贪吃蛇屏幕适配方案

背景：

贪吃蛇大作战App是需要同时适配iPhone与iPad的，面对目前众多尺寸的设备，如果每个设备都要单独布局，不仅会造成设计工作量的提升，对开发也会造成极大的压力。

分析：

目前业界比较常用的布局方式基本上都是 iPhone非全面屏一套方案，全面屏一套方案，然后对于iPad又是一套方案，另外由于很多设备虽然尺寸不同，但是屏幕宽高比却是相同的，再配合相应的缩放即可完成适配。

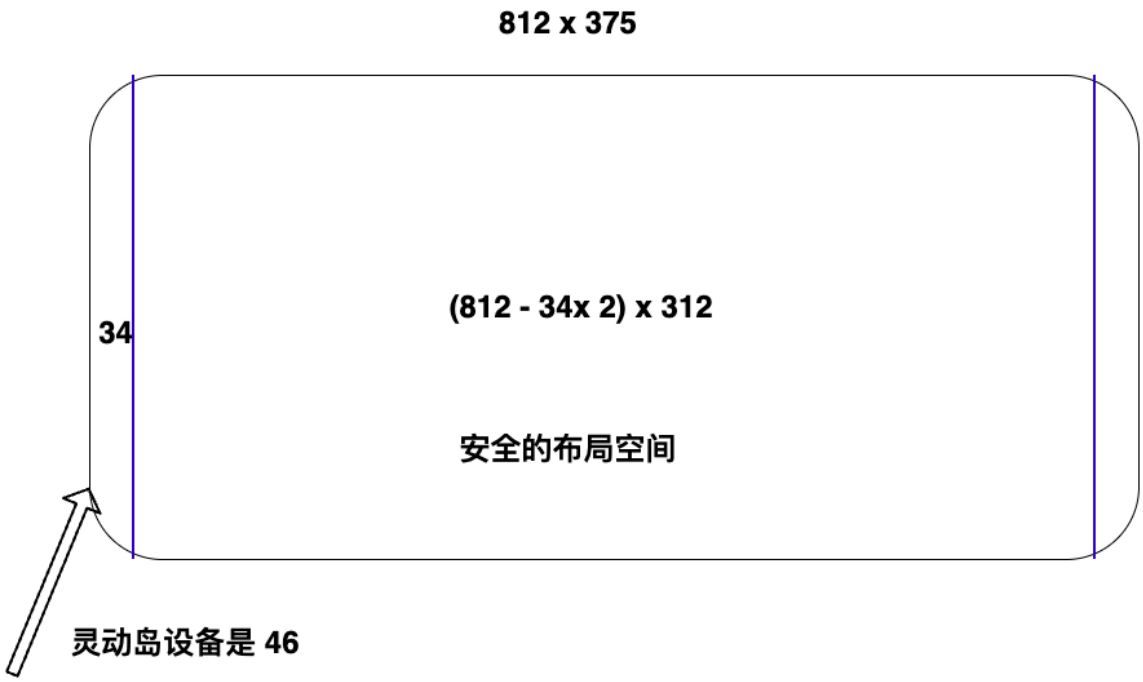
这种方式的优点很明显，就是思路相对比较简单，也能够满足大多数应用的需求，尤其对于竖屏设计还原度上也OK，所以大多数应用采用的就是这种方式。

但其实对于游戏来说，这种方式也有他明显的弊端，游戏页面空间的利用比较珍贵，在横屏时，如果采用这个方式在一些大屏手机例如ProMax上左右留白会比较大，导致很大的空间浪费，而实际上我们其实只需要把刘海那一部分作为安全区域即可，其他部分都是可以发挥的空间；另外，这种方式要求布局时需要考虑非全面屏iPhone，全面屏iPhone，iPad，虽然相较于单独每个设备适配有了很大的开发效率提升，但其实也还是需要做3种布局适配。

基于以上的分析，我们采用了一种更为激进的布局方式，采用一种方式布局所有设备，并最大限度的利用屏幕可利用的空间（扣除刘海后的安全空间）

实现：

设计提供的设计稿如下:



