



191,00

Рейтинг

Dodo Pizza Engineering

О том как IT доставляет пиццу

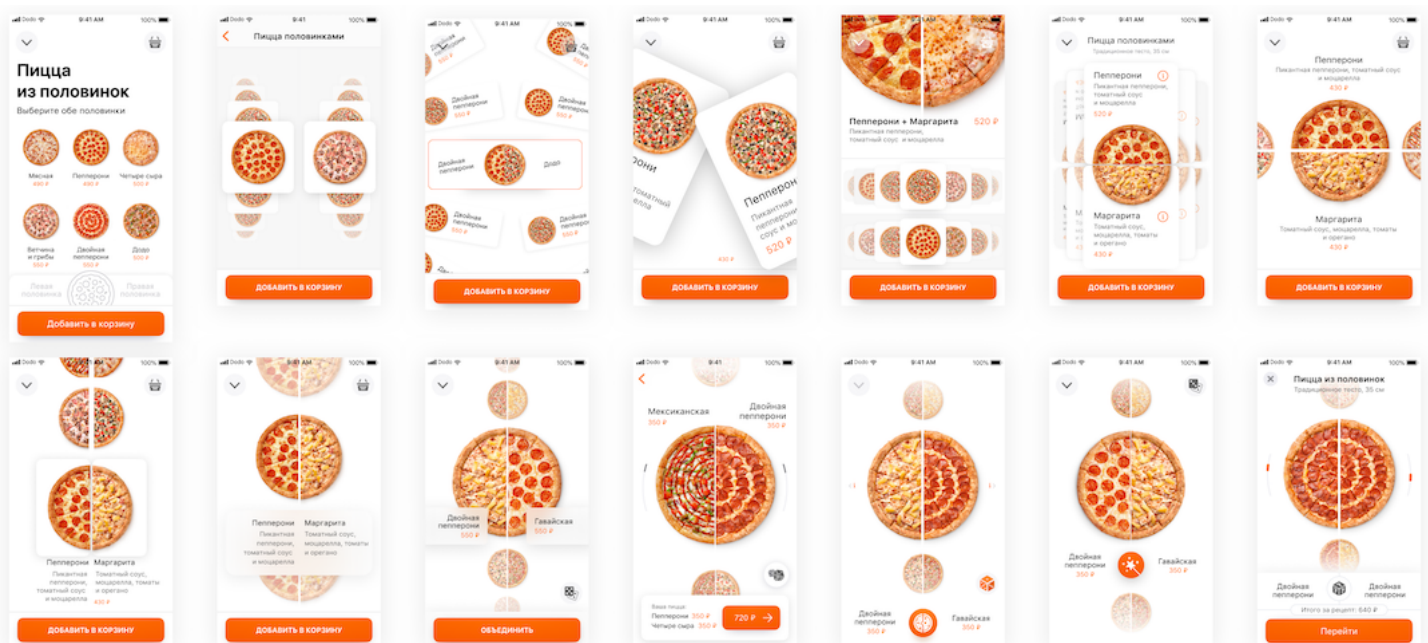


akaDuality 23 мая 2019 в 18:39

UICollectionViewLayout для пиццы из разных половинок

Блог компании Dodo Pizza Engineering, Разработка под iOS

Чтобы сделать пиццу из половинок мы использовали два UICollectionViewLayout. Рассказываю о том, как мы написали такой лейаут для iOS с чем столкнулись и от чего отказались.



Прототип

Когда к нам попала задача сделать интерфейс для пиццы из половинок, мы немного растерялись. Хочется и красиво, и наглядно, и удобно, и крупно, и интерактивно и много как ещё. Хочется сделать круто.

Дизайнеры пробовали разные подходы: сетку из пицц, горизонтальные и вертикальный карточки, но остановились на свайпах половинок. Ка достичь такого результата мы не знали, поэтому начали с эксперимента и взяли две недели на прототип. Даже сырой макет смог порадовать каждого. Реакцию записывали на видео:

Свайп половинок в Додо Пицце

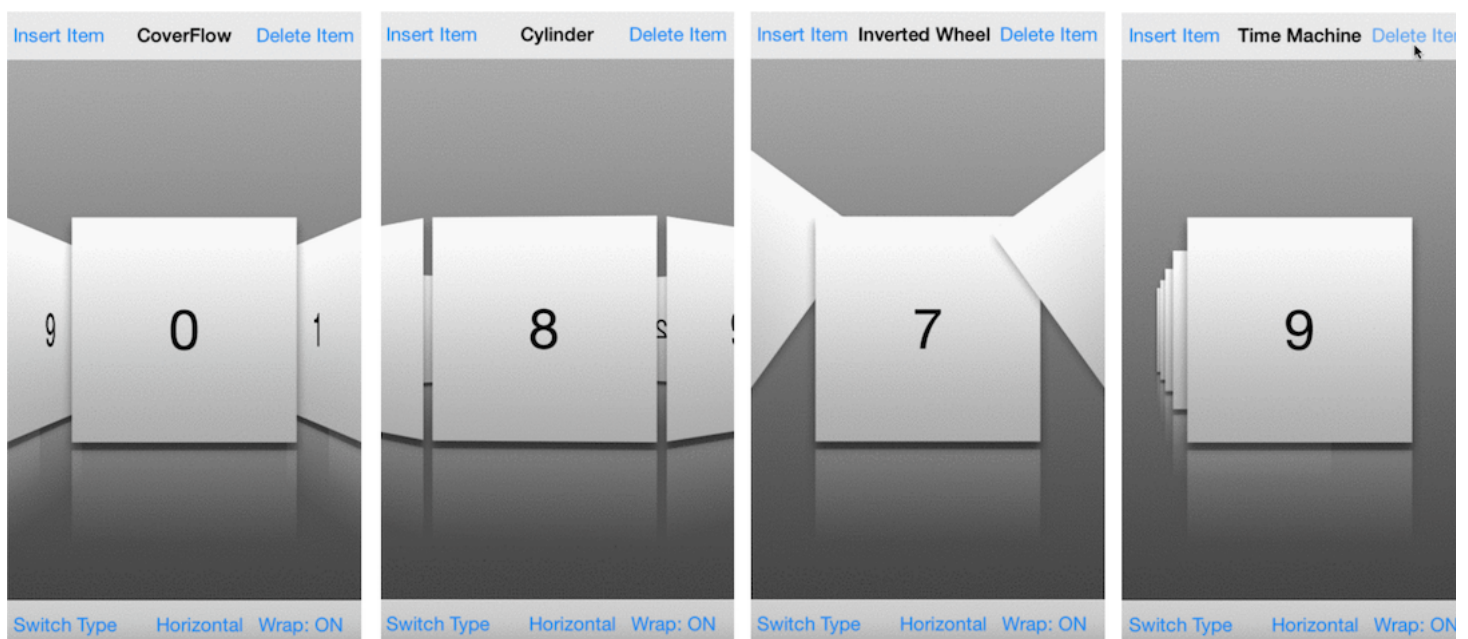


Как работает UICollectionView

UICollectionView — это сабкласс от UIScrollView, а он — это обычный UIView, у которого от свайпа меняется bounds. Перемещая его `.origin` мы смещаем видимую зону, а меняя `.size` влияем на масштаб.

