IMGUI

Jetro Lauha - www.jet.ro

Jari Komppa – www.iki.fi/sol

2007-08-02
ASSEMBLY Summer'07

The What?

Immediate Mode Graphical User Interface

(as opposed to “retained mode”)

“My First UI”

```
printf(“Are you sure? [y/N] “);
fflush(stdout);

if (toupper(getch()) == 'Y')
    exit(0);
```

“My First GUI”

Button

```
FillRect(50, 50, 100, 30);
DrawText(80, 55, “Quit”);

if (mouseButtonDown &&
    mx >= 50 && mx <= 150 &&
    my >= 50 && my <= 80)
    exit(0);
```

“My First GUI”

Slider

```
FillRect(80, 20, 100, 5);
x = 80 + position * 100;
DrawLine(80 + x, 15, 80 + x, 30);

if (mouseButtonDown &&
    mx >= 80 && mx <= 180 &&
    my >= 15 && my <= 30)
    position = (mx - 80) / 100;
```

That Was Simple, But...

- Doesn't behave like proper UI components
- So you typically have a system with...
  - component hierarchy, lifetime management, data synchronization, event handler (loop), event listeners, layouters, rendering, ... and so on ...
- Usage isn't much easier than complexity of the system itself
- GUI development turned into a “retained” model

Small IMGUI example..

```
if (button(GEN_ID, 15, 15, “Quit”))
{
    exit(0);
}
```

Anatomy of a button

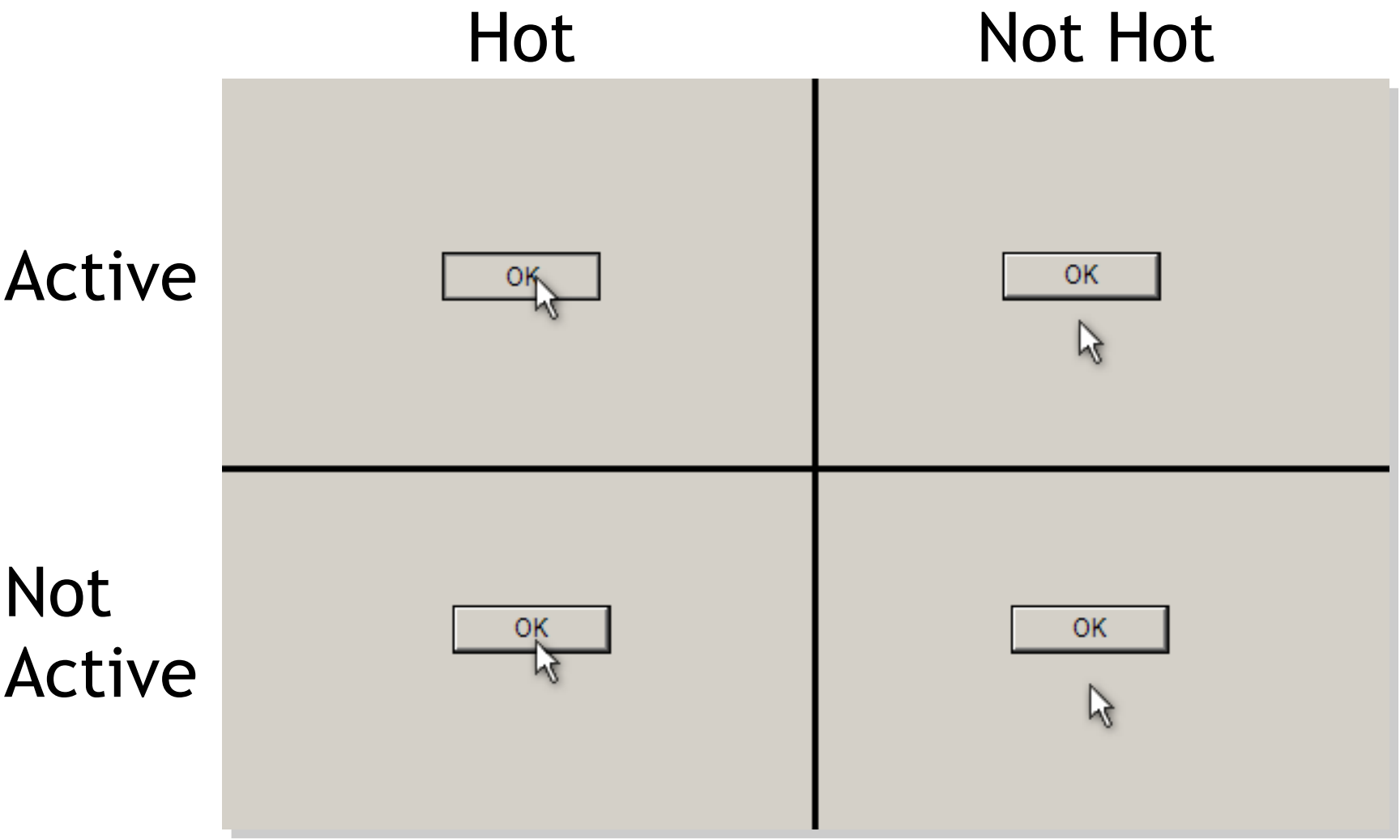
```
bool button(int id, int x, int y, char *text)
{
    // Check whether the button should be hot
    if (mouseInsideRectangle(x, y, strlen(text) * 8 + 16, 48))
    {
        uiState.hotItem = id;
        if (uiState.activeItem == 0 && uiState.mouseDown)
            uiState.activeItem = id;
    }

    ... // Render button

    // If button is hot and active, but mouse button is not
    // down, the user must have clicked the button.
    if (uiState.mouseDown == 0 &&
        uiState.hotItem == id &&
        uiState.activeItem == id) return true;

    return false; // Otherwise, no clicky.
}
```