设计提供的安全布局空间【744 x 375】

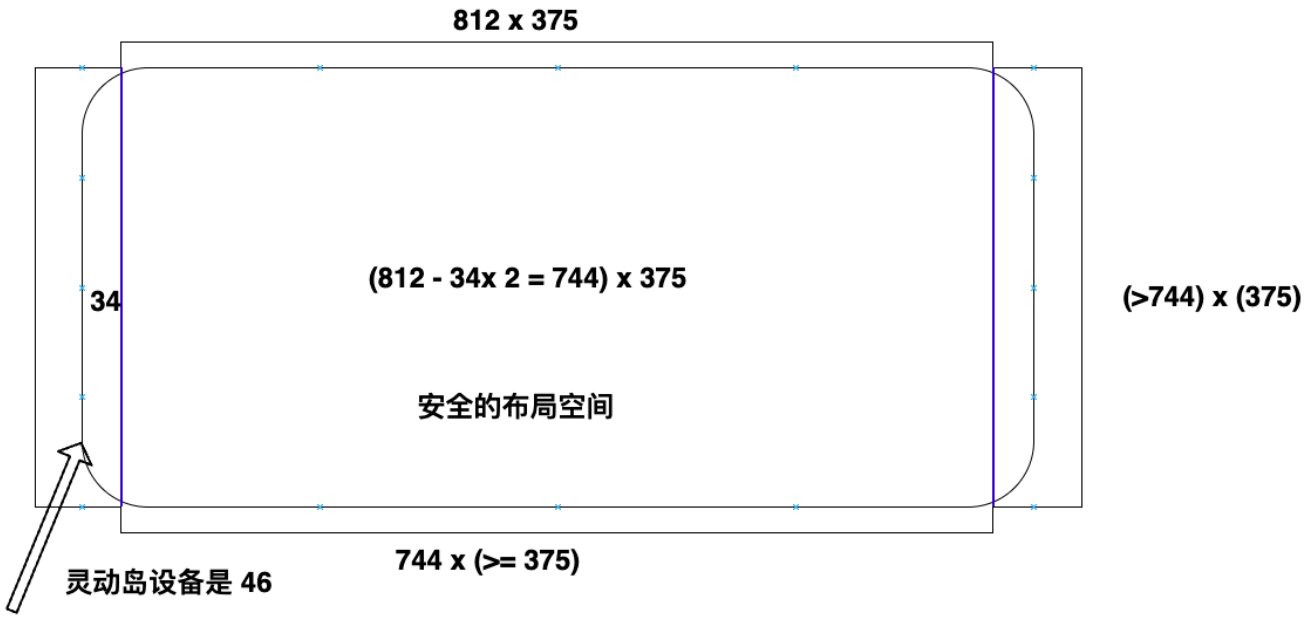
以此为基准，我将所有机型的安全布局尺寸与 【744 x 375】做对比，至少保证宽和高中的一项与此尺寸相同，并且保持所得到的布局尺寸大于或者等于 【744 x 375】，返回一个与屏幕宽高比相同的布局尺寸，代码如下：

```
    @objc static var layoutSize: CGSize {

        let screenWidth = max(UIScreen.main.bounds.width, UIScreen.main.bounds.height)
        let screenHeight = min(UIScreen.main.bounds.width, UIScreen.main.bounds.height)
        let ratio = screenWidth / screenHeight

        let designSize = CGSize(width: 744, height: 375) // 设计尺寸 (812 - 34*2) 744 375
        var layoutSize = CGSize.zero
        // 需要确保计算后的尺寸至少有一个和设计尺寸相同，并且所有值都大于或者等于设计尺寸
        layoutSize.height = designSize.height
        layoutSize.width = layoutSize.height * ratio
        if layoutSize.width < designSize.width {
            layoutSize.width = designSize.width
            layoutSize.height = layoutSize.width / ratio
        }
        // 尺寸宽高比==屏幕宽高比
        return layoutSize
    }
```

得到的两套尺寸(包括【812 x 375】本身)如图



此时在依据该尺寸与屏幕宽高做对比，计算出缩放比例，后面会根据该尺寸做缩放适配，

```
// 屏幕与布局尺寸的缩放比， 使用此进行transform缩放
    @objc static var layoutScale: CGFloat {
        let screenWidth = max(UIScreen.main.bounds.width, UIScreen.main.bounds.height)
        return screenWidth / layoutSize.width
    }
```

到这一步，对于非全面屏，例如iPhoneX以前的机型，iPad，其实已经可以依据一套布局尺寸进行正常的UI布局，最后对根视图依据缩放比例调用缩放就可以了；

重点是对于非全面屏的处理，因为我们并不是在安全区域直接留两个黑块，只是不显示一些常规的设计元素，对于返回按钮，对于页面背景都是要显示的，所以安全边距还是需要的，为了最大限度的利用屏幕，我们没有利用系统提供的 safeAreaInsets 返回的安全边距值

```
@property (nonatomic,readonly) UIEdgeInsets safeAreaInsets API_AVAILABLE(ios(11.0),tvos(11.0));
- (void)safeAreaInsetsDidChange API_AVAILABLE(ios(11.0),tvos(11.0));

// 横竖屏上是动态返回的值，并不一样，这里不做深入探讨

// 非全面屏返回的是 【top: 0 left:0, bottom: 0, right: 0】
// iPhoneX 等刘海屏返回的是 【top: 0 left:44, bottom: 21, right: 44】
// iPhone14 iPhone14Plus 刘海屏返回的是 【top: 0 left:47, bottom: 21, right: 47】
// iPhone14Pro iPhone14ProMax 灵动岛 【top: 0 left:59, bottom: 21, right: 59】
```

