Quickstart

创建第一个应用

在Quickstart中编写：

```
#include "include/Element.hh"
#include "system/SystemIO.hh"

using namespace easy;

int main() {
    Register<Renderer>();
    Element elem = MakeElement();
    Renderer::MainLoop(elem);
}
```

编译运行，出现纯黑的窗口。

此处：

- `#include "system/SystemIO.hh"` 是必要的，其中包含了不同平台Render的实现。`Register<Renderer>();` 语句就是在进行渲染器的注册，一般在 `main` 函数的开头即调用该语句。
- `Element elem = MakeElement();` 创建了一个 `Element` 控件，`Element` 是所有控件的基类。请注意，控件是以引用计数的，一般而言，用户不需要关心控件对象的生命周期（所有控件都是一种 `std::shared_ptr`）。
- `Renderer::MainLoop(elem);` 执行主循环。一般在 `main` 函数的结尾调用该语句，因为程序将不会从这个函数中返回（内部发生了无限循环）。`MainLoop` 函数的参数是任意控件，表示用户将该控件作为图形界面的根控件。

下面我们修改 `main` 中的代码，观察不同的效果。

背景和边框

编写：

```
int main() {
    Register<Renderer>();
    Element elem = MakeElement();
    elem->BackgroundColor = Colors::Red;
    Renderer::MainLoop(elem);
}
```

编译运行，背景变成了红色。

此处：

- 由于 `elem` 实际上是智能指针，需要使用 `->` 而不是 `.` 来指示其成员。
- 由于 `elem` 是根控件，且未指定其大小，默认将铺满整个界面，因此当背景颜色被设置为红色时，整个界面都将是红色。
- `Colors::Red` 是一个内置颜色，这些颜色定义如下：

```
namespace Colors {
    constexpr Color
        White = Color::FromARGB(0xffffff),
        Black = Color::FromARGB(0x000000),
        Red = Color::FromARGB(0xd71345),
        Blue = Color::FromARGB(0x426ab3),
        Green = Color::FromARGB(0x7fb80e),
        Yellow = Color::FromARGB(0xffd400),
        Purple = Color::FromARGB(0x9b95c9),
        Brown = Color::FromARGB(0x74531f),
        Trasparent = Color::FromARGB(0xff000000);
}
```

`Color::FromARGB` 提供了方法可以自行设置颜色。不过请注意，A分量未被完整实现，目前只有两种行为：

- `ARGB = 0xFF000000` ：完全透明
- 其他任何情况：完全不透明

运行：

```
int main() {
    Register<Renderer>();
    Element elem = MakeElement();
    elem->BackgroundColor = Colors::Red;
    elem->Margin = { 0,2,10,60 };
    Renderer::MainLoop(elem);
}
```

红色块不再铺满整个界面，而是露出了一定的边距，边距大小由 `Margin` 属性指定。

此处：

- `Margin` 的类型是EasyGraphics的基本类型 `Rect` ，可以通过大括号直接创建，四个参数分别表示Left、Top、Right、Bottom，同一类型之间能做加减乘除等线性运算。

运行：

```
int main() {
    Register<Renderer>();
    Element elem = MakeElement();
    elem->BackgroundColor = Colors::Red;
    elem->BorderColor = Colors::Green;
    elem->BorderThickness = { 10,10,10,10 };
    elem->Margin = { 0,2,10,60 };
    Renderer::MainLoop(elem);
}
```

出现了绿色边框。

此处：

- `BorderThickness` 同样是 `Rect` ，定义了边框的粗细。请注意边框总是向外部延伸的，因此未显示左侧边框（因为 `Margin.Left = 0` ）。

- `BorderColor` 定义了边框颜色。

网格

```
#include "include/Element.hh"
#include "include/Grid.hh"
#include "system/SystemIO.hh"

using namespace easy;

int main() {
    Register<Renderer>();
    Element elem = MakeElement();
    elem->BackgroundColor = Colors::Red;
    elem->BorderColor = Colors::Green;
    elem->BorderThickness = { 10,10,10,10 };
    elem->Margin = { 0,2,10,60 };
    Grid grid = MakeGrid({ 100,200,0 }, { 200,100,0 });
    grid->Set(2, 2, elem);
    Renderer::MainLoop(grid);
}
```

