Пишем скринсейвер на wxWidgets

Предисловие

Этот материал никоим образом не призывает читателя к написанию скринсейверов, а предназначается, прежде всего, для обзора некоторых возможностей библиотеки wxWidgets.

Мозг – это то, что нам нужно

Итак, о том, как собрать wxWidgets и о том, как создать приложение с использованием этой библиотеки я уже писал неоднократно, поэтому начнем сразу с примера. Для начала нам необходимо минимальное приложение с одной формой. Это будет база для создания скринсейвера.

Изобретать велосипед мы сегодня также не будем, а возьмем в качестве иллюстрации простейший скринсейвер с бегущими символами а-ля Matrix.

Основным объектам, который отображается на экране, у нас является символ. Каждый символ характеризуется следующими параметрами:

- Координаты
- Отображаемое значение
- Скорость перемещения
- Цвет

Для хранения информации о символе создадим такой класс:

```
class MatrixSymbol
{
        wxPoint m_Position;
        wxChar m_Symbol;
        int m_Speed;
        wxColour m_Colour;
public:
        MatrixSymbol()
                : m_Position(wxDefaultPosition), m_Symbol(wxT('0')),
                m_Speed(1), m_Colour(*wxGREEN) {}
        MatrixSymbol(const wxPoint & position, wxChar symbol,
                int speed, const wxColour & colour)
                : m_Position(position), m_Symbol(symbol),
                m_Speed(speed), m_Colour(colour) {}

        const wxPoint & GetPosition() {return m_Position;}
        void SetPosition(const wxPoint & value) {m_Position = value;}
        wxChar GetSymbol() {return m_Symbol;}
        void SetSymbol(wxChar value) {m_Symbol = value;}
        int GetSpeed() {return m_Speed;}
        void SetSpeed(int value) {m_Speed = value;}
        const wxColour & GetColour() {return m_Colour;}
        void SetColour(const wxColour & value) {m_Colour = value;}
};
```

Таких символов у нас на экране должно быть много и, соответственно, для хранения всего этого добра нам необходим массив:

```
#include <wx/dynarray.h>
...
WX_DECLARE_OBJARRAY(MatrixSymbol, MatrixSymbolArray);
...
#include <wx/arrimpl.cpp>

WX_DEFINE_OBJARRAY(MatrixSymbolArray);
```

Лепим графический интерфейс

Отлично, подготовительный этап мы закончили, теперь можно приступать к реализации графического интерфейса.

Создадим новый компонент, унаследованный от wxWindow и добавим в него в виде переменной-члена класса массив объектов `MatrixSymbol`, а также метод, инициализирующий этот массив значениями:

```
class MatrixCanvas: public wxWindow
{
    ...
    void InitMatrix();
    ...
    MatrixSymbolArray m_Symbols;
    ...
};

void MatrixCanvas::InitMatrix()
{
        int width(0), height(0);
        int sw(0), sh(0);
        GetTextExtent(wxT("0"), &sw, &sh);
        GetClientSize(&width, &height);
        m_Symbols.Clear();
        for(int x = 0; x < width; x += sw+2)
        {
                m_Symbols.Add(MatrixSymbol(
                        wxPoint(x, 0),
                        rand()%2 ? wxT('0') : wxT('1'),
                        3+rand()%5,
                        wxColour(0, rand()%200+56, 0)));
        }
}
```

Что у нас делает метод `InitMatrix()`? В зависимости от размеров клиентской области компонента в массив добавляется некоторое количество объектов `MatrixSymbol` со случайными координатами, отображаемым значением «0» или «1», и различными цветами (выбирается случайная градация зеленого).

Теперь нам нужно обеспечить отображение символов на экране. Для этого создадим обработчики событий `wxEVT_PAINT` и `wxEVT_ERASE_BACKGROUND`.

```
BEGIN_EVENT_TABLE( MatrixCanvas, wxWindow )
    EVT_PAINT( MatrixCanvas::OnPaint )
    EVT_ERASE_BACKGROUND( MatrixCanvas::OnEraseBackground )
END_EVENT_TABLE()

void MatrixCanvas::OnPaint( wxPaintEvent& event )
{
    wxBufferedPaintDC dc(this);
        dc.SetBackground(wxBrush(GetBackgroundColour()));
        dc.Clear();
        wxFont font = GetFont();
#if defined(__WXWINCE__)
        int fontSize = 14;
#else
        int fontSize = 48;
#endif
        font.SetPointSize(fontSize);
        dc.SetFont(font);
        dc.SetTextForeground(wxColour(00, 20, 00));
        dc.DrawLabel(wxT("http://wxwidgets.info"),
                wxRect(0, 0, dc.GetSize().GetWidth(), dc.GetSize().GetHeight()),
                wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL);
        dc.SetFont(GetFont());
        for(size_t i = 0; i < m_Symbols.Count(); i++)
        {
                dc.SetTextForeground(m_Symbols[i].GetColour());
                dc.DrawText(wxString::Format(wxT("%c"), m_Symbols[i].GetSymbol()),
```

```
                        m_Symbols[i].GetPosition());
        }
}

void MatrixCanvas::OnEraseBackground( wxEraseEvent& event )
{
}
```

Обработчик события `wxEVT_ERASE_BACKGROUND` пустой (без вызова `event.Skip()`). Это обеспечит нам перерисовку компонента без мерцания.

