Learning objectives

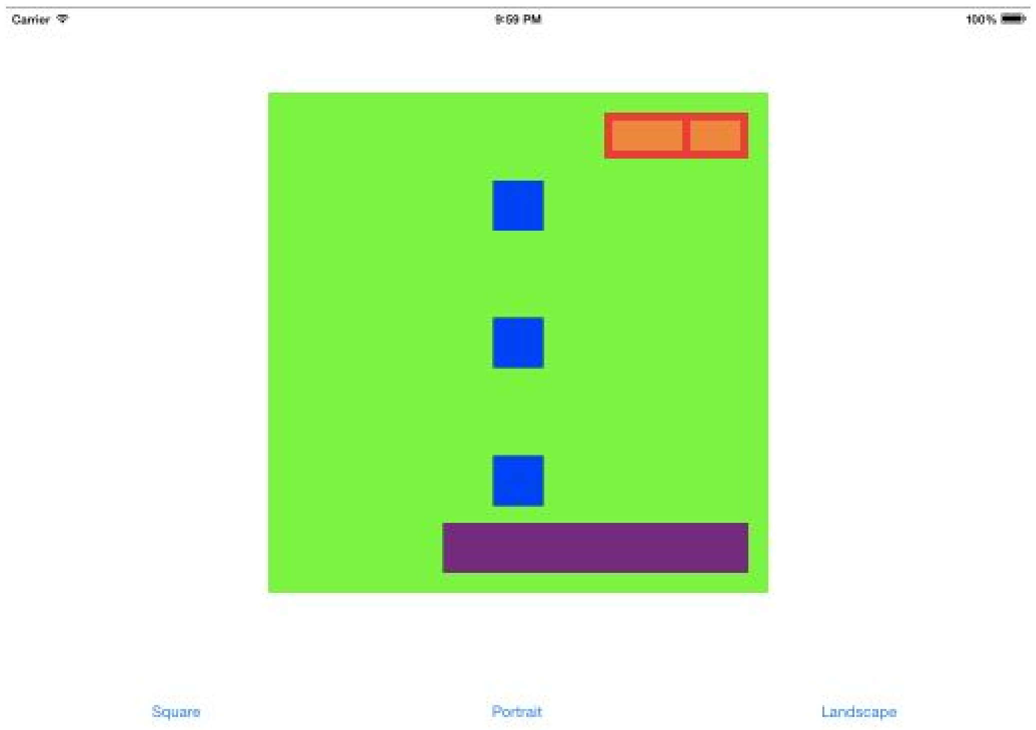
- Understand the Auto Layout system
- Understand view constraints and relationships
- Understand how to define view hierarchies implied in a graphic of a view containing many subviews