此处：

- `MakeGrid` 需要提供两个长度相同的列表分别指示每行和每列的大小，此处建立了一个3x3的网格，网格的第3行第3列（index为2, 2）容纳了 `elem`。
- 指示为0的行或列将自动分配。即，网格控件总是会去铺满整个父元素的空间，然后它首先分配那些指定了大小的行和列，再将多余的部分平均分给所有指示为0的行和列。
- `Grid` 也是一种 `Element`，你可以设置它的 `BackgroundColor` 等属性。

事件

```
int main() {
    Register<Renderer>();
    Element elem = MakeElement();
    elem->BackgroundColor = Colors::Red;
    elem->BorderColor = Colors::Green;
    elem->BorderThickness = { 10,10,10,10 };
    elem->Margin = { 0,2,10,60 };
    Grid grid = MakeGrid({ 100,200,0 }, { 200,100,0 });
    grid->Set(2, 2, elem);

    elem->Drag += [](Element sender, MouseEventArgs args) {
        sender->BackgroundColor.Red += args.offset.X;
        sender->BackgroundColor.Green += args.offset.Y;
        Renderer::Invalidated() = true;
    };

    elem->Click += [](Element sender, MouseEventArgs args) {
        sender->BackgroundColor = Colors::Red;
        Renderer::Invalidated() = true;
    };

    Renderer::MainLoop(grid);
}
```

此处：

- `elem->Drag` 和 `elem->Click` 都是控件 `elem` 的事件，`+=` 运算符将一个接受指定参数的函数添加到事件的侦听列表中，从而使得每次事件发生时，都会调用该函数。一般而言，使用 `lambda` 函数是方便的，对于拖拽和点击这些鼠标事件函数，要求侦听函数形如以下的一种：

  ```
  [](Element sender, MouseEventArgs args) { ... };
  [](auto sender, auto args) { ... };
  void ...(Element sender, MouseEventArgs args) { ... }
  ```

  其中 `sender` 是触发事件的控件，`args` 包含事件发生时鼠标的位置和鼠标的位移（对于拖拽来说）。

- `Renderer::Invalidated() = true;` 是必需的，它告诉渲染器，一些可视属性发生了变化，需要重新渲染。
- `Click` 在抬起鼠标时触发，`Drag` 在拖动时持续触发。

闭包

事件提供的 `MouseEventArgs` 或 `EventArgs` 等信息往往是有限的，我们的事件处理函数不可能只能使用这些信息，例如，如果拖拽 `elem` 时希望改变 `grid` 的颜色该怎么办。

幸运的是，C++的 `lambda` 语法提供闭包功能能够支持这一点，这也是推荐使用 `lambda` 而非传统函数作为事件侦听的原因。

熟悉 `lambda` 语法的用户很容易编写以下例子：

```
int main() {
    Register<Renderer>();
    Element elem = MakeElement();
    elem->BackgroundColor = Colors::Red;
    elem->BorderColor = Colors::Green;
    elem->BorderThickness = { 10,10,10,10 };
    elem->Margin = { 0,2,10,60 };
    Grid grid = MakeGrid({ 100,200,0 }, { 200,100,0 });
    grid->Set(2, 2, elem);
    grid->BackgroundColor = Colors::Blue;

    elem->Drag += [&grid](Element sender, MouseEventArgs args) {
        grid->BackgroundColor.Red += args.offset.X;
        grid->BackgroundColor.Green += args.offset.Y;
        Renderer::Invalidated() = true;
    };

    elem->Click += [&grid, &elem](auto sender, auto args) {
        elem->BackgroundColor = grid->BackgroundColor;
        Renderer::Invalidated() = true;
    };

    Renderer::MainLoop(grid);
}
```

此处：

- 对于控件，采用 `&` 或是 `=` 来捕获，取决于设计需要。尽管大多数情况下它们是等效的，但是：
  - 如果捕获的控件尚未定义，即捕获语句在定义语句前，必须使用引用捕获。

- 如果捕获的控件指针（如前所述，控件变量是智能指针）是局部变量，且在 `Renderer::MainLoop` 执行时已经被析构，必须使用值捕获（否则主循环进行时，引用捕获将试图访问空引用）。

