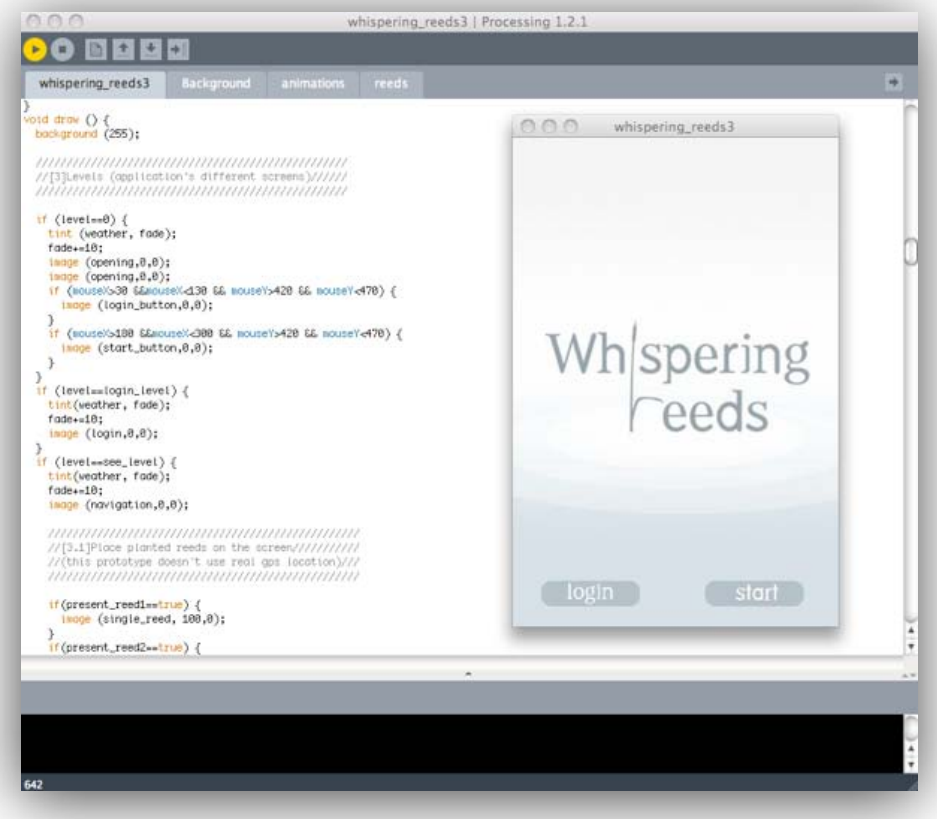
THE CODE:
interesting parts



Coding in Processing is like a **competition** with ourselves, and against Processing itself. Once you started you can't stop. You'll hate it and you'll love it: «*Odi et amo*»... but in the end you'll become great frieds.

Here we want to summarize and show you the parts of the code of our application that we think are the **most interesting** and **useful** for anyone's future projects. You are totally free to copy and use it for your needs.

Mind the fact that there are some modifications we needed to do in order to make the application work on the a smartphone using the **iprocessing** library. We will underline those changes.

#1 levels

Whispering Reeds

The application contains 6 main screens. We made a variable called "level" and assigned each level a number of this variable:

```
int level=0;

int login_level=1;
int see_level=2;
int augmented_level=3;
int map_level=4;
int plant_level=5;
int born_reed=6;
```

Then we called each level in the "void draw" function, like this:

```
if (level==login_level) {
//everything that happens in this level//
 }
```

And assign every button a level to call if pressed, this way:

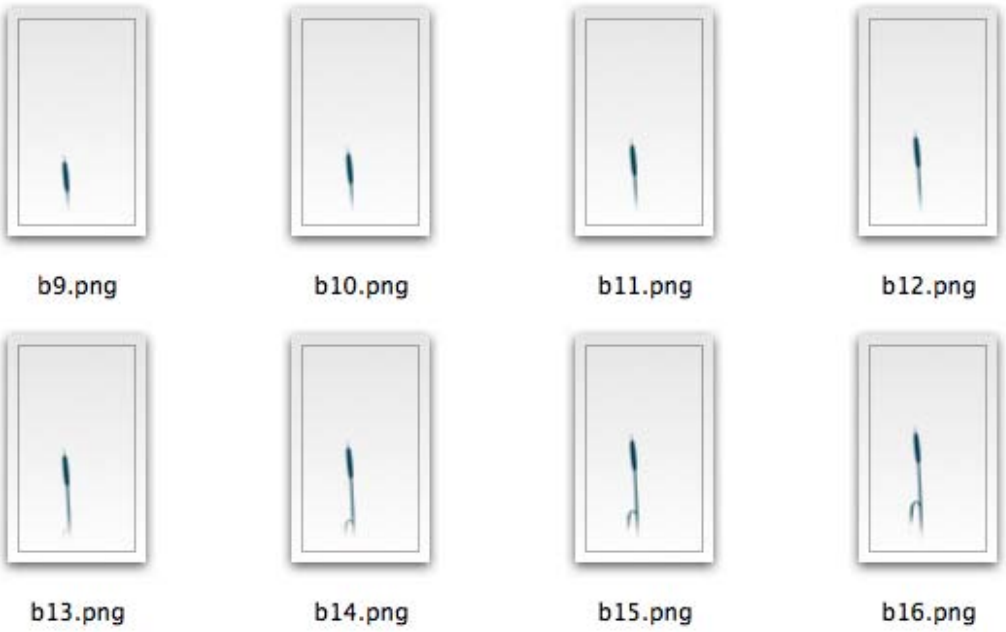
```
void mousePressed() {
 if (map_button==true && mouseX<100 && mouseY>420) {
  level=map_level;
 }
 if (see_button==true && mouseX<220 && mouseY>420 && mouseX>100) {
  level=see_level;
 }
 if (plant_button==true && mouseX>220 && mouseY>420) {
  level=plant_level;
 }
}
```

Note that in order to export our application on a device with a touchscreen we needed tho change "mousePressed" in "touch1started".

#2
animations

Whispering Reeds

We tried to keep the animations as light and simple as possible. they're made of a sequence of png images, loaded one after the other. The files are named with a letter and a number, for instance:



When the animation needs to start, a function is called loading all the frames in a quick succession:

```
if (born_reed_frame<=30) {
    birthing_reed= loadImage ((“b”+ born_reed_frame)+”.png”);
    image (birthing_reed, 85, 120);
    if(frameCount%2==0) {
      born_reed_frame+=1;
    }
```

As you can see, the animation frames go from 1 to 30, so when “born_reed_frame” equals 30, the animation stops.
In order to speed up or slow down the animation we used the “frameCount” variable. This is particulary important when exporting the application on the real device, because the processors' different clockrate will change the animation a lot.

It's better to load all the images at the beginning of the application, under “void setup”: this too will make the program faster on the device.

#3
buttons

Whispering Reeds

Our application has got only three buttons, called “map”, “see” and “plant”. But the avability of the different buttons depends on the level we are in, so for instance we will have both “map” and “plant” buttons in the navigation screen, but only the “see” button in map mode.

So, instead of making a different image whit different buttons for every screen, we made a single png image for each button and then load it only if needed, assigning a boolean variable to the buttons and making it true or false when a new screen appears.

```
if (see_button==true) {
    image (see_symbol,0,10);
}


if (level==map_level) {
    see_button=true;
  }
```

#4
fading

The mood of our application is very light and dreamlike, so we wanted to make the transitions between a screen and the next a little more smooth. In order to create a sort of fading we first made a variable called "fade" and then, at every change of screen, we made the images appear gradually with the "tint" parameter:

```
if (level==login_level) {
   tint(weather, fade);
   fade+=10;
   image (login,0,0);
 }
```

We assigned different switchable colors to the variable "weather": this allows us to change the tint of the whole application with one single command.

Unfortunately this fading system doesn't work on the device because modifying every pixel of the images takes too much time and power for the processor. So instead of changing the images' tint, we can draw lots of rectangles with decreasing opacity. This works better:

```
if (level==login_level) {
   image (login,0,0);
   if(fade > 0) {
     fill(255, fade);
     rect(0,0, 320, 480);
     fade-=10;
   }
```

Of course before every new level we need to reset the "fade" variable:

```
if (level==0) {
   if (mouseX>30 &&mouseX<130 && mouseY>420 && mouseY<470) {
      level=login_level;
       fade=255;
    }
}
```

We hope this snippets will help you. Have a nice coding!