

# 1500D

## Contents

1	CMakeLists.txt	1
2	example/button_example.cxx	1
3	src/Button_widget.hxx	2
4	src/Button_widget.cxx	3
5	src/ISprite_set.hxx	5
6	src/ISprite_set.cxx	7
7	src/Widget.hxx	8
8	src/Widget.cxx	9
9	src/Widget_host.hxx	9
10	src/Widget_host.cxx	11

## 1 CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.3)
2 project(lec15 CXX)
3 include(.cs211/cmake/CMakeLists.txt)
4
5 set(COMMON
6     src/ISprite_set.cxx
7     src/Widget.cxx
8     src/Button_widget.cxx
9     src/Widget_host.cxx)
10
11 include_directories(src)
12
13 add_program(button_example example/button_example.cxx ${COMMON})
14 target_link_libraries(button_example ge211)
```

## 2 example/button\_example.cxx

```
1 #include "../src/Button_widget.hxx"
2 #include "../src/Widget_host.hxx"
3
4 #include <ge211.hxx>
5
6 #include <iostream>
7
8 using namespace widgets;
9
10 static void print_message()
11 {
12     std::cerr << "Thank you\n";
13 }
14
15 static Widget* make_button()
16 {
17     Button_widget* button = new Button_widget("Click Me");
18     button->on_click(print_message);
19     return button;
20 }
21
22 int main()
23 {
24     Widget_host(make_button).run();
25 }
```

## 3 src/Button\_widget.hxx

```
1 #pragma once
2
3 #include "Widget.hxx"
4
5 namespace widgets
6 {
7
8     class Button_widget : public Widget
9     {
10     public:
11         explicit Button_widget(const std::string& title,
12                               int font_size = 20);
13     }
```

#### 4 src/Button\_widget.cxx

```
14     void on_click(std::function<void()>);
15
16 protected:
17     ge211::Dimensions dimensions() const override;
18
19     void draw(ISprite_set& set, Mouse_state) const override;
20
21     bool click(ge211::Position position) override;
22
23 private:
24     struct Look
25     {
26         Look(const std::string&,
27             const ge211::Font&,
28             const Color_pair&);
29
30         void draw(ISprite_set&) const;
31
32         ge211::Text_sprite fg;
33         ge211::Rectangle_sprite bg;
34     };
35
36     ge211::Font font_;
37     Look normal_, hover_, active_;
38
39     std::function<void()> on_click_ = [] { };
40 };
41
42 } // end namespace widgets
```

#### 4 src/Button\_widget.cxx

```
1 #include "Button_widget.hxx"
2
3 using namespace ge211;
4 using namespace widgets;
5
6 static const char* const font_name{"sans.ttf"};
7
8 static const Dimensions padding{10, 10};
9
10 static const Color gray{192, 192, 192},
11     black{0, 0, 0},
```

```

12         nu_purple{59, 47, 119};
13
14     static const Color_pair normal_colors{black, gray},
15         hover_colors{nu_purple, gray},
16         active_colors{gray, nu_purple};
17
18     Button_widget::Look::Look(const std::string& text,
19         const ge211::Font& font,
20         const Color_pair& colors)
21         : fg{Text_sprite::Builder(font).color(colors.fg).message(text).build()},
22         bg{fg.dimensions() + 2 * padding, colors.bg}
23     { }
24
25     void Button_widget::Look::draw(ISprite_set& set) const
26     {
27         Position fg_pos{(bg.dimensions() - fg.dimensions()) / 2};
28
29         set.add_sprite(bg, {0, 0}, 0);
30         set.add_sprite(fg, fg_pos, 1);
31     }
32
33     Button_widget::Button_widget(const std::string& title,
34         int font_size)
35         : font_{font_name, font_size},
36         normal_{title, font_, normal_colors},
37         hover_{title, font_, hover_colors},
38         active_{title, font_, active_colors}
39     { }
40
41     void Button_widget::on_click(std::function<void()> on_click)
42     {
43         on_click_ = on_click;
44     }
45
46     Dimensions Button_widget::dimensions() const
47     {
48         return normal_.bg.dimensions();
49     }
50
51     bool Button_widget::click(ge211::Position position)
52     {
53         on_click_();
54         return true;
55     }
56

```

```

57 void Button_widget::draw(ISprite_set& set, Mouse_state st) const
58 {
59     switch (st) {
60     case Mouse_state::normal:
61         normal_.draw(set);
62         break;
63     case Mouse_state::hover:
64         hover_.draw(set);
65         break;
66     case Mouse_state::active:
67         active_.draw(set);
68         break;
69     }
70 }

```

## 5 src/ISprite\_set.hxx