动画

对上一个例子稍作修改：

```
int main() {
    Register<Renderer>();
    Element elem = MakeElement();
    elem->BackgroundColor = Colors::Red;
    elem->BorderColor = Colors::Green;
    elem->BorderThickness = { 10,10,10,10 };
    elem->Margin = { 0,2,10,60 };
    Grid grid = MakeGrid({ 100,200,0 }, { 200,100,0 });
    grid->Set(2, 2, elem);
    grid->BackgroundColor = Colors::Blue;

    elem->Drag += [&grid](Element sender, MouseEventArgs args) {
        grid->BackgroundColor.Red += args.offset.X;
        grid->BackgroundColor.Green += args.offset.Y;
        Renderer::Invalidated() = true;
    };

    elem->Click += [&grid, &elem](auto sender, auto args) {
        elem->BeginAnimation(
            elem,
            &_Element::BackgroundColor,
            elem->BackgroundColor,
            grid->BackgroundColor,
            500
        );
        Renderer::Invalidated() = true;
    };

    Renderer::MainLoop(grid);
}
```

即，松开鼠标时并不是立即变化颜色，而是存在一个渐变的动画。所有控件都有 `BeginAnimation` 方法，这个函数的签名如下：

```
template<typename T, typename O, typename D>
void BeginAnimation(
    std::shared_ptr<D> object,
    T O::*prop,
    T from,
    T to,
    unsigned miliseconds,
    EaseFunction ease = EaseLinear,
    bool multiple = false,
    double FPS = 40
);
```

此处：

- 该方法在 `object` 的属性 `prop` 上发起一个动画，使其在 `miliseconds` 毫秒内，其值从 `from` 变化到 `to`。
- `object` 参数是执行动画的控件对象，这与动画的管理者（`BeginAnimation` 方法的调用者）并不一定相同。执行动画的对象是 `prop` 属性发生改变的对象，方法的调用者是管理动画的对象，即执行的 `Animation` 将被保存到管理者的动画列表中，如果 `multiple` 不为 `false`，`BeginAnimation` 会中止管理者的动画列表中所有改变 `prop` 属性的动画。
- `prop` 参数是动画实际改变的属性，这个属性必须继承自 `Linear`，如 `Rect, Pos, Size, Color`，或者是C++内置的标量，如 `int, float, double`。具体使用时，可以参考以下格式传入：
  - 如果希望改变 `A` 控件的属性 `B`，传入 `&_A::B`。
- `from` 参数是属性的起始值，下一个时间片，属性就会立即变为这个值，即使当前时刻属性的值与 `from` 所指示的值不同。
- `to` 参数是属性的最终值，如果动画没有被中止（中止往往是因为另一个改变同一属性的动画覆盖了它），动画结束时属性将变化为 `to` 所指示的值。
- `miliseconds` 是动画持续的大致毫秒数，这个值不能为0。
- `ease` 是动画使用的缓动函数，可选的（目前已实现的）函数有：
  - EaseLinear：默认选项，无缓动
  - EaseInCubic：缓入
  - EaseOutCubic：缓出
  - EaseInOutCubic：缓入缓出
  - EaseInBounce：缓入反弹
  - EaseOutBounce：缓出反弹
  - EaseInOutBounce：缓入缓出反弹
- `multiple` 指明了动画是否可叠加，如果不可叠加，`BeginAnimation` 将覆盖动画管理者拥有的改变同一属性的其他动画。
- `FPS` 指明了动画的执行速率，但实际显示的速率不会高于渲染的帧率。

计时器

`Timer` 类包含两个方法：`DelayInvoke` 和 `RecurrentInvoke`。

```
template<typename F, typename ... T>
static TimerHandle DelayInvoke(unsigned delay, F&& f, T&&... args);
```

此处：

- `delay` 参数表明函数调用延迟的毫秒数。
- `f` 参数是待调用的函数，可以是 `lambda` 函数，但不能是成员函数（这种情况可以转化为 `lambda`）。
- `args...` 是 `f` 需要的参数，可以为空。
- 返回类型 `TimerHandle` 是 `std::shared_ptr<bool>`，可以读写其中的值，这个布尔值表明调用是否完成。对其读可以知道是否已经完成调用；对其写（从 `false` 改为 `true`）可以阻止将要发生的调用。

```
template<typename F, typename ... T>
static TimerHandle RecurrentInvoke(unsigned interval, unsigned times, F&& f,
T&&... args);
```