Instructions

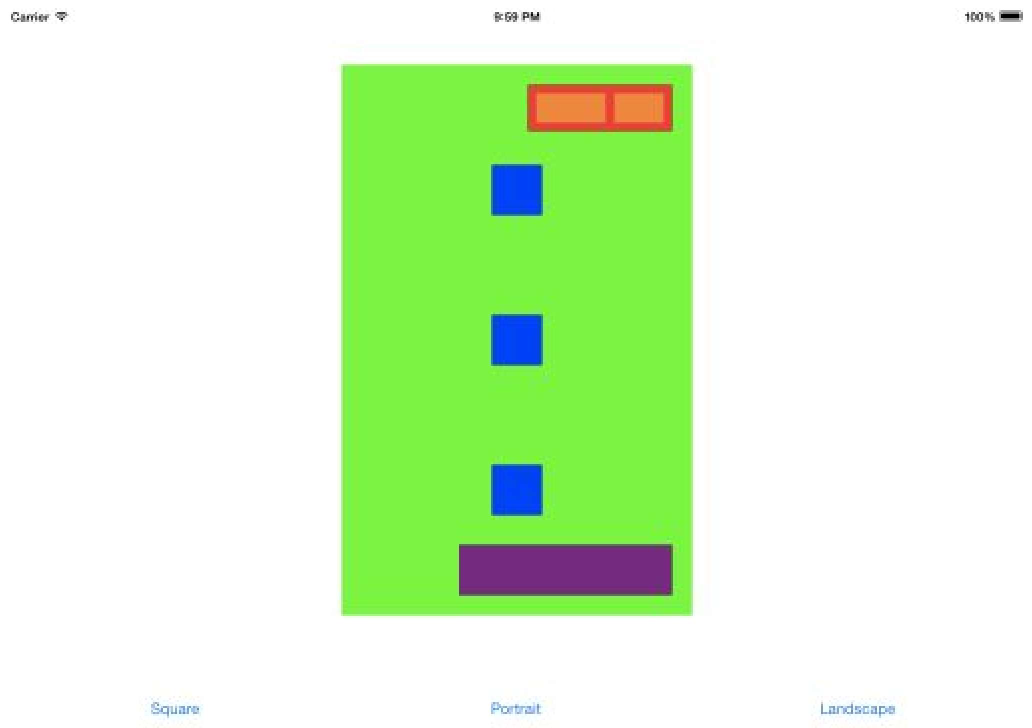
You'll have to write some code in the `ViewController` class that sets up some buttons and a big green view for you, as well as a method that changes the size of the box from square, to portrait, to landscape so that we can test our AutoLayout constraints.

You will need to implement a view hierarchy and Auto Layout constraints that will produce the following (sizes and constraints detailed below):