Are you hot or not?



uiState

```
bool button(int id, int x, int y, char *text)
{
    // Check whether the button should be hot
    if (mouseInsideRectangle(x, y, strlen(text) * 8 + 16, 48))
    {
        uiState.hotItem = id;
        if (uiState.activeItem == 0 && uiState.mouseDown)
            uiState.activeItem = id;
    }

    ... // Render button

    // If button is hot and active, but mouse button is not
    // down, the user must have clicked the button.
    if (uiState.mouseDown == 0 &&
        uiState.hotItem == id &&
        uiState.activeItem == id) return true;

    return false; // Otherwise, no clicky.
}
```

UI State (simple case)

```
struct UIState
{
  int mouseX;
  int mouseY;
  int mouseDown;

  int hotItem;
  int activeItem;
}
uiState;
```

```
void beginGUI()
    - clear hotItem

void endGUI()
    - clear activeItem
      (if needed)
```

id

```
bool button(int id, int x, int y, char *text)
{
    // Check whether the button should be hot
    if (mouseInsideRectangle(x, y, strlen(text) * 8 + 16, 48))
    {
        uiState.hotItem = id;
        if (uiState.activeItem == 0 && uiState.mouseDown)
            uiState.activeItem = id;
    }

    ... // Render button

    // If button is hot and active, but mouse button is not
    // down, the user must have clicked the button.
    if (uiState.mouseDown == 0 &&
        uiState.hotItem == id &&
        uiState.activeItem == id) return true;

    return false; // Otherwise, no clicky.
}
```

GEN_ID

- IDs must be unique for all active widgets
- Many solutions
  - __LINE__
  - Widget's rectangle
  - Incrementing variable
  - Etc.
- All solutions have good and bad sides
- Keep It Simple, Stupid!

The good..

- No object creation
- No cleanup either
- No queries for information
- No message passing
- Data owned by application, not the widget
- Everything is “immediate” - one call per widget, each frame, handles behavior and rendering.

..the bad..

- Requires different kind of thinking
- Wastes CPU time
  - But in games you're re-rendering stuff 50+ fps anyway..
- UI generated from code; No designer-friendly tools.
  - *Unless you make some...*

..and the ugly.