此处：

- `interval` 参数表明函数循环调用间隙的毫秒数。
- `times` 参数是循环调用的次数，这个值为0表示无上限。
- `f` 参数是待调用的函数，可以是 `lambda` 函数，但不能是成员函数（这种情况可以转化为 `lambda` ）。
- `args...` 是 `f` 需要的参数，可以为空。
- 返回类型 `TimerHandle` 是 `std::shared_ptr<bool>` ，可以读写其中的值，这个布尔值表明调用是否完成。对其读可以知道是否已经完成调用；对其写（从 `false` 改为 `true` ）可以阻止将要发生的调用。

快速构建GUI

IMGUI，即Immediate Mode Graphical User Interface，是一种即时模式的图形接口，旨在简化设计，避免陡峭的学习曲线，提高设计效率。EasyGraphics提供了immediate-mode-like GUI，能够以类似IMGUI或XAML的方式构建图形界面（尽管实现上并不是真正的即时模式）。

让我们重新构建上一个例子：

```
#include "include/ImGui.hh"
#include "system/SystemIO.hh"

int main() {
    using namespace easy;
    using namespace easy::imgui;

    begin_im;
    with (MakeGrid({100,200,0}, {200,100,0})) {
        BackgroundColor = Colors::Blue;
        with (MakeElement()) {
            BackgroundColor = Colors::Red;
            BorderColor = Colors::Green;
            BorderThickness = { 10,10,10,10 };
            Margin = { 0,2,10,60 };
            GridPosition = { 2, 2 };
            Drag += [=](Element sender, MouseEventArgs args) {
                Parent->BackgroundColor.Red += args.offset.X;
                Parent->BackgroundColor.Green += args.offset.Y;
                Renderer::Invalidated() = true;
            };

            Click += [=](auto sender, auto args) {
                This->BeginAnimation(
                    This,
                    &_Element::BackgroundColor,
                    This->BackgroundColor,
                    Parent->BackgroundColor,
                    500
                );
                Renderer::Invalidated() = true;
            };
        }
    }
}
```

此处：

- `using namespace easy::imgui;` 几乎是必需的，否则代码将看上去过于冗长。

- `begin_im` 是一个宏，需要在任何 `with` 之前使用。
- `with` 是一个宏，接受一个参数指示创建的控件，其后应当跟随一对大括号，这是 `with` 的作用域。

  在作用域中：

  - `This` 是一个局部变量，指示当前作用域所属的控件。
  - `Parent` 是一个局部变量，指示父级作用域所属的控件。根控件作用域中，`Parent` 为空。
  - 任何属性或事件都可以直接赋值，正常情况下 `A->B = C;` 在此处变为 `B = C` 。
  - 不应该定义名字与任何属性或事件名相同的变量，也不应该定义名为 `This` 或 `Parent` 的变量（这样做会覆盖它们原本的语义）。这些保留名都是大写字母开头的。
  - `GridPosition` 是一个特殊属性，它仅当父级控件为 `Grid` 时被使用，这个属性依次指明了当前控件在父级控件中的列索引和行索引（注意行列顺序和 `Grid->Set` 中相反）。
  - 由于 `This` 和 `Parent` 的局部性，`lambda` 在捕获它们的时候必须按值捕获。
- 根控件的 `with` 块之后的语句不会被执行，因为 `Renderer::MainLoop` 主循环发生在离开根控件的 `with` 块的瞬间。同理，多个根 `with` 存在时，除了第一个，其他都是无效的。
- `with` 只适用于初始化控件，不应该出现在 `main` 函数体以外的任何位置，例如在事件的处理函数中，不能使用它来动态添加控件，也即，不能在 `lambda` 中使用它。