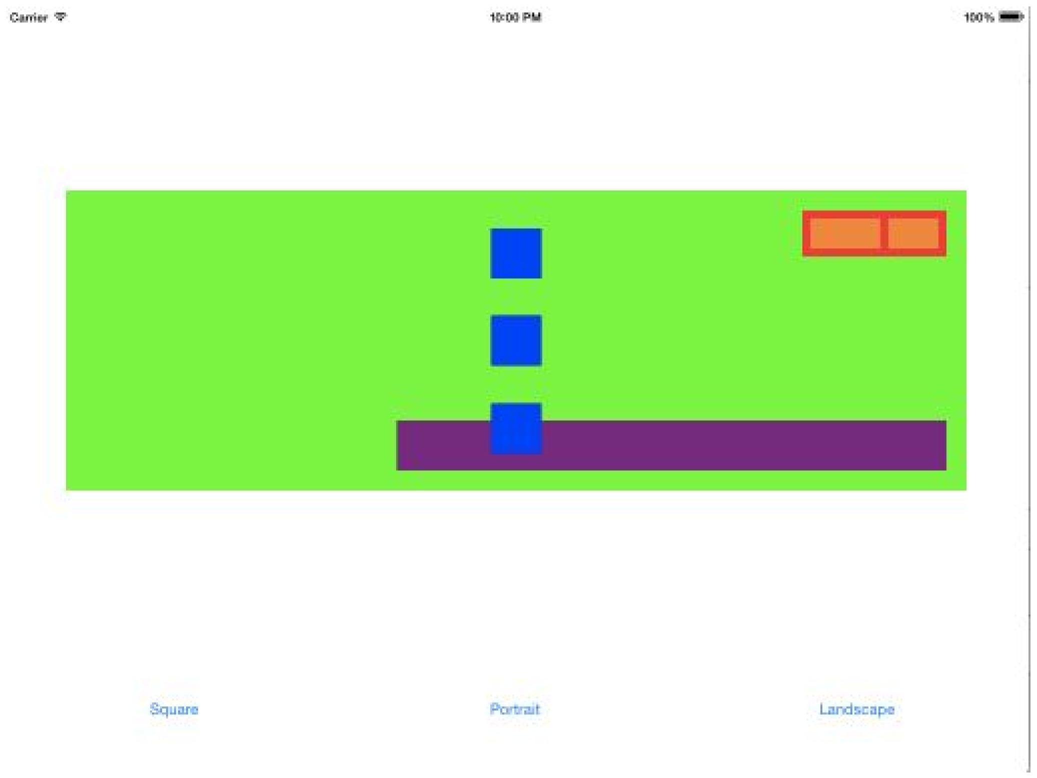
Square



Portrait



Landscape



Notes

- The purple box is 50 points tall and positioned 20 points away from the bottom and right edges of the green box. You can see the purple box changes width, but as one point of reference: when the green box is 500 points wide, the purple box is 305 points wide.
- The orange boxes are 30 points tall, with the left one being 70 points wide and the right one being 50 points wide.
- The red box is positioned with its edges 20 points away from the top and right edges of the green box.
- The blue boxes are 50 points tall by 50 points wide.

- Everything else is the default spacing between views, either as sibling or parent views.

Lets do the purple box together.

- Firstly, the goal we are trying to achieve is to have a purple box that changes width depending on our green box (framingView). To achieve this result, we add constraints on our purple box that change its width in proportion to green box's width. This is the beauty of AutoLayout.
- To start, we create a new UIView and set its frame to CGRectZero. We could have made the box with fixed values ex) CGRectMake(0, 0, 100, 50). This creates a box at (0,0) x,y coordinates and with 100 width and 50 height. However, we want to create something with dynamic properties. Notice that for the purple box x,y and width all change depending on if you are in square, potrait, or landscape mode. To make our size dynamic we set the frame to CGRectZero, which is empty box (equivalent to CGRectMake(0, 0, 0, 0)), and we assign layout constraits to change our x,y, and width depending on framingView's properties (which we framingView's size in resizeFramingView:)
- Then we set translatesAutoresizingMaskIntoConstraints to NO
- Set the view's background colour to purple
- And add the purple box to our view.

Next, we will add 4 constraints (using NSLayoutConstraint constraintWithItem).

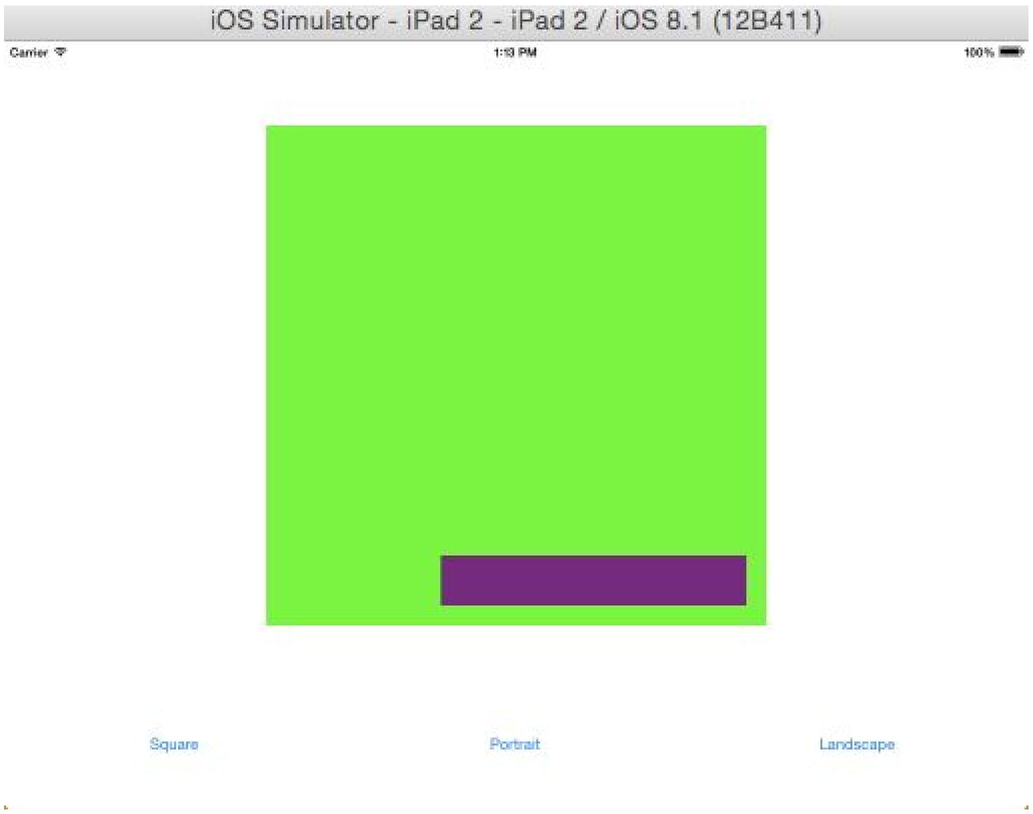
https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/AutolayoutPG/ProgrammaticallyCreatingConstraints.html

