Week 9

UI

Hooman Salamat

2018-03-31

Objectives

- Implement screens and menus
- Design a UI component hierarchy
- Implement containers, labels and buttons

Menus

GUI

- In most cases, the mouse will be used as an input source
  - Triggers button clicks or mouse-overs
- In the ongoing example, the keyboard is used to change menu options
- A GUI framework will be created and the first step is to reserve/create a namespace called `GUI`
- In this namespace we are defining a class called `Component` as shown on the next slide

Component Class

```
namespace GUI
{
    class Component : public sf::Drawable
                    , public sf::Transformable
                    , private sf::NonCopyable
    {
    public:
       typedef std::shared_ptr<Component> Ptr;
    public:
        Component();
        virtual ~Component();
        virtual bool isSelectable() const = 0;
        bool isSelected() const;
        virtual void select();
        virtual void deselect();
        virtual bool isActive() const;
        virtual void activate();
        virtual void deactivate();
        virtual void handleEvent(const sf::Event& event) = 0;
    private:
        bool mIsSelected;
        bool mIsActive;
    };
}
```

Other Classes

- Other classes we define:
  - GUI::Container
  - GUI::Button
  - GUI::Label

- You might recognize some of these as very common GUI types
- They are the most basic components that you will need
  - We can expand the system with more components later

Container Class

```
Container::Container() : mChildren(), mSelectedChild(-1)
{ }

void Container::pack(Component::Ptr component)
{
   mChildren.push_back(component);
   if (!hasSelection() && component->isSelectable())
       select(mChildren.size() - 1);
}

bool Container::isSelectable() const
{
   return false;
}
```

Container Class (cont'd.)

```
void Container::handleEvent(const sf::Event& event)
{
    if (hasSelection() && mChildren[mSelectedChild]->isActive())
    {
        mChildren[mSelectedChild]->handleEvent(event);
    }
    else if (event.type == sf::Event::KeyReleased)
    {
        if (event.key.code == sf::Keyboard::W || event.key.code == sf::Keyboard::Up)
        {
            selectPrevious();
        }
        else if (event.key.code == sf::Keyboard::S || event.key.code == sf::Keyboard::Down)
        {
            selectNext();
        }
        else if (event.key.code == sf::Keyboard::Return || event.key.code == sf::Keyboard::Space)
        {
            if (hasSelection())
                    mChildren[mSelectedChild]->activate();
        }
    }
}
```

Container Class (cont'd.)

```
void Container::select(std::size_t index)
{
    if (mChildren[index]->isSelectable())
    {
        if (hasSelection())
        mChildren[mSelectedChild]->deselect();
        mChildren[index]->select();
        mSelectedChild = index;
    }
}
void Container::selectNext()
{
    if (!hasSelection())
       return;
    // Search next component that is selectable
    int next = mSelectedChild;
    do
       next = (next + 1) % mChildren.size();
    while (!mChildren[next]->isSelectable());
    // Select that component
    select(next);
}
```

Container Class (cont'd.)

```
void Container::selectPrevious()
{
   if (!hasSelection())
      return;
   // Search previous component that is selectable
   int prev = mSelectedChild;
   do
      prev = (prev + mChildren.size() - 1) % mChildren.size();
   while (!mChildren[prev]->isSelectable());
   // Select that component
   select(prev);
}
```

Label Class

```
Label::Label(const std::string& text, const FontHolder& fonts)
: mText(text, fonts.get(Fonts::Label), 16)
{
}

bool Label::isSelectable() const
{
   return false;
}

void Label::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
   states.transform *= getTransform();
   target.draw(mText, states);
}

void Label::setText(const std::string& text)
{
   mText.setString(text);
}
```

Button Class

```
Button::Button(const FontHolder& fonts, const TextureHolder& textures)
// ...
{
    mSprite.setTexture(mNormalTexture);
    mText.setPosition(sf::Vector2f(mNormalTexture.getSize() / 2u));
}

bool Button::isSelectable() const
{
    return true;
}

void Button::select()
{
    Component::select();
    mSprite.setTexture(mSelectedTexture);
}

void Button::deselect()
{
    Component::deselect();
    mSprite.setTexture(mNormalTexture);
}
```

Button Class (cont'd.)

```
void Button::activate()
{
    Component::activate();
    if (mIsToggle)
       mSprite.setTexture(mPressedTexture);
    if (mCallback)
       mCallback();
    if (!mIsToggle)
       deactivate();
}

void Button::deactivate()
{
    Component::deactivate();
    if (mIsToggle)
    {
       if (isSelected())
          mSprite.setTexture(mSelectedTexture);
       else
          mSprite.setTexture(mNormalTexture);
    }
}
```

Updating the Menu

```
MenuState::MenuState(StateStack& stack, Context context)
: State(stack, context), mGUIContainer()
{
   ...
   auto playButton = std::make_shared<GUI::Button>(
      *context.fonts, *context.textures);
   playButton->setPosition(100, 250);
   playButton->setText("Play");
   playButton->setCallback([this] ()
   {
      requestStackPop();
      requestStackPush(States::Game);
   });
   mGUIContainer.pack(playButton);
}
```

Updating the Menu (cont'd.)

```
void MenuState::draw()
{
   sf::RenderWindow& window = *getContext().window;
   window.setView(window.getDefaultView());
   window.draw(mBackgroundSprite);
   window.draw(mGUIContainer);
}

bool MenuState::update(sf::Time)
{
   return true;
}

bool MenuState::handleEvent(const sf::Event& event)
{
   mGUIContainer.handleEvent(event);
   return false;
}
```

SettingsState

```
SettingsState::SettingsState(StateStack& stack, Context context)
: State(stack, context)
, mGUIContainer()
{
    mBackgroundSprite.setTexture(
        context.textures->get(Textures::TitleScreen));
    mBindingButtons[Player::MoveLeft] =
        std::make_shared<GUI::Button>(...);
    mBindingLabels[Player::MoveLeft] =
        std::make_shared<GUI::Label>(...);
    ... // More buttons and labels
    updateLabels();
    auto backButton = std::make_shared<GUI::Button>(...);
    backButton->setPosition(100, 375);
    backButton->setText("Back");
    backButton->setCallback([this] ()
    {
        requestStackPop();
    });
    mGUIContainer.pack(mBindingButtons[Player::MoveLeft]);
    mGUIContainer.pack(mBindingLabels[Player::MoveLeft]);
    ...
    mGUIContainer.pack(backButton);
}
```

SettingsState (cont'd.)

```
void SettingsState::updateLabels()
{
    Player& player = *getContext().player;
    for (std::size_t i = 0; i < Player::ActionCount; ++i)
    {
        sf::Keyboard::Key key =
            player.getAssignedKey(static_cast<Player::Action>(i));
        mBindingLabels[i]->setText(toString(key));
    }
}
```

SettingsState (cont'd.)

```
bool SettingsState::handleEvent(const sf::Event& event)
{
    bool isKeyBinding = false;
    for (std::size_t action = 0; action < Player::ActionCount; ++action)
    {
        if (mBindingButtons[action]->isActive())
        {
            isKeyBinding = true;
            if (event.type == sf::Event::KeyReleased)
            {
              getContext().player->assignKey (static_cast<Player::Action>(action),
                event.key.code);
              mBindingButtons[action]->deactivate();
            }
            break;
        }
    }
    if (isKeyBinding)
        updateLabels();
    else
        mGUIContainer.handleEvent(event);
    return false;
}
```