例：计算器应用

带动画的计算器

```
#include "include/OverlapPanel.hh"
#include "include/Grid.hh"
#include "include/Label.hh"
#include "system/SystemIO.hh"
#include <iostream>
#include <string>
using namespace easy;

int as_int(std::string s) {
    const char* str = s.c_str() + 2;
    int x = 0;
    bool neg = false;
    if (str[0] == '-') str++, neg = true;
    while (*str) {
        x = 10 * x + *str - '0';
        str++;
    }
    return neg ? -x : x;
}

int main() {
    Register<Renderer>();

    int result = 0;
    bool wait_new = true;

    OverlapPanel history = MakeOverlapPanel();
    history->Margin = { 50, 0, 50, 50 };
    auto add_history = [&](std::string info) {
        Label item = MakeLabel();
        item->Text = info;
        item->FontHorizontalAlignment = HorizontalAlignType::Left;
```

```
        item->SpecSize = { 340, 40 };
        item->Margin.Top = 10;
        item->BeginAnimation(
            Element(item),
            &_Element::BackgroundColor,
            Color::FromARGB(0xE6E6E6),
            Color::FromARGB(0xF0F0F0),
            500,
            EaseInOutCubic
        );
        item->BeginAnimation(
            item,
            &_Label::FontColor,
            Color::FromARGB(0xE6E6E6),
            Colors::Black,
            500,
            EaseInOutCubic
        );
        auto [begin, end] = history->GetRange();
        int total = history->Capacity();
        for (int i = 0; begin != end; ++i, ++begin) {
            if (i > total - 6)
                (*begin)->BeginAnimation(
                    *begin,
                    &_Element::Margin,
                    (*begin)->Margin,
                    Rect { 0, 60 + 50 * (total - 1 - i) },
                    500,
                    EaseInOutCubic
                );
            else if (i == total - 6) {
                (*begin)->BeginAnimation(
                    std::dynamic_pointer_cast<_Label>(*begin),
                    &_Label::FontColor,
                    Colors::Black,
                    Color::FromARGB(0xE6E6E6),
                    500,
                    EaseInOutCubic
                );
                (*begin)->BeginAnimation(
                    *begin,
                    &_Element::BackgroundColor,
                    (*begin)->BackgroundColor,
                    Color::FromARGB(0xE6E6E6),
                    500,
                    EaseInOutCubic
                );
            }
        }
        history->Add(item);
    };

    Label input = MakeLabel();
    input->Margin = { 20 };
    input->SpecSize = { 700, 70 };
    input->BackgroundColor = Color::FromARGB(0xF0F0F0);
    input->FontSize = FontSizeType::Large;
    input->FontColor = Colors::Black;
```

```
    input->VerticalAlignment = VerticalAlignType::Center;
    input->HorizontalAlignment = HorizontalAlignType::Left;
    input->FontHorizontalAlignment = HorizontalAlignType::Left;
    input->FontVerticalAlignment = VerticalAlignType::Center;

    Label hint = MakeLabel();
    hint->Margin = { 0, 0, 35 };
    hint->SpecSize.Height = 70;
    hint->FontSize = FontSizeType::Large;
    hint->VerticalAlignment = VerticalAlignType::Center;
    hint->HorizontalAlignment = HorizontalAlignType::Right;
    hint->Text = " ";

    OverlapPanel hint_panel = MakeOverlapPanel();
    hint_panel->Add(input);
    hint_panel->Add(hint);

    Grid btns = MakeGrid({ 80,80,80,80 }, { 80,80,80,80 });
    btns->Margin = { 20 };


    auto MakeBtn = []() {
        Label btn = MakeLabel();
        btn->SpecSize = { 76, 76 };
        btn->VerticalAlignment = VerticalAlignType::Center;
        btn->HorizontalAlignment = HorizontalAlignType::Center;
        btn->BackgroundColor = Color::FromARGB(0xF0F0F0);
        return btn;
    };

    for (int i = 0; i < 10; ++i) {
        Label btn = MakeBtn();
        btn->Text = std::to_string(i);
        btn->Click += [&, i](Element, MouseEventArgs) {
            if (wait_new) input->Text = "  " + std::to_string(i), wait_new =
false;
            else input->Text += std::to_string(i);
            Renderer::Invalidated() = true;
        };
        btn->BackgroundColor = Color::FromARGB(0xFAFAFA);
        if (i) btns->Set((i - 1) / 3, (i - 1) % 3, btn);
        else btns->Set(3, 1, btn);
    }


    for (int i = 0; i < 4; ++i) {
        Label btn = MakeBtn();
        btn->Text = "+-*/"[i];
        btn->Click += [&, i](Element, MouseEventArgs) {
            hint->Text[0] = "+-*/"[i];
            wait_new = true;
            result = as_int(input->Text);
            Renderer::Invalidated() = true;
        };
        btns->Set(i, 3, btn);
    }

    Label eq = MakeBtn();
```

```
    eq->Text = "=";
    eq->Click += [&](Element, MouseEventArgs) {
        std::string info = std::to_string(result);
        if (hint->Text[0] != ' ') info += hint->Text + input->Text.substr(2);
        else info = input->Text.substr(2);
        if (hint->Text[0] == '+') input->Text = "  " + std::to_string(result +
as_int(input->Text));
        else if (hint->Text[0] == '-') input->Text = "  " +
std::to_string(result - as_int(input->Text));
        else if (hint->Text[0] == '*') input->Text = "  " +
std::to_string(result * as_int(input->Text));
        else if (hint->Text[0] == '/' && as_int(input->Text)) input->Text = "  "
+ std::to_string(result / as_int(input->Text));
        hint->Text[0] = ' ';
        result = as_int(input->Text);
        info += "=" + std::to_string(result);
        add_history(info);
        wait_new = true;
        Renderer::Invalidated() = true;
    };
    btns->Set(3, 0, eq);

    Label cl = MakeBtn();
    cl->Text = "C";
    cl->Click += [&](Element, MouseEventArgs) {
        input->Text = "  0";
        hint->Text[0] = ' ';
        result = 0;
        wait_new = true;
        Renderer::Invalidated() = true;
    };
    btns->Set(3, 2, cl);


    Grid panel = MakeGrid({ 0 }, { 320, 0 });
    panel->Set(0, 0, btns);
    panel->Set(0, 1, history);

    Grid form = MakeGrid({ 130, 0 }, { 0 });
    form->BackgroundColor = Color::FromARGB(0xE6E6E6);
    form->Set(0, 0, hint_panel);
    form->Set(1, 0, panel);
    Renderer::MainLoop(form);
}
```

此处：

- `add_history` 按引用捕获控件，如前所述，因为此时 `history` 尚且为空。

例：快速构建计算器应用

[快速构建带动画的计算器]

```
#include "include/ImGui.hh"
```

```cpp
#include "system/SystemIO.hh"
#include <iostream>
#include <string>
using namespace easy;

int as_int(std::string s) {
    const char* str = s.c_str() + 2;
    int x = 0;
    bool neg = false;
    if (str[0] == '-') str++, neg = true;
    while (*str) {
        x = 10 * x + *str - '0';
        str++;
    }
    return neg ? -x : x;
}

int main() {

    int result = 0;
    bool wait_new = true;

    Label input, hint;
    OverlapPanel history;

    auto MakeBtn = []() {
        Label btn = MakeLabel();
        btn->SpecSize = { 76, 76 };
        btn->VerticalAlignment = VerticalAlignType::Center;
        btn->HorizontalAlignment = HorizontalAlignType::Center;
        btn->BackgroundColor = Color::FromARGB(0xF0F0F0);
        return btn;
    };

    auto add_history = [&](std::string info) {
        Label item = MakeLabel();
        item->Text = info;
        item->FontHorizontalAlignment = HorizontalAlignType::Left;
        item->SpecSize = { 340, 40 };
        item->Margin.Top = 10;
        item->BeginAnimation(
            Element(item),
            &_Element::BackgroundColor,
            Color::FromARGB(0xE6E6E6),
            Color::FromARGB(0xF0F0F0),
            500,
            EaseInOutCubic
        );
        item->BeginAnimation(
            item,
            &_Label::FontColor,
            Color::FromARGB(0xE6E6E6),
            Colors::Black,
            500,
            EaseInOutCubic
        );
        auto [begin, end] = history->GetRange();
        int total = history->Capacity();
```

```
        for (int i = 0; begin != end; ++i, ++begin) {
            if (i > total - 6)
                (*begin)->BeginAnimation(
                    *begin,
                    &_Element::Margin,
                    (*begin)->Margin,
                    Rect {
                        0, 60 + 50 * (total - 1 - i)
                    },
                    500,
                    EaseInOutCubic
                    );
            else if (i == total - 6) {
                (*begin)->BeginAnimation(
                    std::dynamic_pointer_cast<_Label>(*begin),
                    &_Label::FontColor,
                    Colors::Black,
                    Color::FromARGB(0xE6E6E6),
                    500,
                    EaseInOutCubic
                );
                (*begin)->BeginAnimation(
                    *begin,
                    &_Element::BackgroundColor,
                    (*begin)->BackgroundColor,
                    Color::FromARGB(0xE6E6E6),
                    500,
                    EaseInOutCubic
                );
            }
        }
        history->Add(item);
    };

    using namespace imgui;

    begin_im;
    with (MakeGrid({ 130, 0 }, { 0 })) {
        BackgroundColor = Color::FromARGB(0xE6E6E6);
        with (MakeOverlapPanel()) {
            GridPosition = { 0, 0 };
            with_named (input, MakeLabel()) {
                Margin = { 20 };
                SpecSize = { 700, 70 };
                BackgroundColor = Color::FromARGB(0xF0F0F0);
                FontSize = FontSizeType::Large;
                FontColor = Colors::Black;
                VerticalAlignment = VerticalAlignType::Center;
                HorizontalAlignment = HorizontalAlignType::Left;
                FontHorizontalAlignment = HorizontalAlignType::Left;
                FontVerticalAlignment = VerticalAlignType::Center;
            }
            with_named (hint, MakeLabel()) {
                Margin = { 0, 0, 35 };
                SpecSize = { 0, 70 };