При смещении экрана UICollectionView создаёт (или повторно использует) ячейки, а правила их отображения описаны в UICollectionViewLayout. С ним мы и будем работать.

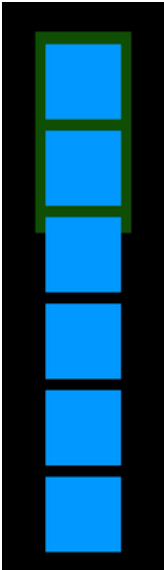
Возможности у UICollectionViewLayout большие, можно задать любое отношение между ячейками. Например, можно сделать очень похоже на то, что умеет [iCarousel](#):



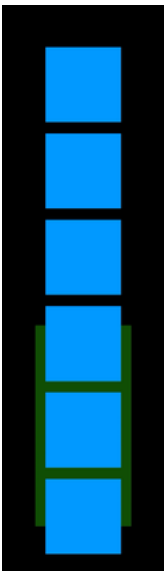
Первый подход

Смена взгляда на перемещение экрана помогла мне проще понять устройство лейаута.

Мы привыкли, что ячейки перемещаются по экрану (зелёный прямоугольник — это экран телефона):



Но всё наоборот, это экран перемещается относительно ячеек. Деревья неподвижные, это поезд едет:



На примере фреймы ячеек не меняются, а изменяется bounds самого коллекшена. `Origin` этого bounds — известный нам `contentOffset`.

Для создания лейаута надо пройти два этапа:

- просчитать размеры всех ячеек
- показать на экране только видимые.

Простой лейаут как в UITableView

Лейаут не работает с ячейками напрямую. Вместо них используются `UICollectionViewLayoutAttributes` — это набор параметров, которые будут применены к ячейке. `Frame` — основной из них, отвечает за положение и размер ячейки. Другие параметры: прозрачность, смещение, положение в глубине экрана и т.д.

```
@available(iOS 6.0, *)
open class UICollectionViewLayoutAttributes : NSObject, NSCopying, UIDynamicItem {

    open var frame: CGRect

    open var center: CGPoint

    open var size: CGSize

    open var transform3D: CATransform3D

    @available(iOS 7.0, *)
    open var bounds: CGRect

    @available(iOS 7.0, *)
    open var transform: CGAffineTransform

    open var alpha: CGFloat

    open var zIndex: Int // default is 0

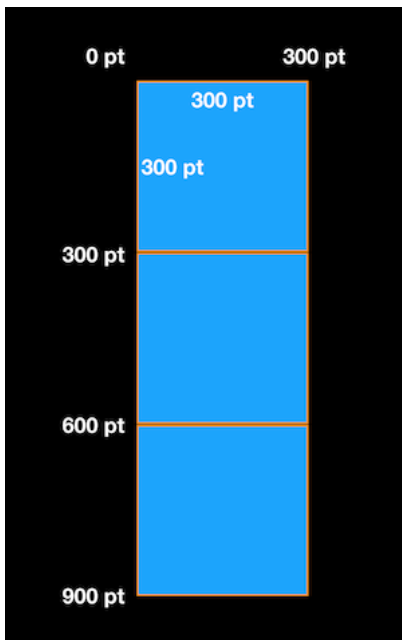
    open var isHidden: Bool // As an optimization, UICollectionView might not create a view for
                             items whose hidden attribute is YES
```

Для начала напишем простой UICollectionViewLayout, который повторяет поведение UITableView: ячейки занимают всю ширину, идут одна за другой.

Впереди 4 шага:

- Рассчитать frame для всех ячеек в методе prepare.
- Вернуть видимые ячейки в методе layoutAttributesForElements(in:).
- Вернуть параметры ячейки по её индексу в методе layoutAttributesForItem(at:). Например, этот метод используется при вызове у коллекшена метода scrollToItem(at:).
- Вернуть размеры получившегося контента в collectionViewContentSize. Так коллекшен узнает, где граница, до которой можно скроллить

На большинстве устройств размер пиццы будет 300 точек, тогда координаты и размеры всех ячеек:



Расчёты я вынес в отдельный класс. Он состоит из двух частей: просчитывает все фреймы в конструкторе, а потом лишь даёт доступ к готовым результатам:

```
class TableLayoutCache {

    // MARK: - Calculation
    func recalculatedDefaultFrames(numberOfItems: Int) {
        defaultFrames = (0..

```

```
private var defaultFrames = [CGRect]()  
}
```

Тогда в классе лейаута нужно только передать параметры из кэша.

1. Метод `prepare` вызывает расчёт всех фреймов.
2. `layoutAttributesForElements(in:)` отфильтрует фреймы. Если фрейм пересекается с видимой областью, то значит ячейку нужно отобразить: рассчитать все атрибуты и вернуть её в массиве видимых ячеек.
3. `layoutAttributesForItem(at:)` - рассчитывает атрибуты для одной ячейки.

```
class TableLayout: UICollectionViewLayout {  
  
    override var collectionViewContentSize: CGSize {  
        return cache.contentSize  
    }  
  
    override func prepare() {  
        super.prepare()  
  
        let numberOfItems = collectionView!.numberOfItems(inSection: section)  
  
        cache = TableLayoutCache(itemSize: itemSize,  
                                collectionWidth: collectionView!.bounds.width)  
        cache.recalculateDefaultFrames(numberOfItems: numberOfItems)  
    }  
  
    override func layoutAttributesForElements(in rect: CGRect) -> [UICollectionViewLayoutAttributes]? {  
        let indexes = cache.visibleRows(in: rect)  
  
        let cells = indexes.map { (row) -> UICollectionViewLayoutAttributes? in  
            let path = IndexPath(row: row, section: section)  
            let attributes = layoutAttributesForItem(at: path)  
            return attributes  
        }.compactMap { $0 }  
  
        return cells  
    }  
  
    override func layoutAttributesForItem(at indexPath: IndexPath) -> UICollectionViewLayoutAttributes? {  
  
        let attributes = UICollectionViewLayoutAttributes(forCellWith: indexPath)  
        attributes.frame = cache.defaultCellFrame(atRow: indexPath.row)  
  
        return attributes  
    }  
  
    var itemSize: CGSize = .zero {  
        didSet {  
            invalidateLayout()  
        }  
    }  
  
    private let section = 0  
    var cache = TableLayoutCache.zero  
}
```

Меняем под свои нужды

С табличным представлением разобрались, но теперь нам нужно сделать динамичный лейаут. При каждом смещении пальца будем пересчитывать атрибуты ячеек: брать фреймы, которые уже посчитали, и менять их с помощью `.transform`. Все изменения будем делать в подклассе `PizzaHalfSelectorLayout`.

