Interfaz de usuario: Tema 6

Máster en Programación de Videojuegos

Ignacio Martínez Rodríguez

Curso 2013-2014

Contenido

Tema 6: Arquitectura lógica de una GUI

- Enfoque
- Diagrama de clases
- Control base
- Listeners
- Mensaje.
- Manager.
- Caso práctico: un botón
- Práctica

Enfoque

- Nuestra GUI consiste principalmente en una colección de controles, tan extensa como necesitemos (ventanas, botones, checkboxes, labels, sliders…)
- Estos controles se agrupan en jerarquías, de tal manera que un control siempre es hijo de otro.
- Estos controles comparte muchas funcionalidades, por tanto antes de empezar a crear controles pensemos en qué atributos tienen en común:

  - Posición
  - Tamaño
  - Nombre
  - Está activo?
  - Es visible?
  - Tiene el foco?
  - Control padre
  - Controles hijos
  - Recepción de eventos

Enfoque

- Esto si lo pasamos al mundo de las clases quiere decir que podemos crear una clase base para nuestros controles, de la cual heredarán todos los controles que creemos.
- Al ser una clase base, no tiene sentido que se pueda instanciar, así que será una clase abstracta.

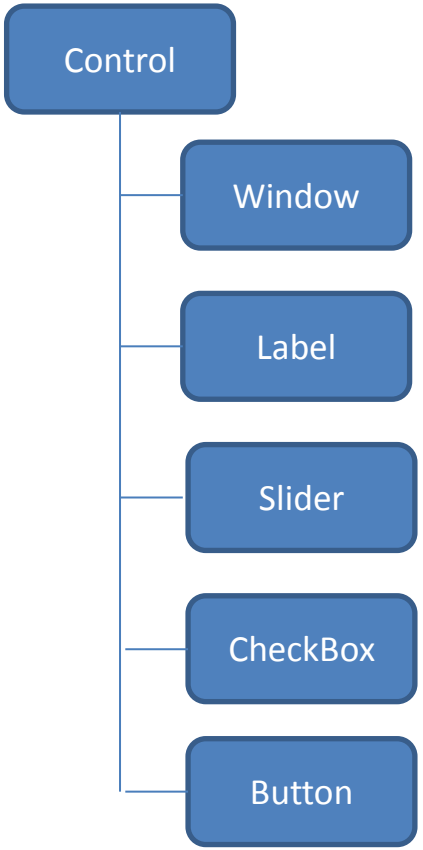
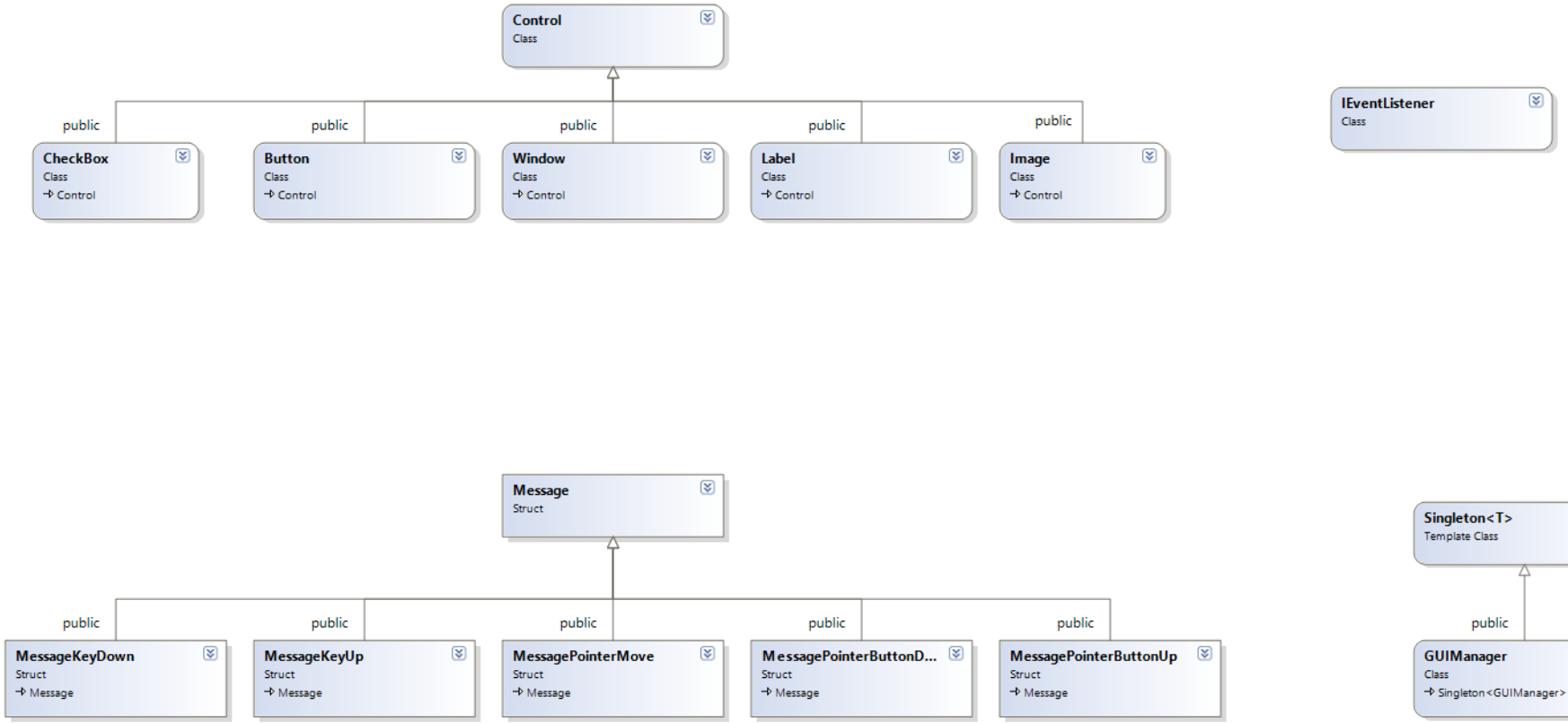


Diagrama de clases



Control base

- **Control**: es la clase base de la que heredan todos los controles.

- Métodos importantes:

```
//  Input del usuario que vendrá de nuestro InputManager.
void injectInput( const Message& message );

// Cada tipo de control tendrá su método de actualización.
virtual void update() = 0;

// Cada tipo de control implentará su render.
virtual void render() = 0;

// Se implemetará como se responde a los eventos de entrada para cada tipo de control.
virtual void onInputEvent( const Message& message ) = 0;

// Establecemos a donde notifican los controles sus eventos.
void setEventListener( IEventListener* eventListener );
```

Listeners

- **IEventListener**: es una interfaz que tienen que implementar las clases que quieran recibir notificaciones de un control.
- Métodos importantes:

```
virtual void onClick( Control* sender );
virtual void onGotFocus( Control* sender );
virtual void onLostFocus( Control* sender );
virtual void onKeyDown( Control* sender, int keyCode );
virtual void onKeyUp( Control* sender, int keyCode );
```

- Cada control será responsable de notificar a sus Listeners de los eventos pertenentes.

Mensajes

- **Message**: es una clase base para los mensajes internos de la GUI.
- Estas clases encapsularán eventos de ratón, teclado, joystick…
- Ejemplo:

```
struct MessagePointerMove : public Message
{
  MessagePointerMove()
  {
    type = mtPointerMove;
  }

  MessagePointerMove( float _x, float _y)
  {
    type = mtPointerMove;
    x = _x;
    y = _y;
  }

  float x;
  float y;
};
```

Manager

- **GUIManager**: es la clase desde la que se centralizan los procesos de la GUI.

- Métodos importantes:

```
// Establece el control padre o «desktop»
void setRootControl( Control* control );

// Llama a los updates de todos los controles
void update();

// Llama a los render de todos los controles
void render();

// Punto de entrada de del input; desde aquí se enruta hacia el control adecuado
void injectInput( const Message& message );
```

Caso práctico: un botón

- Para crear un **botón** necesitaremos los siguientes elementos:
  - Gráfico de botón en estado normal.
  - Gráfico de botón en estado pulsado.
  - (Opcional) Gráfico de botón deshabilitado.
  - (Opcional) Texto a pintar en el botón.

Caso práctico: un botón

- **Declaración** de la clase **Button**

```
class Button : public Control
{
public:
  Button();

  bool init(const string name, const Vector2& position, const string& normalImage, const string& pushImage);

  void update();
  void render();
  virtual void onInputEvent( const Message& message );

protected:
  Image* m_normalImage;
  Image* m_pushImage;
  bool m_pushed;
};
```

Caso práctico: un botón

- **Implementación**
- En el constructor inicializamos las variables a sus valores por defecto.

```
Button::Button()
{
  m_pushed = false;
}
```

- Inicialización del botón:

```
bool Button::init(const string name, const Vector2& position, const string& normalImage, const string& pushImage)
{
  m_name= name;
  m_position = position;
  m_normalImage = new Image( normalImage.c_str() );
  m_pushImage = new Image( pushImage.c_str() );
  m_size = Vector2( (float)m_normalImage->getWidth(), (float)m_normalImage->getHeight() );

  return true;
}
```

Caso práctico: un botón

- Tratamiento de eventos de input:

```
void Button::onInputEvent( const Message& message )
{
  switch( message.type )
  {
    case mtPointerButtonDown:
      m_pushed = true;
      break;

    case mtPointerButtonUp:
      if( m_pushed )
      {
        list<IEventListener*>::const_iterator it = m_eventListeners.begin();
        while( it != m_eventListeners.end() )
        {
          IEventListener *listener = *it;
          listener->onClick();
          it++;
        }
      }
      m_pushed = false;
      break;
  }
}
```

Caso práctico: un botón

- Render

```
void Button::render()
{
  if( m_visible )
  {
    Vector2 pos = getAbsolutePosition();
    Renderer::instance().setBlendingMode(Renderer::ALPHA );

    if( m_pushed )
    {
      Renderer::instance().drawImage( m_pushImage, pos.x, pos.y );
    }
    else if( !m_pushed )
    {
      Renderer::instance().drawImage(m_normalImage, pos.x, pos.y );
    }
}
}
```

Práctica 3

- Añadir las siguientes características al botón:
  - Que se le pueda poner un texto
  - Que pueda aparecer deshabilitado, con el gráfico correspondiente y que no responda al input del usuario si lo está.

Práctica 3

- Crear los menús para un juego típico:
  - **Pantalla principal** con los siguientes botones:
    - **[COMENZAR PARTIDA]**: nos llevará a una pantalla que no hará nada y de la que podremos salir con ESC para volver al menú principal.
    - **[CONFIGURACIÓN]:** nos lleva a la pantalla de configuración.
    - **[CRÉDITOS]:** nos lleva a la pantalla de créditos.
    - **[SALIR]**: sale de la aplicación, preguntando antes al usuario en una ventana popup si está seguro de que desea salir (**[Si]** o **[No]**).
  - **Pantalla de configuración**:
    - **Modo «gore»**: on/off
    - **Partículas**: on/off
    - **Autoguardado**: on/off
    - ****Opcional: Volumen**: 0-100
    - **[Volver]**

Práctica 3

- **Pantalla de créditos**:
  - **Texto de créditos centrado**
  - **[Volver] :** botón para regresar a la pantalla principal

- **NOTAS**: será necesario crear los siguientes controles para esta práctica:
  - un control **Label** que sirva para pintar textos entre otras cosas el texto de los botones o cualquier otro elemento que tenga.
  - Un control **Checkbox** para la configuración
  - Si se quiere hacer el volumen (opcional) deberá ser un **Slider**.