                FontSize = FontSizeType::Large;
                VerticalAlignment = VerticalAlignType::Center;
                HorizontalAlignment = HorizontalAlignType::Right;
```

```
                Text = " ";
            }
        }
        with (MakeGrid({ 0 }, { 320, 0 })) {
            GridPosition = { 0, 1 };

            with_named (history, MakeOverlapPanel()) {
                GridPosition = { 1, 0 };
                Margin = { 50, 0, 50, 50 };
            }

            with (MakeGrid({ 80,80,80,80 }, { 80,80,80,80 })) {
                GridPosition = { 0, 0 };
                Margin = { 20 };

                for (int i = 0; i < 10; ++i) {
                    with (MakeBtn()) {
                        Text = std::to_string(i);
                        Click += [&, i](Element, MouseEventArgs) {
                            if (wait_new) input->Text = "  " +
std::to_string(i), wait_new = false;
                            else input->Text += std::to_string(i);
                            Renderer::Invalidated() = true;
                        };
                        BackgroundColor = Color::FromARGB(0xFAFAFA);
                        if (i) GridPosition = { (i - 1) % 3, (i - 1) / 3 };
                        else GridPosition = { 1, 3 };
                    }
                }

                for (int i = 0; i < 4; ++i) {
                    with (MakeBtn()) {
                        Text = std::string("+-*/").substr(i, 1);
                        Click += [&, i](Element, MouseEventArgs) {
                            hint->Text[0] = "+-*/"[i];
                            wait_new = true;
                            result = as_int(input->Text);
                            Renderer::Invalidated() = true;
                        };
                        GridPosition = { 3, i };
                    }
                }

                with (MakeBtn()) {
                    Text = "=";
                    Click += [&](Element, MouseEventArgs) {
                        std::string info = std::to_string(result);
                        if (hint->Text[0] != ' ') info += hint->Text + input-
>Text.substr(2);
                        else info = input->Text.substr(2);
                        if (hint->Text[0] == '+') input->Text = "  " +
std::to_string(result + as_int(input->Text));
                        else if (hint->Text[0] == '-') input->Text = "  " +
std::to_string(result - as_int(input->Text));
                        else if (hint->Text[0] == '*') input->Text = "  " +
std::to_string(result * as_int(input->Text));
                        else if (hint->Text[0] == '/' && as_int(input->Text))
input->Text = "  " + std::to_string(result / as_int(input->Text));
```

```
                        hint->Text[0] = ' ';
                        result = as_int(input->Text);
                        info += "=" + std::to_string(result);
                        add_history(info);
                        wait_new = true;
                        Renderer::Invalidated() = true;
                    };
                    GridPosition = { 0, 3 };
                }

                with (MakeBtn()) {
                    Text = "C";
                    Click += [&](Element, MouseEventArgs) {
                        input->Text = "  0";
                        hint->Text[0] = ' ';
                        result = 0;
                        wait_new = true;
                        Renderer::Invalidated() = true;
                    };
                    GridPosition = { 2, 3 };
                }
            }
        }
    }
}
```

此处：

- `with_named` 是一个宏，它相较于 `with` 额外将当前控件赋值给指定变量。

其他例子

简单计算器

消消乐