Считаем индекс текущей пиццы

Для удобства можно забыть про `contentOffset` и заменить его номером текущей пиццы. Тогда больше не нужно будет думать о координатах все решения будут вокруг номера пиццы и степени смещения её от центра экрана.

Нужно два метода: один конвертирует `contentOffset` в номер пиццы, второй наоборот:

```
extension PizzaHalfSelectorLayout {  
  
    func contentOffset(for pizzaIndex: Int) -> CGPoint {  
        let cellHeight = itemSize.height  
        let screenHalf = collectionView!.bounds.height / 2  
  
        let midY = cellHeight * CGFloat(pizzaIndex) + cellHeight / 2  
        let newY = midY - screenHalf  
  
        return CGPoint(x: 0, y: newY)  
    }  
  
    func pizzaIndex(offset: CGPoint) -> CGFloat {  
        let cellHeight = itemSize.height  
  
        let proposedCenterY = collectionView!.screenCenterYOffset(for: offset)  
        let pizzaIndex = proposedCenterY / cellHeight  
  
        return pizzaIndex  
    }  
}
```

Расчёт `contentOffset` для центра экрана вынесен в extension:

```
extension UIScrollView {  
    func screenCenterYOffset(for offset: CGPoint? = nil) -> CGFloat {  
        let offsetY = offset?.y ?? contentOffset.y  
        let contentOffsetY = offsetY + bounds.height / 2  
  
        return contentOffsetY  
    }  
}
```

Останавливаемся на пицце в центре

Первое, что нам нужно сделать — останавливать пиццу в центре экрана. Метод `targetContentOffset(forProposedContentOffset:)` спрашивает, где остановиться, если с текущей скоростью он собирался остановиться в `proposedContentOffset`.

Расчёт простой: посмотреть в какую пиццу попадёт `proposedContentOffset` и проскролить так, чтобы она встала в центре:

```
override func targetContentOffset(forProposedContentOffset proposedContentOffset: CGPoint,  
                                  withScrollingVelocity velocity: CGPoint) -> CGPoint {  
    let pizzaIndex = Int(self.pizzaIndex(offset: proposedContentOffset))  
    let projectedOffset = contentOffset(for: pizzaIndex)  
  
    return projectedOffset  
}
```

У `UIScrollView` есть две скорости прокрутки: `.normal` и `.fast`. Нам больше подойдёт `.fast`:

```
collectionView!.decelerationRate = .fast
```

Но есть одна проблема: если мы проскролили совсем чуть-чуть, то нужно остаться на пицце, а не перескакивать на следующую. Метода для изменения скорости нет, поэтому обратный отскок хоть и на маленькое расстояние, но с очень большой скоростью:



Осторожно, хак!

Если ячейка не меняется, то мы возвращаем текущий `contentOffset`, так скролл остановится. Затем, мы сами скролим до прежнего места с помощью стандартного `scrollToItem`. Увы, скролить придётся ещё и асинхронно, чтобы код вызывался уже после `return`, тогда не будет маленького замирания во время анимации.

```
override func targetContentOffset(forProposedContentOffset proposedContentOffset: CGPoint,
                                  withScrollingVelocity velocity: CGPoint) -> CGPoint {
    let pizzaIndex = Int(self.pizzaIndex(offset: proposedContentOffset))
    let projectedOffset = contentOffset(for: pizzaIndex)

    let sameCell = pizzaIndex == currentPizzaIndexInt
    if sameCell {
        animateBackwardScroll(to: pizzaIndex)
        return collectionView!.contentOffset // Stop scroll, we've animated manually
    }

    return projectedOffset
}

/// A bit of magic. Without that, high velocity moves cells backward very fast.
/// We slow down the animation
private func animateBackwardScroll(to pizzaIndex: Int) {
    let path = IndexPath(row: pizzaIndex, section: 0)

    collectionView?.scrollToItem(at: path,
                                at: .centeredVertically, animated: true)

    // More magic here. Fix double-step animation.
    // You can't do it smoothly without that.
    DispatchQueue.main.async {
        self.collectionView?.scrollToItem(at: path,
                                            at: .centeredVertically, animated: true)
    }
}
```

Проблема ушла, теперь пицца возвращается плавно:



Увеличиваем центральную пиццу

Пересчитываем лейаут при движении

Нужно сделать так, чтобы центральная пицца плавно увеличивалась при приближении к центру. Для этого рассчитывать параметры нужно не один раз на старте, а каждый раз при смещении. Включается просто:

```
override func shouldInvalidateLayout(forBoundsChange newBounds: CGRect) -> Bool {  
    return true  
}
```

Теперь при каждом смещении будут вызываться методы `prepare` и `layoutAttributesForElements(in:)`. Так мы сможем обновлять `UICollectionViewLayoutAttributes` много раз подряд, плавно меняя положение и прозрачность.

Трансформируем ячейки

В табличном лейауте ячейки лежали друг под другом и их координаты считались один раз. Теперь будем менять их в зависимости от положения относительно центра экрана. Добавим метод, который будет изменять их на лету.