В обработчике события `wxEVT_PAINT` у нас создается контекст устройства, устанавливается цвет фона равный цвету фона нашего компонента, затем происходит очистка (это равноценно заливке цветом). После этого в центре компонента отрисовывается надпись и затем в цикле происходит отрисовка всех символов из массива.

Далее нам необходимо добавить обработчик события `wxEVT_SIZE` для того чтобы при изменении размера компонента символы отображались по всей ширине. В обработчике мы просто будем вызывать метод `InitMatrix()`, который заполняет массив символами.

```
BEGIN_EVENT_TABLE( MatrixCanvas, wxWindow )
    ...
    EVT_SIZE( MatrixCanvas::OnSize )
END_EVENT_TABLE()
...
void MatrixCanvas::OnSize( wxSizeEvent& event )
{
        InitMatrix();
        Refresh();
}
```

Так, отображения символов мы добились, но нам еще необходимо сделать так чтобы символы перемещались по экрану. Код, обеспечивающий перемещение символов по экрану, лучше всего выполнять в обработчике события таймера.

```
class MatrixCanvas: public wxWindow
{
    ...
    wxTimer * m_MovementTimer;
};

MatrixCanvas::MatrixCanvas(wxWindow* parent, wxWindowID id, const wxPoint& pos, const wxSize& size,
long style)
{
    Init();
    Create(parent, id, pos, size, style);
}

bool MatrixCanvas::Create(wxWindow* parent, wxWindowID id, const wxPoint& pos, const wxSize& size,
long style)
{
    wxWindow::Create(parent, id, pos, size, style);
    CreateControls();
    return true;
}

MatrixCanvas::~MatrixCanvas()
{
        wxDELETE(m_MovementTimer);
}

void MatrixCanvas::Init()
{
    m_PreviewMode = false;
}

void MatrixCanvas::CreateControls()
{
    this->SetForegroundColour(wxColour(0, 255, 0));
    this->SetBackgroundColour(wxColour(0, 0, 0));
```

```
    int timerID = wxNewId();
        m_MovementTimer = new wxTimer(this, timerID);
        Connect(timerID, wxEVT_TIMER,
                wxTimerEventHandler(MatrixCanvas::OnMovementTimer));
        InitMatrix();
        Refresh();
        m_MovementTimer->Start(30);
}

void MatrixCanvas::OnMovementTimer( wxTimerEvent & event )
{
        for(size_t i = 0; i < m_Symbols.Count(); i++)
        {
                int y = m_Symbols[i].GetPosition().y + m_Symbols[i].GetSpeed();
                if(y > GetClientSize().GetHeight())
                {
                        y = -20;
                        m_Symbols[i].SetSpeed(3+rand()%5);
                        m_Symbols[i].SetColour(wxColour(0, rand()%200+56, 0));
                        m_Symbols[i].SetSymbol(rand()%2 ? wxT('0') : wxT('1'));
                }
                m_Symbols[i].SetPosition(wxPoint(
                        m_Symbols[i].GetPosition().x, y));
        }
        Refresh();
}
```

Как видно, в методе `CreateControls()` создается таймер и с помощью метода `Connect()` ему назначается обработчик события. В деструкторе таймер удаляется.

Отлично. Скринсейвер может работать в обычном режиме и в режиме предварительного просмотра. В обычном режиме нам необходимо обеспечить реакцию на действия пользователя, а именно на нажатия клавиш, а также на клик мышкой. Для этого мы создадим переменную-член класса `bool m_PreviewMode` и, в зависимости от ее значения, в обработчиках событий нажатия клавиш и нажатия кнопок мыши будем закрывать главную форму приложения.

```
void MatrixCanvas::OnMouse( wxMouseEvent& event )
{
        if(event.LeftDown() || event.MiddleDown() || event.RightDown())
        {
                if(!m_PreviewMode)
                {
                        wxFrame * frame = wxDynamicCast(wxTheApp->GetTopWindow(), wxFrame);
                        if(frame) frame->Close();
                }
        }
}

void MatrixCanvas::OnChar( wxKeyEvent& event )
{
    if(!m_PreviewMode)
    {
        wxFrame * frame = wxDynamicCast(wxTheApp->GetTopWindow(), wxFrame);
        if(frame) frame->Close();
    }
}
```

Собственно, на этом работа над компонентом завершена. Теперь надо поместить его на главную форму:

```
void MatrixEffectMainFrame::CreateControls()
{
    MatrixEffectMainFrame* itemFrame1 = this;

    wxBoxSizer* itemBoxSizer2 = new wxBoxSizer(wxVERTICAL);
    itemFrame1->SetSizer(itemBoxSizer2);

    m_Canvas = new MatrixCanvas( itemFrame1, ID_CANVAS, wxDefaultPosition, wxSize(100, 100),
wxNO_BORDER );
    itemBoxSizer2->Add(m_Canvas, 1, wxGROW, 0);
}
```

Вообще супер. На этом работу над графическим интерфейсом мы закончим.