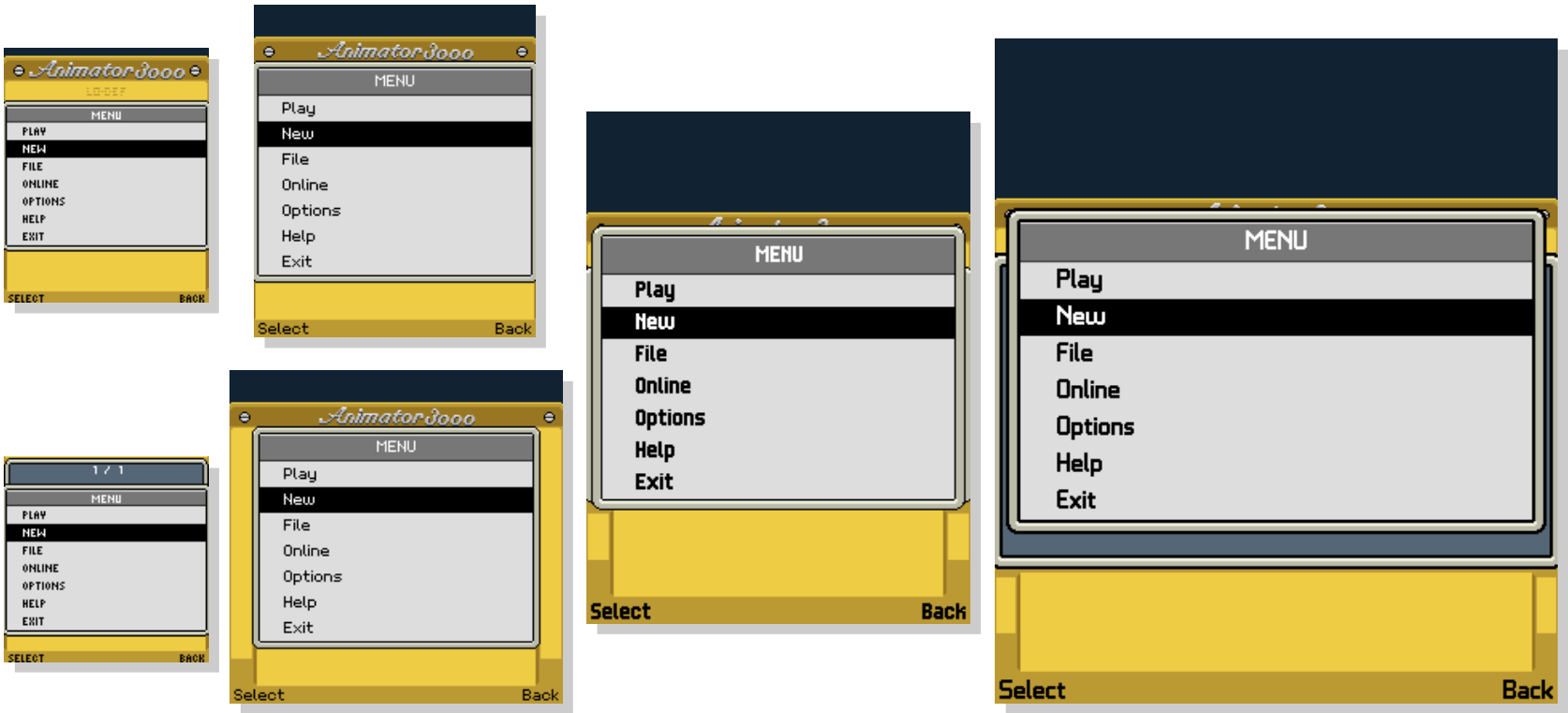
- While making easy things dead easy, makes complicated things very complicated.
  - The UI system *internals* may become even more complex than in “traditional” GUI library!
- UI logic interleaved to rendering
  - Can be overcome by more complex internals.
- Pretty “anti-OOP” (although this is debatable)
- Not a silver bullet.

Case Studies

- IMGUI with J2ME
  - *Habbo Animator*, yet to be released project by Sulake Corporation
  - Works on wide set of J2ME devices with very limited resources
- IMGUI with PS3
  - *Super Stardust™ HD* - created by Housemarque
  - For PLAYSTATION®3
    - Available now in PlayStation Network

Case: IMGUI with J2ME

- Scales dynamically from tiny to big resolutions
  - All resolutions supported by the *same build*



Case: IMGUI with J2ME

- Key-based actions
  - Key presses saved to a ring buffer
- Screen & focus managing by framework
  - Application screens have an enumerated type
    - Focus remembered for each screen type
  - Focus reset when entering screen, but recalled when returning to it
  - Likewise component types are enumerated
    - Some data saved per type, e.g. scroll position

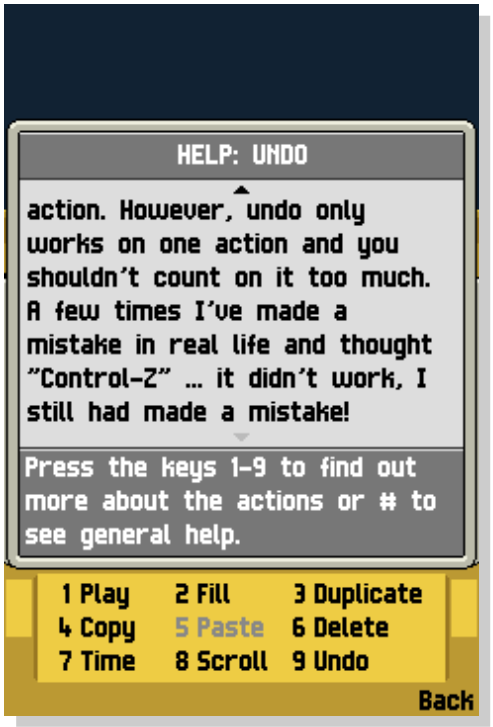
Case: IMGUI with J2ME

- List-based UI component
  - listBegin(listId, …),
    listButton(listId, index, …), … ,
    listEnd(listId, itemCount, …)
  - Draw customizations after
    calling listButton

  - Handles all pending movements from key buffer
    - Better usability on very slow devices where FPS is lower
  - Draws arrows to indicate scrolling possibility
    - In listEnd(), as itemCount is then known

Case: IMGUI with J2ME

- Scroll panels
  - Given rectangle, font and text...
    - Text printed with word wrapping, also amount of rows counted for the scrolling arrows
  - Amount of visible rows in the rectangle is reduced with lower resolutions

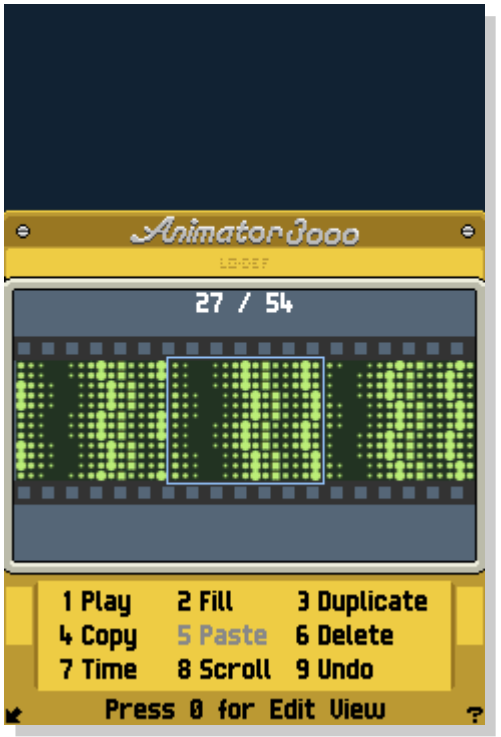


Case: IMGUI with J2ME

- Multi-tap text input
  - Manages the key tap timeouts, current key and char index
  - Component given a text editing temp array and characters for each key
- Other things
  - Timed out pop-ups for notifications
    - If key is pressed, it is consumed and popup is dismissed
  - Shortcut support for menus (expert mode)