В методе `layoutAttributesForElements` нужно получить атрибуты из суперкласса, отфильтровать атрибуты ячеек и передать их в метод `updateCells`:

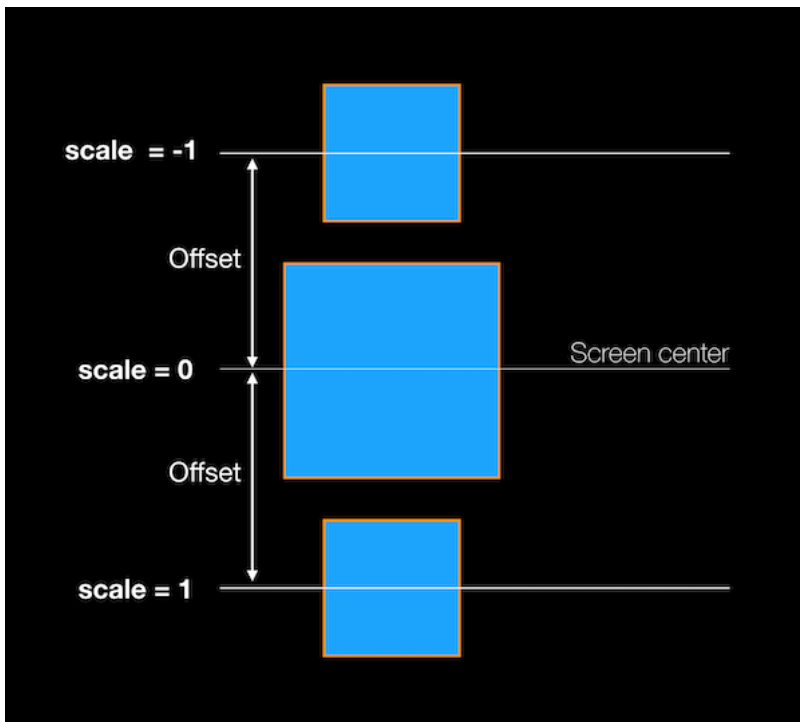
```
override func layoutAttributesForElements(in rect: CGRect) -> [UICollectionViewLayoutAttributes]? {  
    guard let elements = super.layoutAttributesForElements(in: rect) else {  
        return nil  
    }  
  
    let cells = elements.filter { $0.representedElementCategory == .cell }  
    self.updateCells(cells)  
}
```

Теперь будем менять атрибуты ячейки в одной функции:

```
private func updateCells(_ cells: [UICollectionViewLayoutAttributes])
```

Во время движения нам нужно менять прозрачность, размер и держать пиццы вдоль центра.

Положение ячейки относительно центра экрана удобно представить в нормализованном виде. Если ячейка в центре, то параметр равен 0, если смещается, то и параметр изменяется от -1 при движении вверх, до 1 при движении. Если значения стали дальше от нуля чем 1/-1, то это значит, что ячейка больше не центральная и перестала меняться. Я назвал этот параметр `scale`:



Нужно посчитать разницу между центром фрейма и центром экрана. Разделив разницу на константу, мы нормализуем значение, а min и max приведут к диапазону от -1 до +1:

```
extension PizzaHalfSelectorLayout {

    func scale(for row: Int) -> CGFloat {
        let frame = cache.defaultCellFrame(atRow: row)
        let scale = self.scale(for: frame)
        return scale.normalized
    }

    func scale(for frame: CGRect) -> CGFloat {
        let criticalOffset = PizzaHalfSelectorLayout.criticalOffsetFromCenter // 200 pt
        let centerOffset = offsetFromScreenCenter(frame)
        let relativeOffset = centerOffset / criticalOffset

        return relativeOffset
    }

    func offsetFromScreenCenter(_ frame: CGRect) -> CGFloat {
        return frame.midY - collectionView!.screenCenterYOffset()
    }
}

extension CGFloat {
    var normalized: CGFloat {
        return CGFloat.minimum(1, CGFloat.maximum(-1, self))
    }
}
```

Размер

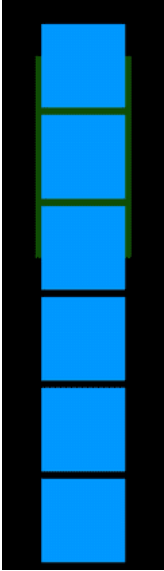
Имея нормализованный scale, можно делать что угодно. Изменения от -1 до +1 слишком сильные, для размера их нужно преобразовать. Например, мы хотим, чтобы размер уменьшался максимум до 0.6 от размера центральной пиццы:

```
private func updateCells(_ cells: [UICollectionViewLayoutAttributes]) {
    for cell in cells {
        let normScale = scale(for: cell.indexPath.row)
    }
}
```

```
let scale = 1 - PizzaHalfSelectorLayout.scaleFactor * abs(normScale)

cell.transform = CGAffineTransform(scaleX: scale, y: scale)
}
```

`.transform` изменяет размер относительно центра ячеек. У центральной ячейки `normScale = 0`, её размер не меняется:

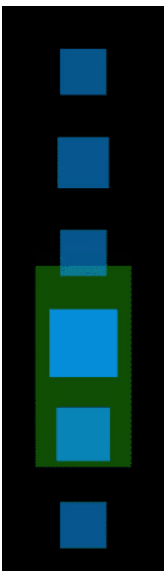


Прозрачность

Прозрачность можно поменять через параметр `alpha`. Подойдёт тоже значение `scale`, которое мы использовали в `transform`.

```
cell.alpha = scale
```

Теперь пицца меняет размер и прозрачность. Уже не так скучно, как в обычных таблицах.