显然，系统返回的安全边距太大了，不利于我们合理利用屏幕，因此基于机型我们自己按照设备的物理遮挡，手动返回了一套安全尺寸，

```
// 屏幕上的物理安全区域
    @objc static var physicalSafeInsets: UIEdgeInsets {
        switch Device.current {
        case .iPhone14ProMax, // 430 x 932
                .iPhone14Pro, // 393 x 852
                .simulator(.iPhone14Pro),
                .simulator(.iPhone14ProMax):
            return UIEdgeInsets(top: 0, left: 46, bottom: 21, right: 46)

        case .iPhone14Plus,  // 428 x 926
                .iPhone13ProMax,
                .iPhone12ProMax,
                .simulator(.iPhone14Plus),
                .simulator(.iPhone13ProMax),
                .simulator(.iPhone12ProMax),
                .iPhone14,  // 390 x 844
                .iPhone13Pro,
                .iPhone13,
                .iPhone12Pro,
                .iPhone12,
                .simulator(.iPhone14),
                .simulator(.iPhone13Pro),
                .simulator(.iPhone13),
                .simulator(.iPhone12Pro),
                .simulator(.iPhone12),
                .iPhone13Mini,  // 360 x 780
                .iPhone12Mini,
                .simulator(.iPhone13Mini),
                .simulator(.iPhone12Mini),
                .iPhone11Pro, // 375 x 812
                .iPhoneX,
                .iPhoneXS,
                .simulator(.iPhone11Pro),
                .simulator(.iPhoneX),
                .simulator(.iPhoneXS),
                .iPhone11ProMax, // 414 x 896
                .iPhone11,
                .iPhoneXR,
                .iPhoneXSMax,
                .simulator(.iPhone11ProMax),
                .simulator(.iPhone11),
```

```
                .simulator(.iPhoneXR),
                .simulator(.iPhoneXSMax):
            return UIEdgeInsets(top: 0, left: 34, bottom: 21, right: 34)
        // 后续有变动，可以在此添加，即可完成新机型的适配

        default: // 非全面屏iPhone，iPad 屏幕都可以展示
            return .zero
        }
    }
```

到这里我们只用对安全边距进行反缩放，得到一个布局尺寸就可以直接使用布局的安全边距进行正常布局了

```
// 布局使用的安全边距
    @objc static var layoutSafeInset: UIEdgeInsets {
        let safeInsets = physicalSafeInsets
        let scale = layoutScale
        return UIEdgeInsets(top: 0, left: safeInsets.left / scale, bottom: safeInsets.bottom / scale, right: safeInsets.right / scale)
    }
```

具体调用

```
    self.vcView = [SNKHomeVCView new];
    [self.view addSubview:self.vcView];

    // 根视图布局  iPhoneXDeviceResizeSuitableSize 值的是 layoutSize
    [self.vcView mas_makeConstraints:^(MASConstraintMaker *make) {
        make.center.equalTo(self.view);
        make.size.mas_equalTo(iPhoneXDeviceResizeSuitableSize()); // 给定布局尺寸
    }];
    // 缩放比例为 layoutScale
    [self.vcView autoResizingBasedIphoneX]; // 对根视图进行缩放

    // 内部布局示例 SNKSafeAreaInsets 指的是 layoutSafeInset
    [self.eatClubBtn mas_makeConstraints:^(MASConstraintMaker *make) {
        make.size.mas_equalTo(CGSizeMake(49, 45));
        make.right.equalTo(self).offset(-17 - SNKSafeAreaInsets.right);
        make.bottom.equalTo(self);
    }];
```

这样就完成所有页面一套代码进行布局，并且尽可能的利用了屏幕空间，当出现了新的机型时，我们只需要修改调整 physicalSafeInsets 的返回值就可以完成适配。

> 注意点：
>
> 这套布局需要我们在写布局代码的时候稍稍注意下，思考下页面元素是适合基于 上 - 左 - 底 - 右 - 居中 布局，然后基于此写布局代码，这样才能尽可能的还原设计。