



Contents

- [1 First case: one main screen plus one configuration screen](#)
- [2 1. Main screen: as clean as possible and with the minimum amount of controls](#)
- [3 2. Configuration screen: with all the controls](#)
- [4 Overlaid help](#)
- [5 Wire structure](#)
- [6 Second case: Many different screen with equal importance](#)
- [7 Language and librarie used for the examples](#)
- [8 External links](#)

First case: one main screen plus one configuration screen

Some kind of applications may need a 'complex' configuration and just a small amount of interaction from the user for the normal use. One approach could be to split the application in 2 different screens:

1. Main screen: as clean as possible and with the minimum amount of controls



the main screen's task must be clear and focused on the goal (show the time and be pretty) while the configuration screen can be as complex as the application need.

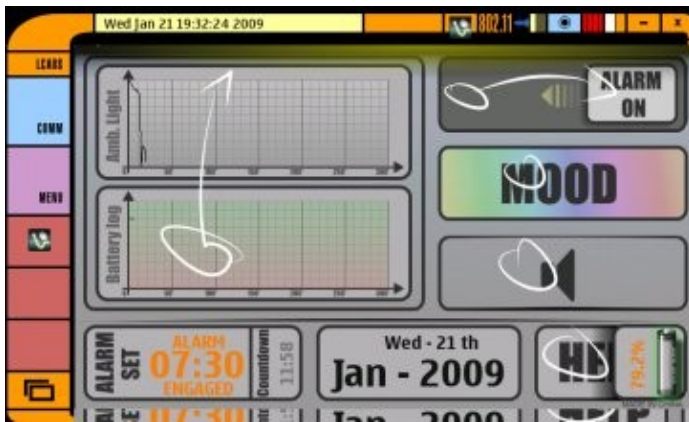
2. Configuration screen: with all the controls



Dividing in 2 the application the 'interactions style' could be a bit more free from the 'consistency rule' (let the user use the fingers for the main screen and give some stylus licences on the config screen).

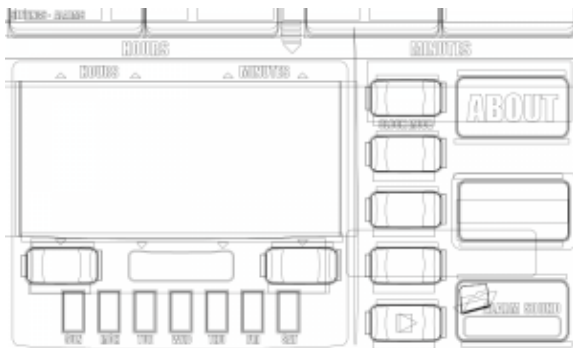
Overlaid help

If the application need more gestures is possible to give hints to the user with an overlay graphics:



Wire structure

Since python facilitates the port between different platforms is recommended to create the graphic elements using a vector base program [1] (Adobe Illustrator, Inkscape etc.)



Second case: Many different screen with equal importance

Different case when the application need more interaction from the user like, for example, in an audio player. It is important to always leave a clue of where the application is and to keep always available the main controls



Language and librarie used for the examples

Python is the ideal choice for rapid prototyping and have 'binds' with almost every graphics library on any possible device. Since is an interpreted language^[2] (more specific compiled at runtime) it works without any modify to the source on every device. Since the interface is the application part that really 'talk' with the user is important to have a library for the graphics that can react and be responsive as much as possible like a videogame. That's the reason behind the pygame choice but any library can do the trick (Qt, GTK, CAIRO, etc)

[Official Python page <http://www.python.org>]

S60

[Python <http://opensource.nokia.com/projects/pythonfors60/>]

Maemo

[Python <http://pymaemo.garage.maemo.org/>]

[Official Pygame site <http://www.pygame.org>]

S60

[Pygame <http://code.google.com/p/pygame-symbian-s60/>]

Maemo

Maemo Pygame page

[Pygame

Mobile_Design_Pattern: _multi_screens_application_approach

http://repository.maemo.org/extras/pool/diablo/free/p/pygame/python2.5-pygame_1.7.1-1osso2_armel.deb
(direct link to the installation .deb)

External links

[Project page with sources of the example application: <https://garage.maemo.org/projects/flipclock/>]

[Clip (version 0.5) <http://www.youtube.com/watch?v=WRMxSStlGxg>]