Делим пополам

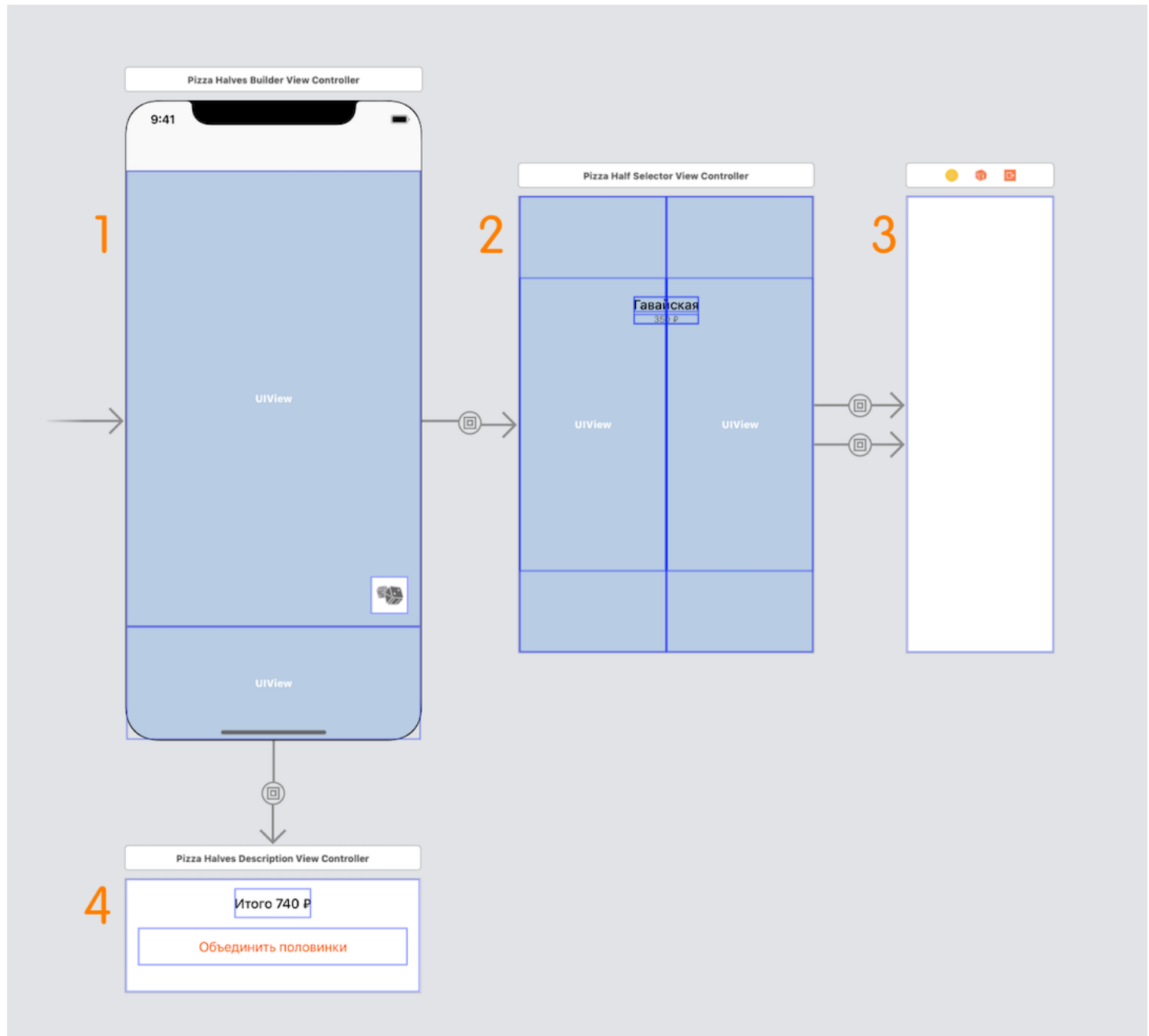
До этого мы работали с одной пиццей: задали систему отсчёта от центра, изменили размер и прозрачность. Теперь нужно поделить пополам

Использовать один коллекшен для этого слишком сложно: нужно будет писать свой обработчик жеста для каждой половины. Проще сделать

два коллекшена, в каждом свой лейаут. Только теперь вместо целой пиццы будут половинки.

Два контроллера, один контейнер

Почти всегда я разбиваю один экран на несколько UIViewController, каждый со своей задачей. В этот раз получилось так:



1. Основной контроллер: в нём собираются все части и кнопка «перемешать».
2. Контроллер с двумя контейнерами для половинок, центральной подписью и скрол индикаторами.
3. Контроллер с коллекшеном (правый белый).
4. Нижняя панель с ценой.

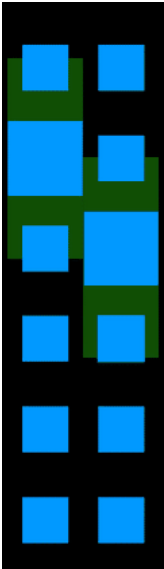
Чтобы различать левую и правую половинку, я завёл `enum`, он хранится в лейауте в проперти `.orientation`:

```
enum PizzaHalfOrientation {  
    case left  
    case right  
}
```

```
func opposite() -> PizzaHalfOrientation {
    switch self {
        case .left: return .right
        case .right: return .left
    }
}
```

Смещаем половинки к центру

Прежний лейаут перестал делать то, что мы ожидаем: половинки стали смещаться к центру своих коллекшенов, не к центру экрана:



Исправить просто: нужно горизонтально сместить ячейки наполовину к центру экрана:

```
func centerAlignedFrame(for element: UICollectionViewLayoutAttributes, scale: CGFloat) -> CGRect {

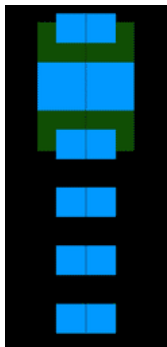
    let hOffset = horizontalOffset(for: element, scale: scale)

    switch orientation {
        case .left: // Align to right
            return element.frame.offsetBy(dx: +hOffset - spaceBetweenHalves / 2, dy: 0)
        case .right: // Align to left
            return element.frame.offsetBy(dx: -hOffset + spaceBetweenHalves / 2, dy: 0)
    }
}

private func horizontalOffset(for element: UICollectionViewLayoutAttributes, scale: CGFloat) -> CGFloat {
    let collectionViewWidth = collectionView!.bounds.width
    let scaledElementWidth = element.frame.width * scale
    let hOffset = (collectionViewWidth - scaledElementWidth) / 2

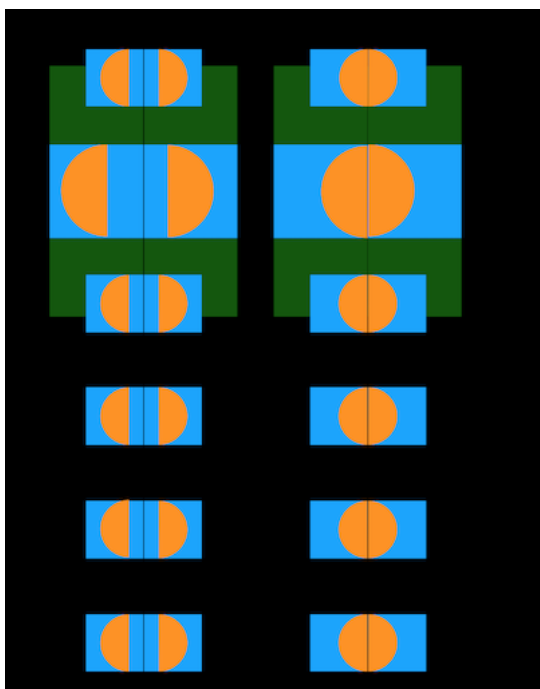
    return hOffset
}
```

Тут же контролируется расстояние между половинками.