Обработка параметров командной строки

Итак, теперь нам осталась самая интересная часть работы, а именно обработка параметров командной строки.

Скринсейвер в Windows может запускаться с тремя различными аргументами командной строки:

- “/s” или “/S” – непосредственно для запуска скринсейвера
- “/c:<parent>” – для вызова окна настройки скринсейвера, где <parent> - численное представление дескриптора родительского окна.
- “/p:<parent>” – для запуска скринсейвера в режиме предварительного просмотра, где <parent> - численное представление дескриптора родительского окна.

Для обработки аргументов командной строки в wxWidgets существует класс `wxCmdLineParser`. Его мы и будем использовать.

```
bool wxMatrixEffectApp::OnInit()
{
#if defined(__WXMSW__) && !defined(__WXWINCE__)
        wxCmdLineParser parser(argc, argv);
        parser.AddSwitch(wxT("S"), wxEmptyString,
                wxEmptyString, wxCMD_LINE_PARAM_OPTIONAL);
        parser.AddSwitch(wxT("s"), wxEmptyString,
                wxEmptyString, wxCMD_LINE_PARAM_OPTIONAL);
        parser.AddOption(wxT("c"), wxEmptyString,
                wxEmptyString, wxCMD_LINE_VAL_NUMBER, wxCMD_LINE_PARAM_OPTIONAL);
        parser.AddOption(wxT("p"), wxEmptyString,
                wxEmptyString, wxCMD_LINE_VAL_NUMBER, wxCMD_LINE_PARAM_OPTIONAL);
        if(parser.Parse(false) == 0)
        {
                long parentHWND(0);
```

```
                if(parser.Found(wxT("S")) || parser.Found(wxT("s")))
                {
                        MatrixEffectMainFrame* mainWindow =
                                new MatrixEffectMainFrame( NULL );
                        mainWindow->ShowFullScreen(true);
                        return true;
                }
                else if(parser.Found(wxT("c")))
                {
                        wxMessageBox(
                                _("No settings for this screensaver.\n\
For more information visit http://wxwidgets.info"));
                        return false;
                }
                else if(parser.Found(wxT("p"), &parentHWND))
                {
                        wxWindow * parent = new wxWindow;
                        parent->SetHWND((HWND)parentHWND);
                        RECT r;
                        GetWindowRect((HWND)parent->GetHWND(), &r);
                        MatrixCanvas* mainWindow = new MatrixCanvas(
                                parent, ID_MATRIXEFFECTMAINFRAME,
                                wxPoint(0,0), wxSize(r.right-r.left, r.bottom-r.top),
                                wxNO_BORDER);
                        mainWindow->SetPreviewMode(true);
                        SetTopWindow(mainWindow);
                        mainWindow->Show(true);
                        return true;
                }
        }
    return false;
#else
        MatrixEffectMainFrame* mainWindow =
                new MatrixEffectMainFrame( NULL );
        SetTopWindow(mainWindow);
#if defined(__WXWINCE__)
        mainWindow->Show(true);
#else
        mainWindow->ShowFullScreen(true);
#endif
        return true;
#endif
}
```

Как видно из кода, в методе `OnInit()` класса приложения мы создаем объект `wxCmdLineParser`, добавляем описания всех возможных параметров:

- Метод `AddSwitch()` указывает что командная строка может содержать аргумент вида `/<argument>` или `--<argument>` без дополнительных параметров.
- Метод `AddOption()` указывает, что командная строка может содержать аргумент вида `/<argument>=<value>` или `--<argument>=<value>` или `/<argument>:<value>`, т.е. параметр командной строки, содержащий значение (строковое, численное, дату).

После настройки объекта `wxCmdLineParser` происходит вызов метода `Parse()`, который проверяет соответствие командной строки указанным настройкам.

Затем происходит проверка наличия параметров.

- Если программа запущена с параметром “/s” или “/S”, то отображается главное окно приложения.
- Если программа запущена с параметром “/c”, то отображается окно сообщения о том, что наш скринсейвер не требует настройки.
- Если программа запущена с параметром “/p”, то извлекается дескриптор родительского окна, из него создается объект `wxWindow` и вместо главного окна приложения создается компонент `MatrixCanvas`, который размещается на родительском окне.

Еще раз посмотрим на код, создающий `wxWindow` из дескриптора окна:

```
wxWindow * parent = new wxWindow;
parent->SetHWND((HWND)parentHWND);
```

На самом деле это очень полезный прием и может быть использован для взаимодействия wxWidgets-приложений с приложениями, написанными с помощью других библиотек.

Ну вот, собственно и все. В результате мы получим что-то вроде этого: