   
      добавляем ее квадратики в стакан,   
      анализируем получившееся содержимое стакана и рекурсивно сбрасываем одинаковые последовательности (>=3) (на это время таймер можно остановить),  
      генерируем новую фигурку (для этого достаточно поменять местами curи next,)
   3. Если сразу некуда падать – завершение игры (MessageBoxи **остановка таймера**)

## Виджет для отображения следующей фигурки

* Создайте класс, производный от QWidget (или QFrame), например, NextFigure. В этом классе достаточно:
  + объявить указатель Figure\* (указатель на следующую фигурку)
  + Слот, который будет обрабатывать появление новой следующей фигуры
  + Перегрузить виртуальный метод PaintEvent()
* Преобразуйтевиджет, созданный с помощью дизайнера в NextFigure

### МетодvoidNextFigure::*paintEvent*(QPaintEvent\*event):

чтобы было красиво, должен отображать фигурку посередине (а не слева!) =>отрисуйте фигурку с учетом размеров виджета и размеров клетки. Подсказка: можно вспомнить про метод QPainter::translate()

## Электронный индикатор - LCDNumber

Для отображения текущих достижений можно использовать виджетLCDNumber. В общем случае в таком виджете можно отображать группы целых значений - сегменты (как на электронных часах).

Возможности:

* Количество отображаемых сегментов можно задать в конструкторе или посредством метода setNumDigits()
* Отображение целых в десятичной, шестнадцатеричной, восьмеричной и двоичной системах счисления
* сегменты можно разделить точкой

Когда происходит анализ текущего содержимого стакана и сброс последовательностей одного цвета, нужно посчитать количество «сброшенных» клеток. А по окончании анализа сэмитировать сигнал, например:

voidsetScore(int);

Чтобы новое значение отображалось в LCDNumber, свяжите сигнал Glass::setScore() со слотом QLCDNumber::display(int)