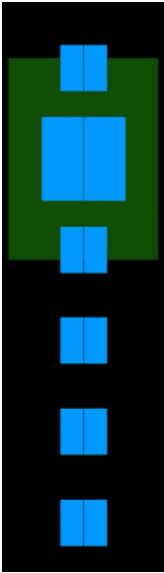
Смещение внутри ячейки

Круглую пиццу легко было вписать в квадрат, а для половинки нужно пол квадрата:



Можно переписать расчёт фреймов: уменьшить ширину вдвое, выровнять фреймы к центру по-разному для каждой половины. Для простоты всего лишь поменяем contentMode картинки уже внутри ячейки:

```
class PizzaHalfCell: UICollectionViewCell {
    var halfOrientation: PizzaHalfOrientation = .left {
        didSet {
            imageView?.contentMode = halfOrientation == .left ? .topRight : .topLeft
            self.setNeedsLayout()
        }
    }
}
```



Прижимаем пиццы по вертикали

Пиццы уменьшились, но расстояние между их центрами не изменилось, появились большие разрывы. Компенсировать их можно так же, как мы выравнивали половинки по центру.

```
private func verticalOffset(for element: UICollectionViewLayoutAttributes, scale: CGFloat) -> CGFloat {
    let offsetFromCenter = offsetFromScreenCenter(element.frame)
    let vOffset: CGFloat = PizzaHalfSelectorLayout.verticalOffset(
        offsetFromCenter: offsetFromCenter,
        scale: scale)

    return vOffset
}

static func verticalOffset(offsetFromCenter: CGFloat,
                           scale: CGFloat) -> CGFloat {
    return -offsetFromCenter / 4 * scale
}
```

В итоге, все компенсации выглядят вот так:

```
func centerAlignedFrame(for element: UICollectionViewLayoutAttributes, scale: CGFloat) -> CGRect {

    let hOffset = horizontalOffset(for: element, scale: scale)
    let vOffset = verticalOffset (for: element, scale: scale)

    switch orientation {
    case .left: // Align to right
        return element.frame.offsetBy(dx: +hOffset - spaceBetweenHalves / 2,
                                       dy: vOffset)
    case .right: // Align to left
        return element.frame.offsetBy(dx: -hOffset + spaceBetweenHalves / 2,
                                       dy: vOffset)
    }
}
```

А настройка ячейки — вот так:

```
private func updateCells(_ cells: [UICollectionViewLayoutAttributes]) {
    for cell in cells {
        let normScale = scale(for: cell.indexPath.row)
        let scale = 1 - PizzaHalfSelectorLayout.scaleFactor * abs(normScale)
    }
}
```

```
        cell.alpha      = scale
        cell.frame      = centerAlignedFrame(for: cell, scale: scale)
        cell.transform  = CGAffineTransform(scaleX: scale, y: scale)
        cell.zIndex     = cellZLevel
    }
}
```

Не перепутай: настройка фрейма должна быть до трансформации. Если поменять порядок, то результат вычислений будет совсем другой.

Готово! Мы разрезали половинки и выровняли их к центру:



Добавляем подписи

Хедеры создаются так же как и ячейки, только вместо UICollectionViewLayoutAttributes(forCellWith:) нужно использовать конструктор UICollectionViewLayoutAttributes(forSupplementaryViewOfKind:) и вернуть их вместе с параметрами ячеек в layoutAttributesForElements(in:)

Сначала опишем метод для получения хедера по IndexPath:

```
override func layoutAttributesForSupplementaryView(ofKind elementKind: String,
                                                    at indexPath: IndexPath)
    -> UICollectionViewLayoutAttributes? {

    let attributes = UICollectionViewLayoutAttributes(
        forSupplementaryViewOfKind: elementKind, with: indexPath)

    attributes.frame = defaultFrameForHeader(at: indexPath)
    attributes.zIndex = headerZLevel

    return attributes
}
```

Расчёт фрейма спрятан в методе defaultFrameForHeader (будет позже).

Теперь можно получить IndexPath видимых ячеек и показать подписи для них:

```
override func layoutAttributesForElements(in rect: CGRect) -> [UICollectionViewLayoutAttributes]? {
    ...

    let visiblePaths = cells.map { $0.indexPath }
```



```
    let headers = self.headers(for: visiblePaths)
    updateHeaders(headers)

    return cells + headers
}
```

Ужасно длинный вызов функций спрятан в методе headers(for:):

```
func headers(for paths: [IndexPath]) -> [UICollectionViewLayoutAttributes] {
    let headers: [UICollectionViewLayoutAttributes] = paths.map {
        layoutAttributesForSupplementaryView(
            ofKind: UICollectionView.elementKindSectionHeader, at: $0)
    }.compactMap { $0 }

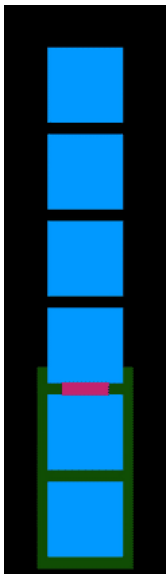
    return headers
}
```

zIndex

Сейчас ячейки и подписи находятся на одном уровне «высоты», поэтому могут наслаиваться друг на друга. Чтобы заголовки всегда были выше, поставьте им zIndex больше нуля. Например, 100.

Фиксируем позицию (на самом деле нет)

Фиксированные на экране подписи немного ломают голову. Вы хотите зафиксировать, а нужно наоборот, постоянно двигать вместе с bounds



В коде всё просто: получаем положение подписи на экране и смещаем его на contentOffset:

```
func defaultFrameForHeader(at indexPath: IndexPath) -> CGRect {
    let inset = max(collectionView!.layoutMargins.left,
                    collectionView!.layoutMargins.right)

    let y = collectionView!.bounds.minY
    let height = collectionView!.bounds.height
    let width = collectionView!.bounds.width

    let headerWidth = width - inset * 2
    let headerHeight: CGFloat = 60
    let vOffset: CGFloat = 30

    let screenY = (height - itemSize.height) / 2 - headerHeight / 2 - vOffset
```

```
return CGRect(x: inset,
              y: y + screenY,
              width: headerWidth,
              height: headerHeight)
}
```

Высота у подписей может быть разной, считать её лучше в делегате (и кешировать там же).

Анимлируем подписи

Всё очень похоже на ячейки. Опираясь на текущий `scale`, можно рассчитывать прозрачность ячейки. Смещение можно задать через `.transform`, так надпись будет смещаться по отношению к своему фрейму:

```
func updateHeaders(_ headers: [UICollectionViewLayoutAttributes]) {
    for header in headers {
        let scale = self.scale(for: header.indexPath.row)

        let alpha = 1 - abs(scale)
        header.alpha = alpha

        let translation = 20 * scale
        header.transform = CGAffineTransform(translationX: 0,
                                                    y: translation)
    }
}
```



Оптимизируем

После добавления заголовков производительность сильно просела. Так получилось, потому что мы скрыли подписи, но всё равно возвращаем их `UICollectionViewLayoutAttributes`. От этого хедеры добавляются в иерархию, участвуют в лейауте, но не отображаются. Ячейки мы показывали только те, которые пересекаются с текущим `bounds`, а хедеры нужно фильтровать по `alpha`:

```
override func layoutAttributesForElements(in rect: CGRect) -> [UICollectionViewLayoutAttributes]? {
    ...

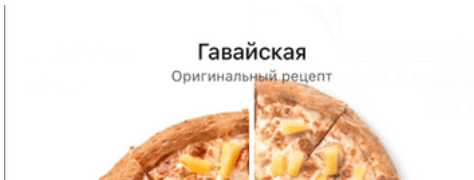
    let visibleHeaders = headers.filter { $0.alpha > 0 }

    return cells + visibleHeaders
}
```

Согласовываем с центральной подписью (оригинальный рецепт)

Мы проделали большую работу, но в интерфейсе нашлось противоречие — если выбрать две одинаковые половинки, то они превращаются в обычную пиццу.

Мы решили так и оставить, но правильно обработать состояние, показав, что это обычная пицца. Наша новая задача — для одинаковых пицц показать одну надпись по центру, а по краям скрывать.



Одним только лейаутом решить такое слишком сложно, потому что надпись на стыке двух коллекшенов. Получится проще, если контроллер, который содержит в себе обе коллекции, согласует движение всех подписей.

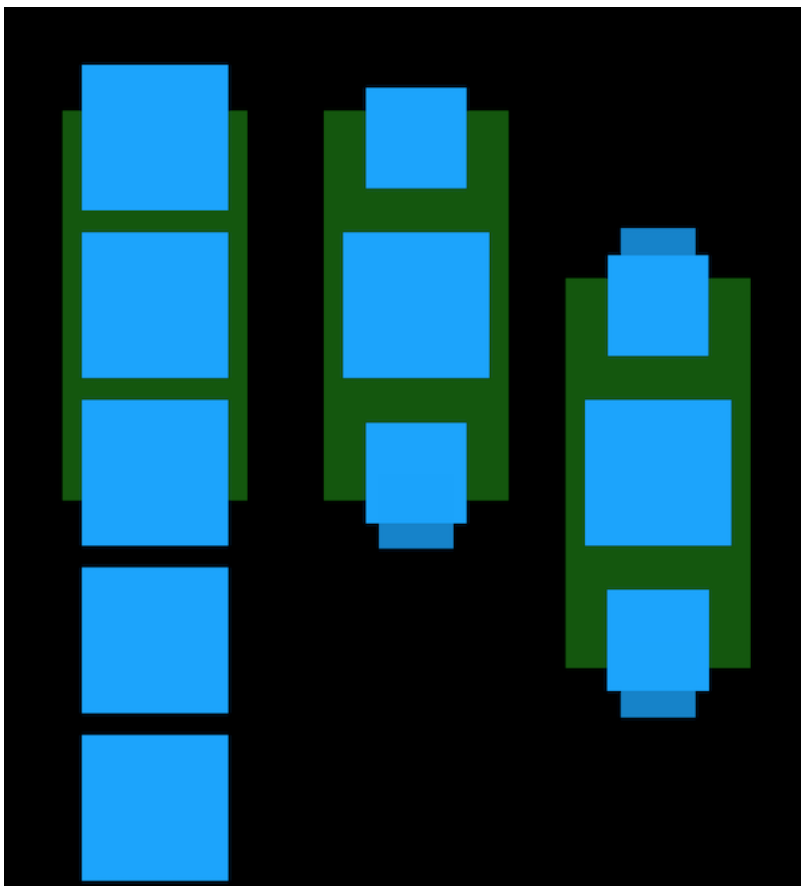
При скроле мы передаём текущий индекс в контроллер, он отправляет индекс в противоположную половинку. Если индексы совпадают, то о показывает заголовок оригинальной пиццы, а если разные, то видны подписи для каждой половинки.

Как придумывать свои лейауты

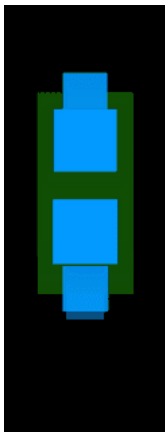
Самым сложным было понять, как переложить свою идею на вычисления. Например, я хочу, чтобы пиццы скроились как барабан. Для понимания задачи я прошёл 4 обычных шага:

1. Нарисовал пару состояний.
2. Понял, как элементы связаны с положением экрана (элементы двигаются относительно центра экрана).
3. Создал переменные, с которыми удобно работать (центр экрана, фрейм центральной пиццы, scale).
4. Придумал простые шаги, каждый из которых можно проверить.

Состояния и анимации легко рисовать в Keynote. Я взял стандартную раскладку и нарисовал два первых шага:



На видео получается так:



Понадобилось три изменения:

1. Вместо фреймов из кеша будем брать centerPizzaFrame.
2. С помощью scale считать офсет от этого фрейма.
3. Пересчитывать zIndex.

```
func centerAlignedFrame(for element: UICollectionViewLayoutAttributes, scale: CGFloat) -> CGRect {
    let hOffset = self.horizontalOffset(for: element, scale: scale)
    let vOffset = self.verticalOffset (for: element, scale: scale)

    switch self.pizzaHalf {
    case .left: // Align to right
        return centerPizzaFrame.offsetBy(dx: hOffset - spaceBetweenHalves / 2,
                                           dy: vOffset)
    case .right: // Align to left
        return centerPizzaFrame.offsetBy(dx: -hOffset + spaceBetweenHalves / 2,
                                           dy: vOffset)
    }
}

private func horizontalOffset(for element: UICollectionViewLayoutAttributes, scale: CGFloat) -> CGFloat {
    let collectionWidth = self.collectionView!.bounds.width
    let scaledElementWidth = centerPizzaFrame.width * scale
    let hOffset = (collectionWidth - scaledElementWidth) / 2

    return hOffset
}

private func verticalOffset(for element: UICollectionViewLayoutAttributes, scale: CGFloat) -> CGFloat {
    let totalProgress = self.scale(for: element.frame).normalized(by: 1)
    let criticalOffset = PizzaHalfSelectorLayout.criticalOffsetFromCenter * 1.1

    return totalProgress * criticalOffset
}
```