Case: IMGUI with J2ME

- Post mortem observations
  - Implementation of the framework was slightly harder than previous non-IMGUI one
    - Both had same design constraints
  - IMGUI a bit easier for new screens
    - Clearly better for dynamic stuff
  - No significant difference in memory usage compared to previous system
    - Probably slightly less separate objects overall

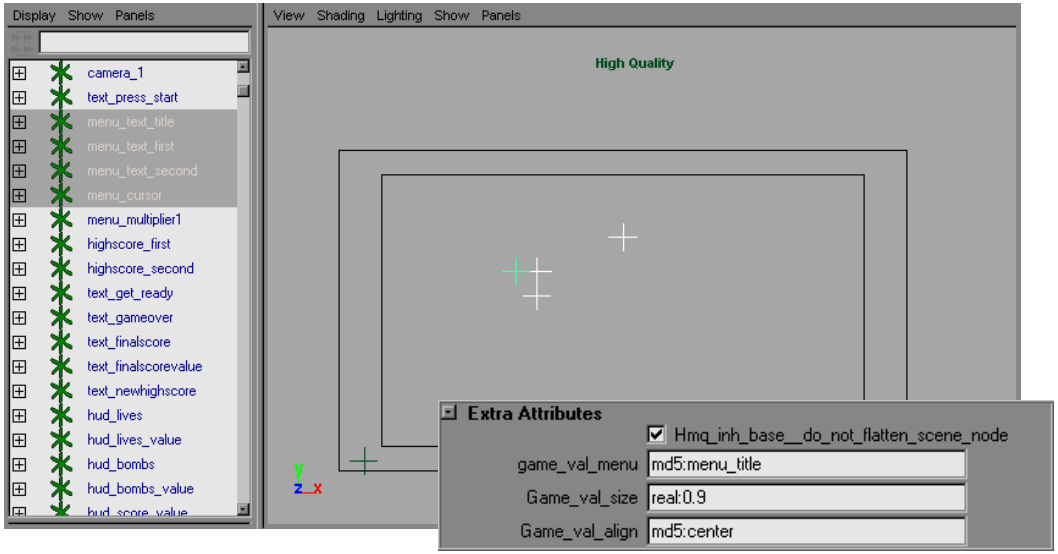


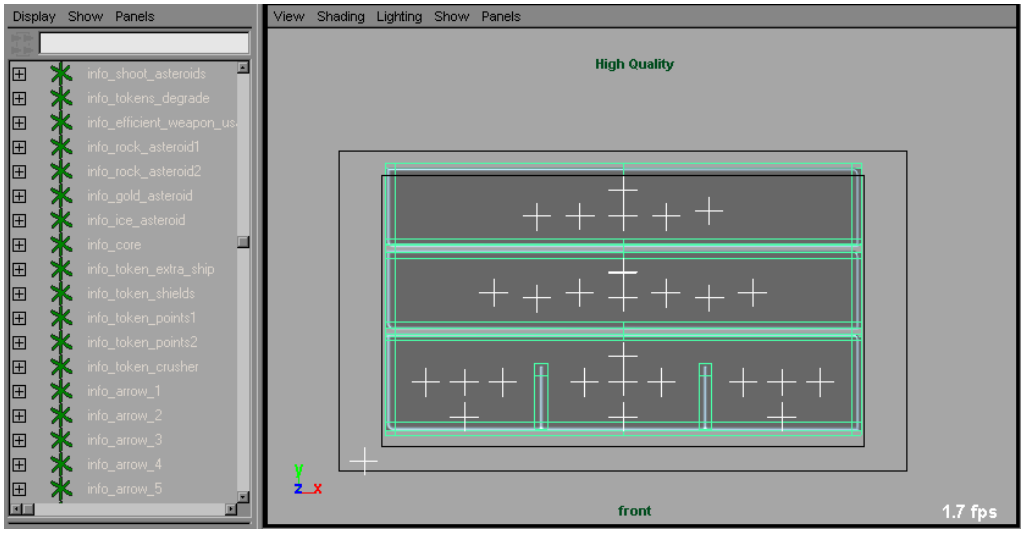
Case: IMGUI with PS3 - SSHD

Case: IMGUI with PS3 - SSHD

- Designers define UI using locators
  - Position, size, text alignment, custom attributes
  - Explicit id defined for each component
- Rendering separated from UI logic
  - Visual update animates screens (also enter/leave)
  - Logic update is the actual IMGUI code
    - Uses locator id to identify components
  - Structurally mostly static screens

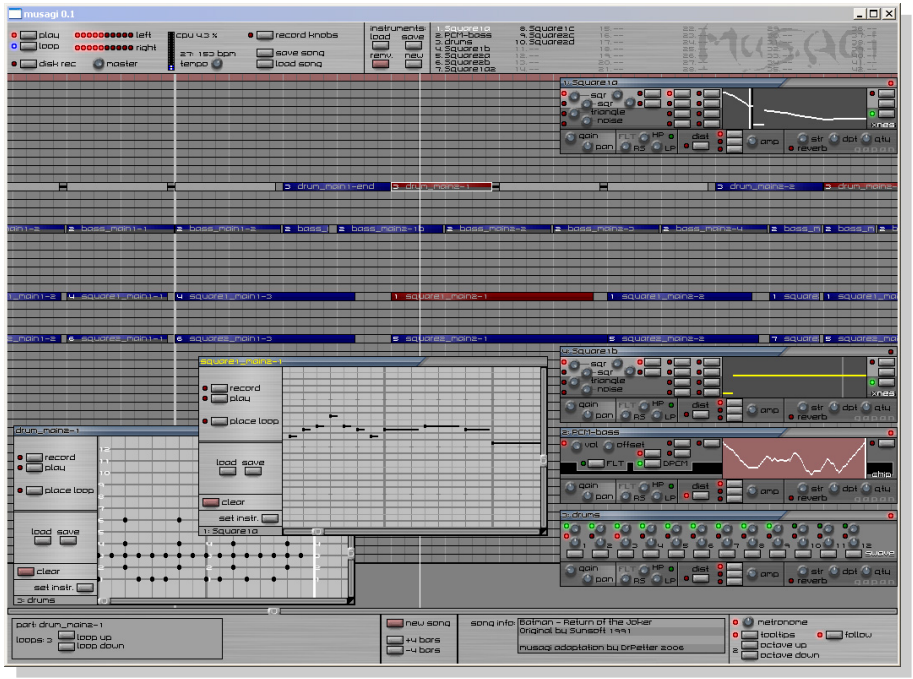
Case: IMGUI with PS3 - SSHD





Other Cases

- *Cinnamon Beats*
  - In Assembly'07 game development compo. :-)
- Zero Memory Widget library
  - whitepaper and first implementation from 2003
- *Boom! Boom! Driller*
  - Asm'06 game
- Musagi
  - music editor and synthesizer with fairly complex UI



More info

http://iki.fi/sol/imgui/
- Tutorial, different widgets, keyboard use etc.

http://www.mollyrocket.com/video/imgui.avi
- The original video lecture

https://mollyrocket.com/forums/viewforum.php?f=10
- original IMGUI forums

Game Developer magazine, September 2005, Volume 12, Number 8, Pages 34-36

http://www710.univ-lyon1.fr/~exco/ZMW/
- Zero Memory Widget library

http://www.cyd.liu.se/~tompe573/hp/
- Musagi

Backup Slide:

Decoupling Logic and Visual Look

- Button as an example
  - Define a ButtonStyle class
    - Defines isInside() and render() methods
- Inherit and create a custom one
  - E.g. ImageButtonStyle which takes in an image for each button state (on/off for hot and active)
  - Give pointer to the ButtonStyle object in the button(...) call

Backup Slide:

From Prototype Version to Final

- One idea to make it easier to move from quick prototype to the fine-tuned final version:
  - Create hard-coded UI (positions etc.) with each component having an textual id as well
  - Support loading of UI screen definition files
  - If the file has a component definition for a given textual id, it overrides the hard-coded values