Groking GUI

How I stopped worrying and ~~love~~ accept *Flash*

…

Nacho Lasheras

Game Programmer

@supernacho@mastodon.gamedev.place

1 July 2016

Description of the problem

UI is a source of headaches during development

It was something that we usually offload to a junior programmer, as most people don't enter into the industry to make the game menu and they prefer working on graphics or the AI code.

But that is not viable if your UI is big enough (and most PC or mobile games would qualify there)

UI in books

Let's see what can we find in the game programming best sellers:

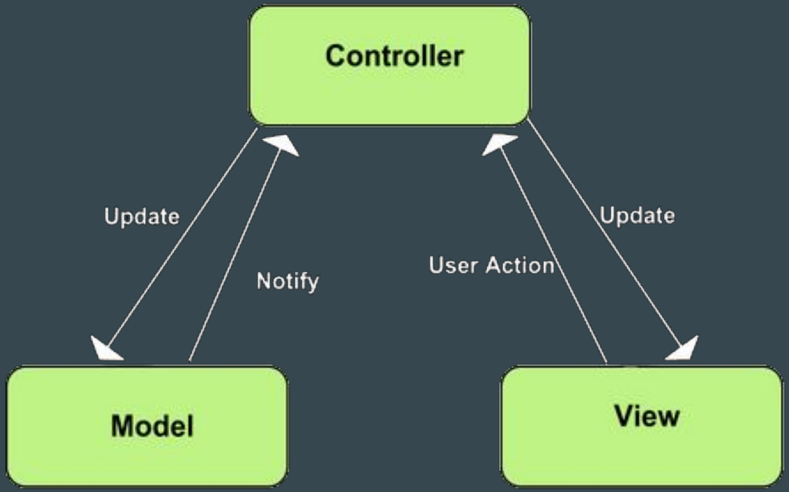
Game Engine Architecture (*the Jason Gregory book*) - nothing

3D Game Engine Design - brief mention to use of screen-space polygons to render the menus

Game Coding Complete - a full chapter explaining a basic but functional UI system using DXUT. Ends recommending to buy an off-the-shelf solution like *Scaleform* or *Iggy*

Peeking at software engineering

If somebody knows about making UI is people that work in CRUD applications. There I found a model which roughly worked for me, the MVC pattern.



The other boxes of MVC

But I didn't apply that pattern in very orthodox fashion, but it was a natural split of responsibilities that worked for our problem.

Model ≈ Game Logic → Gameplay programmer

View ≈ UI Layout → UI artist

Controller ≈ UI code → UI programmer

Model

Model is not really that interesting...

It's the core logic, rules and data of the game

Shouldn't know anything about the view or the controllers

You probably are working like that, as the decoupling between logic and render is a quite common pattern in game architecture

View

View

It defines the visual look of a UI

The root source of that is an artist that works f.ex in *Photoshop* and usually ends up with a *mockup* of how the UI should look.

Why are we treating UI artists different to the other artists in our company?

Typical asset pipelines

3D model/animation → exporter → data build → (binary) asset → graphic engine loads it and displays it

2D mockup ↠ texture assets + ui layout → ui code loads it and displays it

There is a lot of extra work to get the vision of the UI artist into the game!

Ideal UI pipeline

2D view → (data build) → asset → ui code loads it and displays it

If we are building a UI library for our engine, we want to set up a pipeline that enables artists to iterate the views without programming interaction

Anatomy of a view

What it it’s inside of a view?

- A sequence of *widgets* (usually linear, although widget composition would be very helpful)
    - type
    - size, position
    - the rest of the properties that will depend on the widget type (text for a *label*, the asset for a *picture*)
- (Optional) Animations. At least, the animation for fading in the view (we can play it backwards to fade it out)

How we build it?

- Text resources (custom format or XML, JSON)
  - Easy to implement and debug. Works great with source control. It's not the friendliest system for artists, but it's a first step.
- Custom editor (outputs text resources or binary asset)
  - Making an editor that solves positioning widgets is not too hard and will make artists life easier. The animation part is a little more challenging.
- Other tool format (*Flash*, *XAML*)
  - Loading assets from another tool is a great solution if that tool solves all the editing (which will do much better than your editor). You should choose the tool that your artists are using.

There is no silver bullet!

Widgets

You want to have as few different widget types as possibly.

- It makes your UI easier to understand
- Making a widget takes time to do it well, and the users are used to very polished toolkits like *Cocoa*.

Think and implement *microinteractions* by default.

Custom Views

Sometimes is hard to fit a screen to our set of widgets

- Puzzle level selection
- Maps
- Esoteric character evolution systems
- Mini-games

You can implement this screens as a full screen widget that handles everything, but keep in mind that you are re-implementing a widget toolkit just for that screen, so don't abuse this possibility.

Controller

Controller

Is the code that hooks the view with the data

We need one controller per screen

- Transform the logic data into the representation that we see on screen.
- Manages the input (handles button clicks, selection changes, etc.)

Who controls the Controllers?

Optionally, we may have a screen manager that handles the display and fade of each screen

- It makes easy to change and manage the flow
- Can store some state

Are you repeating to me?

There is a lot of shared behaviour in each screen

- Data binding (set labels to values from data)
- Configure/handle buttons
- Add contents to different kind of containers

A lot of this behaviour can be automated for many cases

- Keep the manual method and settle for an imperfect solution!

Closing

Multiple resolutions

One of the big responsibilities of a UI library in mobile is to resize the views to the proper screen size

There is multiple ways to handle the different sizes:

- Virtual units (*scale and center*)
- Anchors
- Springs

It's a complex problem that deserves its own talk. Try to think about it early in development!

Sharing styles

You may want to let you define widgets by data

- This data-widgets are *alias*es to a set of properties
- Useful because a UI designer works like this
- If your views are text resources, it will save a lot of *copy and paste*

Not just 2D quads...

Even if your UI is 2D based, at some point you will want to render 3D elements.

Think how are you going to place the 3D object in the widget rect

- Fixed camera and render to texture
- Use math to fit the BBox of the object into the widget area

Depending on your needs you may need to break the batching if you want to display the 3D object in a precise order.

Buy or build?

Generally speaking, you almost never should roll your own your UI library, unless...

- You understand how much it will cost
- You have a requirement that no commercial solution meets
- Company is on board with the idea (game team, artists, etc.)

Never roll your own UI solution because it will be cheaper

Thanks

@supernacho@mastodon.gamedev.place