http://rshankar.com/how-to-programatically-add-autolayout-constraints/

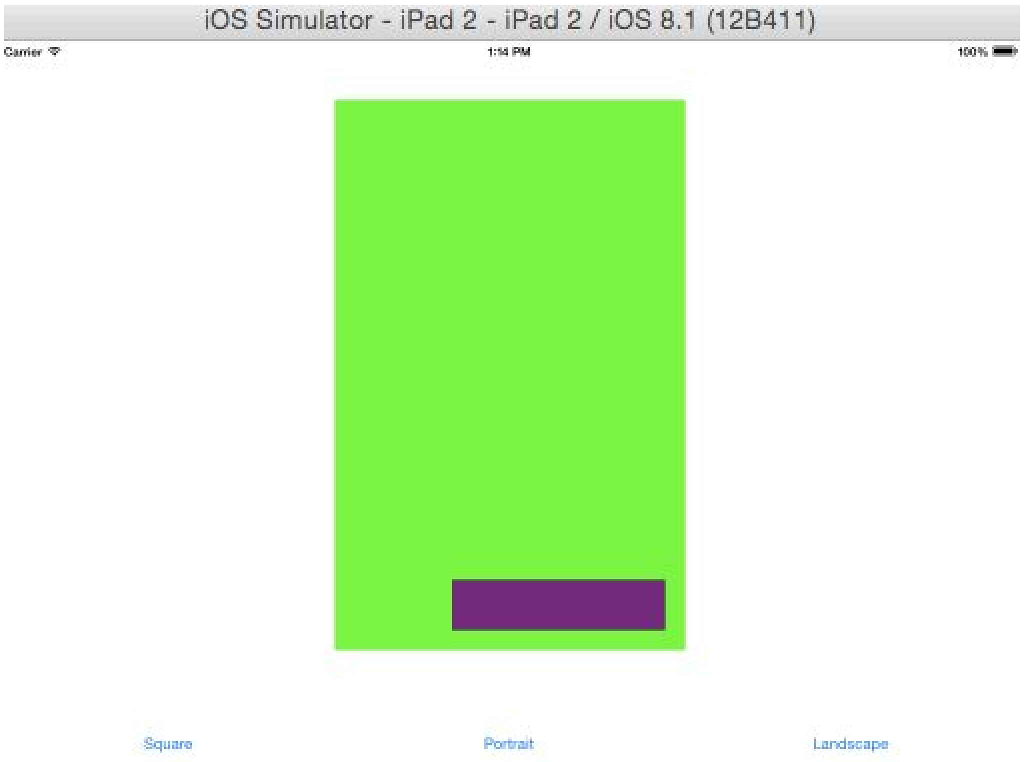
- One constraint will set the purple box's right margin equal to a 20 points offset to framingView's right margin
- One constraint will set the purple box's bottom margin equal to a 20 points offset to framingView's bottom margin
- One constraint will set the width of purple box to (305.0/500.0) of the framingView's width. Use the multipler property to set this.
- One constraint will set the height of purple box to a fixed size of 50 points. Since this constraint is not in relation to another our View, set toItem to nil and toItem's attribute to NSLayoutAttributeNotAnAttribute.