Раз ячейки накладываются друг на друга, то нужно задавать им правильный порядок с помощью zIndex. Идея такая: чем ячейка ближе к центральной пицце, тем ближе она к экрану, и тем больше zIndex.

```
private func updateCells(_ cells: [UICollectionViewLayoutAttributes]) {
    for cell in cells {
        let normScale = self.scale(for: cell.indexPath.row)
        let scale = 1 - PizzaHalfSelectorLayout.scaleFactor * abs(normScale)

        cell.alpha = 1//scale
    }
}
```

```
cell.frame = self.centerAlignedFrame(for: cell, scale: scale)
cell.transform = CGAffineTransform(scaleX: scale, y: scale)
cell.zIndex = self.zIndex(row: cell.indexPath.row)
}
}

private func zIndex(row: Int) -> Int {
    let numberOfCells = self.cache.defaultFrames.count
    if row == self.currentPizzaIndexInt {
        return numberOfCells
    } else if row < self.currentPizzaIndexInt {
        return row
    } else {
        return numberOfCells - row - 1
    }
}
```

Тогда, если третья ячейка текущая, то получится вот так:

```
row: zIndex`
0: 0
1: 1
2: 2
3: 10 — текущая ячейка
4: 5
5: 4
6: 3
7: 2
8: 1
9: 0
```

В продакшен такой лейаут не попал, для этого были бы нужны картинки с прозрачным фоном.

Релиз

Создавая такой интерфейс, мы ввязались в небольшой эксперимент: впервые написали настолько необычный лейаут, а позже добавили интерактивный переход на экран карточки. Эксперимент себя оправдал: пицца из половинок ворвалась в топ продаж и заняла второе место, уступив лишь классической пепперони.

Конечно, для продакшена нужно было сделать больше работы:

- разрезать картинки пицц пополам,
- обернуть в стейт контроллер, чтобы можно было загрузить пиццы: показать загрузку, кнопку повтора или сами пиццы,
- откликаться хаптиком при переключении пицц,
- анимировать транзишен для перехода в карточку продукта,
- показать процент промотки через круглые скролл-индикаторы,
- добавить кнопку «перемешать»,
- обеспечить доступность через Voice Over.

Итоговый конструктор пицц половинок работает вот так:

Пицца из половинок



Код можно посмотреть на [github](#), а заказать пиццу из половинок в приложении.

Если хотите узнать больше о работе UICollectionViewLayout, то можно начать с лекции [A Tour of UICollectionView](#)

А если вам интересны события поменьше, то подписывайтесь [на канал в телеграмме](#).

Теги: [iOS](#), [UIKit](#), [dodomobile](#), [UICollectionViewLayout](#)

↑ +29 ↓ 44 5,2k 11



Dodo Pizza Engineering 191,00

О том как IT доставляет пиццу



6,0

Карма

30,3

Рейтинг

17

Подписчики

Рубанов Михаил @akaDuality

iOS-developer, Dodo Pizza

Сайт

Поделиться публикацией

ПОХОЖИЕ ПУБЛИКАЦИИ

25 апреля 2019 в 18:25

Я прочитал 80 резюме, у меня есть вопросы

↑ +41 83,2k 406 634

27 декабря 2018 в 13:05

Контроллер-луковка. Разбиваем экраны на части

↑ +16 5,5k 43 2

10 декабря 2018 в 19:59

Контроллер, полегче! Выносим код в UIView

↑ +31

👁 8,6k

📌 57

💬 12



ВАКАНСИИ КОМПАНИИ DODO PIZZA ENGINEERING

Мой круг

UX/UI Designer в B2B

Москва • Полный рабочий день

Senior .NET Developer / Старший Разработчик C#

Москва • Полный рабочий день

от 160000 до 200000 ₽

Вакансии компании

[Создать резюме](#)

Комментарии 11



TheGodfather

23 мая 2019 в 23:25



0

Код можно посмотреть на github

Я уж подумал, что ссылка будет на репозиторий Додо, что решили хотя бы частично заопенсорсить свою ИС... Но нет.



akaDuality

24 мая 2019 в 09:23



0

Частично заопенсорсили :D



Sinner007

24 мая 2019 в 12:34



0

А где промо код =)) надо было в конце статьи приложить для тех кто дочитал =))



saniaxxx

24 мая 2019 в 14:54



0

Познавательная статья, а вот пиццы пробовал — не понравились



akaDuality

24 мая 2019 в 15:01



0

Можете написать мне в личные сообщения: что не понравилось и какой адрес? Я передам партнёру.



egordeev

24 мая 2019 в 18:11



0

а если из 4х разных частей?



akaDuality

24 мая 2019 в 18:32



0

Можно заказать пиццу «4 сезона» или какое-нибудь комбо из нескольких пицц



egordeev

24 мая 2019 в 18:37



+1

то есть вы отдельный layout делали, чтобы 4х разных частей пиццу собрать?



akaDuality

24 мая 2019 в 18:40



0

Не, есть пицца которая состоит из 4 фиксированных частей. Отличный вариант попробовать несколько вкусов.

Вот так выглядит



anivaros

сегодня в 13:40



0

Например, iCarousel умеет вот так

В iCarousel нет UICollectionView, там полностью самописный движок.

 **akaDuality** сегодня в 14:50    

Спасибо, поправил в тексте.

 0

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

САМОЕ ЧИТАЕМОЕ

Сутки

Неделя

Месяц

Болен-здоров

 +81

 20,9k

 70

 84

Huawei получает новый удар: компания лишилась возможности выпускать смартфоны с microSD

 +15

 23,3k

 9

 150

Изучаем здоровье спутников Starlink Илона Маска

 +51

 17,4k

 25

 38

Самая дорогая ошибка в моей жизни: подробно об атаке на порт SIM-карты

 +55

 49,4k

 126

 164



Интернет-спутники SpaceX, выстраивающиеся на орбите Земли, попали на видео

 +17

 12,4k

 4

 34

Аккаунт	Разделы	Информация	Услуги	Приложения
Войти	Публикации	Правила	Реклама	<div> </div>
Регистрация	Новости	Помощь	Тарифы	
	Хабы	Документация	Контент	
	Компании	Соглашение	Семинары	
	Пользователи	Конфиденциальность		
	Песочница			

Если нашли опечатку в посте, выделите ее и нажмите Ctrl+Enter, чтобы сообщить автору.