```

1  #pragma once
2
3  #include <ge211.hxx>
4
5  namespace widgets {
6
7  /// Interface (abstract class) for allowing different classes
8  /// that are used like `Sprite_set`s.
9  struct ISprite_set
10 {
11     /// Adds the given sprite with the given position, z index,
12     /// and transformation.
13     virtual void add_sprite(ge211::Sprite const&,
14                             ge211::Position,
15                             int z,
16                             ge211::Transform) = 0;
17
18     /// Adds the given sprite with the given position and optional
19     /// z index.
20     virtual void add_sprite(ge211::Sprite const&,
21                             ge211::Position,
22                             int z = 0);
23 };
24
25 /// Wraps a `ge211::Sprite_set` so that we can use it as an
26 /// `ISprite_set`.

```

```

27 class Sprite_set_adapter : public ISprite_set
28 {
29 public:
30     explicit Sprite_set_adapter(ge211::Sprite_set&);
31
32     void add_sprite(ge211::Sprite const&,
33                   ge211::Position,
34                   int z,
35                   ge211::Transform) override;
36
37 private:
38     ge211::Sprite_set& adaptee_;
39 };
40
41 /// The data associated with a sprite placement (used by the next two
42 /// classes).
43 struct Placed_sprite
44 {
45     const ge211::Sprite* sprite;
46     ge211::Position position;
47     int z;
48     ge211::Transform transform;
49 };
50
51 /// For testing, a transparent sprite set that just collects all
52 /// the sprites so we can look at them.
53 struct Mock_sprite_set : ISprite_set
54 {
55     void add_sprite(ge211::Sprite const&,
56                   ge211::Position,
57                   int z,
58                   ge211::Transform) override;
59
60     std::vector<Placed_sprite> sprites;
61 };
62
63 /// A sprite set that can add its sprites to another sprite set
64 /// at a specified offset (position and z index).
65 class Sprite_set_layer : public ISprite_set
66 {
67 public:
68     void add_sprite(ge211::Sprite const&,
69                   ge211::Position,
70                   int z,
71                   ge211::Transform) override;

```

## 6 src/ISprite\_set.cxx

```
72
73     /// Adds the sprites in this layer to `set`, offsetting them
74     /// so their lowest z index is the given `z` and translating
75     /// them by `offset`. Returns the highest z index used.
76     int add_to(ISprite_set& set,
77               ge211::Dimensions xyoffset = {0, 0},
78               int zoffset = 0) const;
79
80 private:
81     std::vector<Placed_sprite> sprites_;
82     int zmin_ = 0, zmax_ = 0;
83 };
84
85 } // end namespace widgets
```

## 6 src/ISprite\_set.cxx

```
1  #include "ISprite_set.hxx"
2
3  using namespace ge211;
4  using namespace widgets;
5
6  void ISprite_set::add_sprite(Sprite const& sprite,
7                               Position position,
8                               int z)
9  {
10     add_sprite(sprite, position, z, Transform{});
11 }
12
13 Sprite_set_adapter::Sprite_set_adapter(Sprite_set& adaptee)
14     : adaptee_{adaptee}
15 { }
16
17 void Sprite_set_adapter::add_sprite(Sprite const& sprite,
18                                     Position position,
19                                     int z,
20                                     Transform transform)
21 {
22     adaptee_.add_sprite(sprite, position, z, transform);
23 }
24
25 void Mock_sprite_set::add_sprite(Sprite const& sprite,
26                                   Position position,
```

## 7 src/Widget.hxx

```
27         int z,
28         Transform transform)
29 {
30     sprites.push_back({&sprite, position, z, transform});
31 }
32
33 void Sprite_set_layer::add_sprite(Sprite const& sprite,
34     Position position,
35     int z,
36     Transform transform)
37 {
38     zmin_ = std::min(zmin_, z);
39     zmax_ = std::min(zmax_, z);
40     sprites_.push_back({&sprite, position, z, transform});
41 }
42
43 int Sprite_set_layer::add_to(ISprite_set& set,
44     Dimensions xyoffset,
45     int z) const
46 {
47     int zoffset = z - zmin_;
48     int zfinal  = z + zmax_;
49
50     for (const auto& placed : sprites_)
51         set.add_sprite(*placed.sprite,
52             placed.position + xyoffset,
53             placed.z + zoffset,
54             placed.transform);
55
56     return zfinal;
57 }
```

## 7 src/Widget.hxx

```
1 #pragma once
2
3 #include <functional>
4 #include <ISprite_set.hxx>
5
6 namespace widgets {
7
8     class Widget_host;
9 }
```



## 8 src/Widget.cxx