You should get the below results:

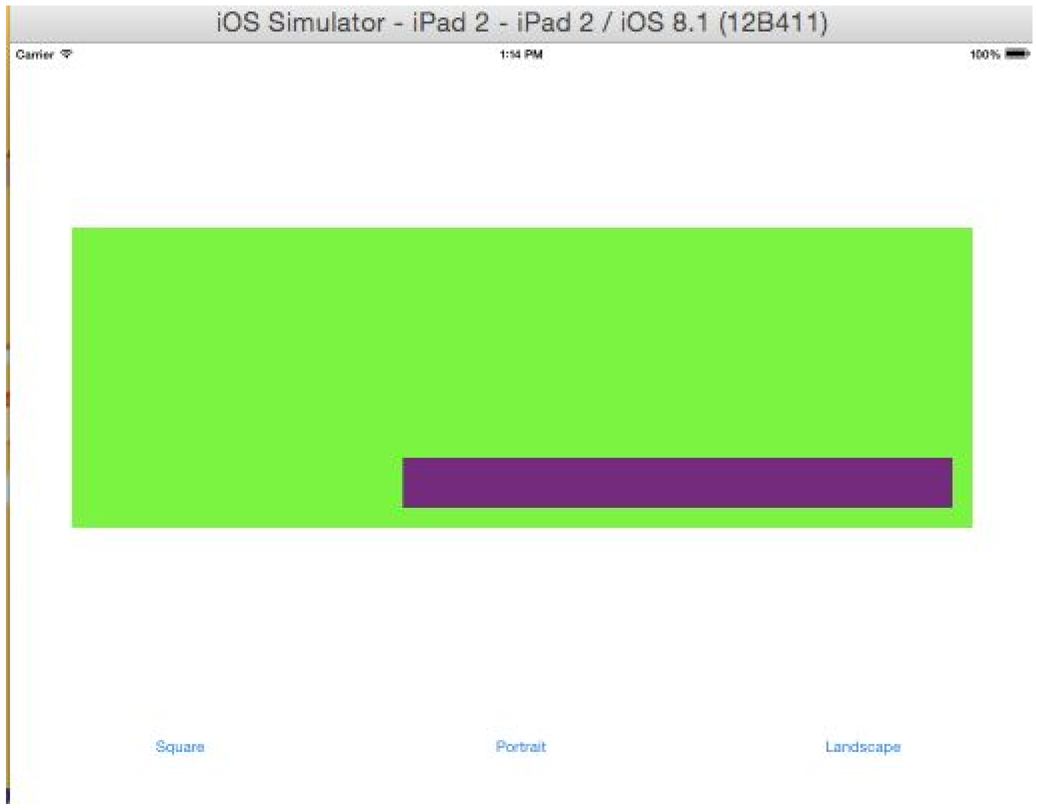
Square- Purple Box



Portrait- Purple Box



Landscape- Purple Box



References

- [Auto Layout Guide]()
- [10 Things You Need To Know About Cocoa Auto Layout]()

- Visual Format Language

Stretch Goals

Toggle-able Yellow Footer

Add a 150 point tall yellow view to the green view, and make it be always as wide as the green view. Add a constraint so that the bottom edge of the yellow view always aligns with the bottom edge of the green view. Add a button that shows or hides the yellow view (by setting the ‘hidden’ property on the yellow view). When the yellow view is hidden, the the other views should align as shown above. When the yellow view is shown, the views that have constraints that involve the bottom edge of the green view should refer to the top edge of the yellow view instead. This will mean that the three blue views will not fit all above the yellow box in every size; if they do not, they should be evenly spaced between the top and bottom edges of the green box instead. Refer to the idea of priority in layout constraints in order to figure this out.