

ANATOMÍA DE GUI

Nacho Lasheras

Gameplay Programmer

@supernacho@mastodon.gamedev.place

7 March 2015





Daniel Cook

@danctheduck

"UI will always take up way more time than you initially expect. Always." Eternal project management advice from @Wertle



Dan MacDonald

@samuraidan

@danctheduck @Wertle that's especially true in F2P games that often rely on the metagame to monetize, visualizing that requires a lot of UI

In the beginning the UI was simple



Most of the player time was spent on the *sec to sec*, so as developers we focus our effort there.

But that is not enough anymore



Why this mobile games need so much UI?

Sec to sec = intrinsically rewarding



EVE Online promises epic fleet battles...

Metagame = excite player with a extra 1%



EVE is, in reality, a game about waiting a month to put four launchers in your spaceship.

What can F2P learn from EVE?

- EVE is a service. *Retention* is key to CCP as it is to us in F2P.
- EVE has quite simple gameplay: you select a target and mash your weapons.
- EVE players create the “content” for other players.

What does it have to do with UI?

Successful F2P games are *hobbies*, they need a deep enough *metagame* to engage players enough to keep playing.

⇒ This systems are usually heavily UI reliant.

Who builds it?

A lot of people involved

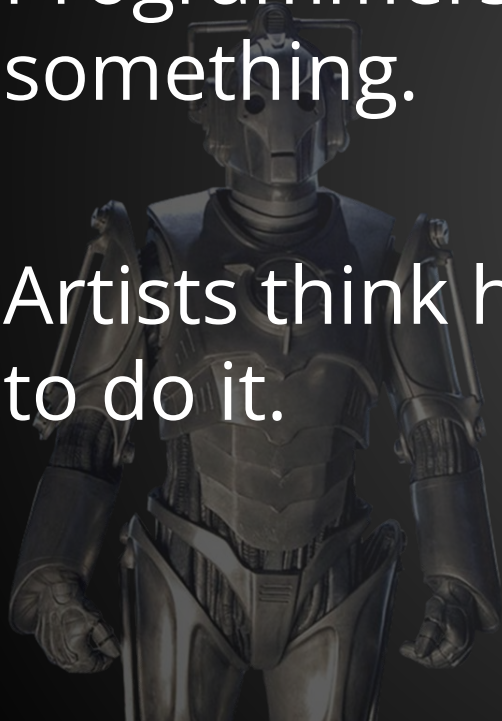
- Game designer
- 2D artist
- UI/UX designer
- Programmer

But who should do what?

Artists/Programmers

Programmers think about the best way to do something.

Artists think how can use the tools they have to do it.



Art is a hack, but a very pretty one

Artist content may not be elegant, but it will be prettier than a programmer made one.



Artists don't fix bugs

Offloading UI work to art team is a net win:

- ⬆ Better quality layouts
- ⬆ More time invested in them (more animation and polish)
- ⬆ Understanding of the platform limitations

**How to work with your
artist?**

What is a *layout*?

If you want artists doing the UI, they need a way of “building” the screens with the minimum amount of help.

Ideally the UI layout is just *data*, like any other *asset*.

Design Patterns

There is a lot of literature about MVC, MVVM and MVP.

Depending on your use case a different approach will be better for gluing the *model* and the *view*.

How the UI and *sec to sec* dance?

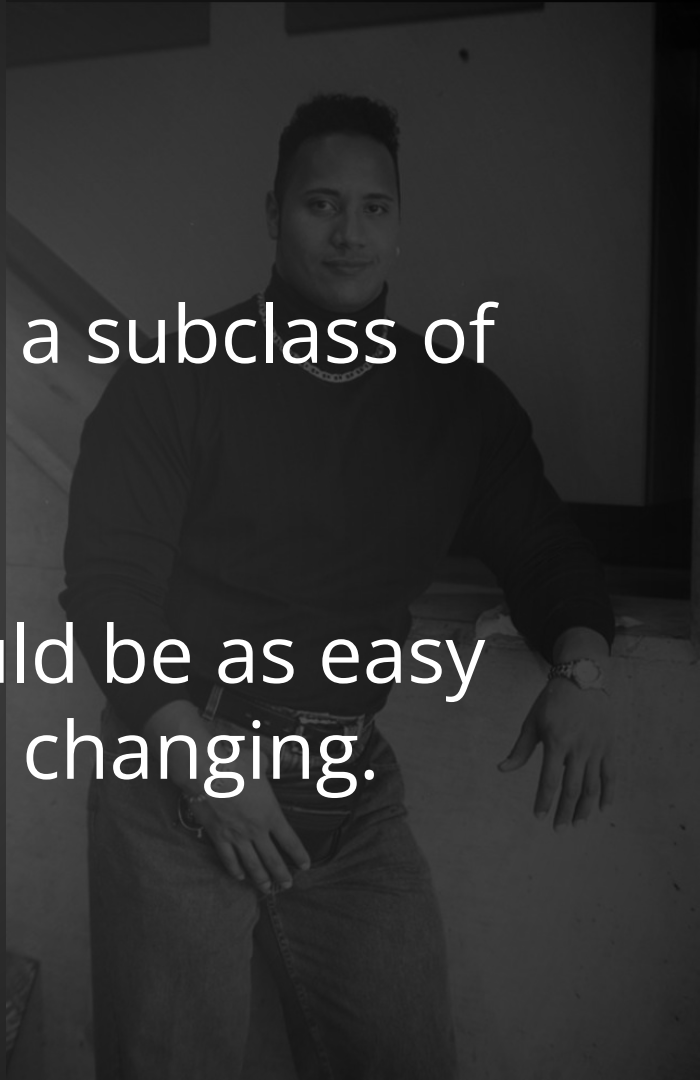
- The *sec to sec* is one of the states of the UI flow.
- The “main game loop” shows the screens



Style

You want your `ButtonRed` be a subclass of `MyGameButton`

Changing the game style should be as easy as possible. Because design is changing.



Atlases

If your engine supports batching, you want to maximize it on the UI.

Keep an eye on the draw calls of the UI.



But don't *atlas* too soon

Beware of packing stuff when your game style is not set.



Resolutions

That is a very pretty layout... it would be a shame if it didn't fit the screen

This topic could fill it's own talk

There are many different approaches to scale UI to different sizes: anchors, springs, grid based systems, ...

Making a UI dynamic is quite tricky, especially if you have to support very different sizes.

Don't try to solve it for everything

Set a “base” resolution to develop (most common target device is better)

You want to use your dynamic system as minimum as possible, as it needs a lot of tweaking to make widgets resize to any size.

Resizing example

To resize from iPhone 4 (640x960) to iPhone 6 (750x1334) you need to resize 110 pixels in width and 374 in height.

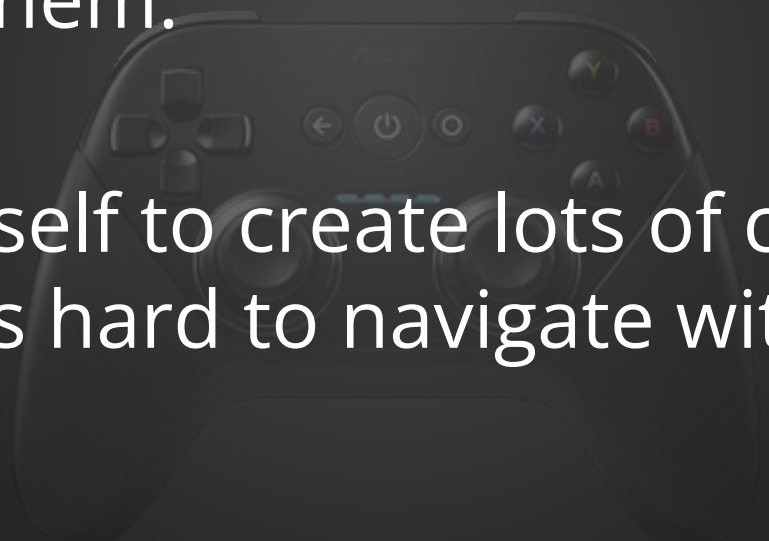
1. Scale the screen to 750x1140
2. Use anchors to resize the layout 194 pixels in height.

Result of this approach usually looks better. NGUI and New UI can be made to work that way out of the box.

Think about gamepads

If you want to support gamepads, think a little about them.

Touch lets itself to create lots of clickable stuff which is hard to navigate with a gamepad.



Sometimes, size matters

In theory, a iPhone and iPad have very similar resolutions.

In practice, you need a different UI because the screen size is much bigger.

***“A image is worth a
thousand words”***

But text has their own set of problems in
the world of UI

Localization

- Word lengths (English is very short)
- Hardcoded string archeology
- Culture (decimal separators, currency position)



Text Rendering

That demo in *CodeProject* is happy enough with ASCII, and it will be probably ok for EFIGS+Russian, but won't work for Asian languages.

Latin-9: 256

Chinese (Simplified): ~7,000

Chinese (Traditional): ~13,000

Korean: ~11,000 syllables

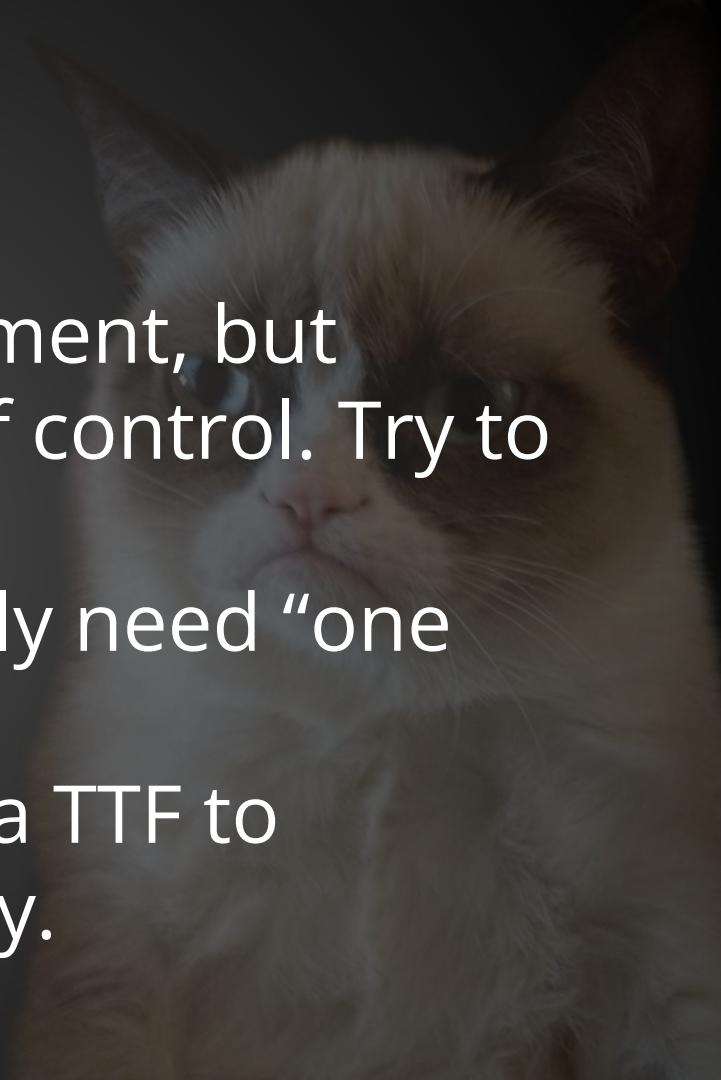


한글

And Arab is even more complex.

Fonts: Bitmap vs TTF

- Bitmaps are easy to implement, but package size can get out of control. Try to minimize number of faces.
- TTF are slower, but you only need “one font” and they look nicer.
- A tradeoff would be using a TTF to generate bitmaps on the fly.



If you really need to use Bitmap

Try to keep text scaling to the minimum.
Never scale up (except on animations).

Generate the fonts so they are displayed as close to pixel-perfect.

If possible, evaluate using **SDF**



Some final tips

Lessons learned that I couldn't fit
elsewhere.

Scripting

Very basic scripting will help you with a lot of common cases.

Full-featured scripting will solve almost all of them.



Some Unity tips



- DONT CODE YOUR UI FRAMEWORK
 - Use New UI or buy NGUI (Noesis is also fine)
- No nested prefabs :(
- Each screen should be “runnable”
- PlayMaker for making UI flow
 - Mecanim looks like it could be used for that, but Animation state machines are not FSM

Questions?

Nacho Lasheras

@supernacho@mastodon.gamedev.place