```
10 class Widget
11 {
12     friend Widget_host;
13
14 public:
15     virtual ~Widget() {}
16
17     enum class Mouse_state
18     {
19         normal,
20         hover,
21         active,
22     };
23
24 protected:
25     /// Returns the dimensions of the widget.
26     virtual ge211::Dimensions dimensions() const = 0;
27
28     /// Adds the widget to a sprite set. Gets the current state
29     /// of the mouse so that this can affect how it looks.
30     virtual void draw(ISprite_set&, Mouse_state) const = 0;
31
32     /// Override this to respond to clicks. Returning `true` if
33     /// the click is accepted by the widget, or false to allow the
34     /// click to pass through this widget.
35     virtual bool click(ge211::Position) { return false; }
36 };
37
38 struct Color_pair
39 {
40     ge211::Color fg, bg;
41 };
42
43 } // end namespace widgets
```

## 8 src/Widget.cxx

```
1 #include "Widget.hxx"
2
3 using namespace ge211;
4 using namespace widgets;
```

## 9 src/Widget\_host.hxx

```

1  #pragma once
2
3  #include "Widget.hxx"
4
5  #include <ge211.hxx>
6
7  #include <memory>
8
9  namespace widgets {
10
11  /// Attaches a widget to a GE211 window.
12  class Widget_host : public ge211::Abstract_game
13  {
14  public:
15      /// Takes ownership of the `Widget*`.
16      explicit Widget_host(std::function<Widget*()>);
17
18      /// Since `root_` is owned and will be deleted by the destructor,
19      /// it wouldn't be safe to copy a `Widget_host`, so we disable
20      /// the copy constructor.
21      Widget_host(const Widget_host&) = delete;
22
23      /// Disable the copy-assignment operator for the same reason.
24      Widget_host& operator=(const Widget_host&) = delete;
25
26      /// Deletes the root widget.
27      ~Widget_host() override;
28
29  protected:
30      void draw(ge211::Sprite_set&) override;
31
32      void on_mouse_down(ge211::Mouse_button, ge211::Position) override;
33
34      void on_mouse_up(ge211::Mouse_button, ge211::Position) override;
35
36      void on_mouse_move(ge211::Position) override;
37
38  private:
39      bool is_position_inside_(ge211::Position) const;
40      ge211::Rectangle bbox_() const;
41      Widget& root_() const;
42      Widget::Mouse_state mouse_state_() const;

```

```

43
44     std::function<Widget*> create_;
45     mutable Widget* root_ptr_ = nullptr;
46     bool is_mouse_inside_ = false;
47     bool is_mouse_down_   = false;
48 };
49
50 } // end namespace widgets

```

## 10 src/Widget\_host.cxx

```

1  #include "Widget_host.hxx"
2
3  using namespace ge211;
4  using namespace widgets;
5
6  Widget_host::Widget_host(std::function<Widget*> create)
7      : create_{create}
8  { }
9
10 Widget_host::~Widget_host()
11 {
12     delete root_ptr_;
13 }
14
15 void Widget_host::draw(Sprite_set& set)
16 {
17     Sprite_set_layer layer;
18     root_.draw(layer, mouse_state_());
19
20     Sprite_set_adapter adapter{set};
21     Rectangle bbox{bbox_()};
22     layer.add_to(adapter, {bbox.x, bbox.y});
23 }
24
25 void Widget_host::on_mouse_down(Mouse_button button, Position posn)
26 {
27     if (button == Mouse_button::left)
28         is_mouse_down_ = true;
29 }
30
31 void Widget_host::on_mouse_up(Mouse_button button, Position posn)
32 {

```

```

33     if (button == Mouse_button::left) {
34         if (is_mouse_inside_ && is_mouse_down_)
35             root_().click(posn);
36
37         is_mouse_down_ = false;
38     }
39 }
40
41 void Widget_host::on_mouse_move(Position posn)
42 {
43     is_mouse_inside_ = is_position_inside_(posn);
44 }
45
46 static bool is_position_inside(Position posn, Rectangle rect)
47 {
48     auto limit = rect.bottom_right();
49     return rect.x <= posn.x && posn.x < limit.x &&
50         rect.y <= posn.y && posn.y < limit.y;
51 }
52
53 bool Widget_host::is_position_inside_(Position posn) const
54 {
55     return is_position_inside(posn, bbox_());
56 }
57
58 Widget& Widget_host::root_() const
59 {
60     if (!root_ptr_) root_ptr_ = create_();
61     return *root_ptr_;
62 }
63
64 Rectangle Widget_host::bbox_() const
65 {
66     Dimensions window = get_window().get_dimensions();
67     Dimensions widget = root_().dimensions();
68     Position top_left{(window - widget) / 2};
69     return Rectangle::from_top_left(top_left, widget);
70 }
71
72 Widget::Mouse_state Widget_host::mouse_state_() const
73 {
74     using M = Widget::Mouse_state;
75
76     if (is_mouse_inside_)
77         if (is_mouse_down_)

```

```
78         return M::active;
79     else
80         return M::hover;
81 else
82     return M::normal;
83 }
```