Wordle 2.0 App-> My replica of the popular word game Wordle, developed programmatically using UIKit and Swift to build the UI, logics and various extensions.

Functionalities/Demo:

- Similar rules to the original Wordle App. A user has six tries to guess a five-letter word. To begin, enter a word. The App will reveal whether the word entered has any correct letters. A green letter is in the correct spot, an orange letter is in the word but in an incorrect spot, and a gray letter is not in the word. The user will win by guessing the word correctly within six tries.

Developing Languages, Tools, and Techniques Needed:

Xcode 14.0.1
Xcode Simulator iOS 16.0 iPhone 14 Pro
(Compatible with iOS 15.2+)
SwiftUI
Swift 5

Prerequisites & Setup:

Create two Cocoa Class files `KeyboardViewController.swift` and `BoardViewController.swift` in `Core`.

Give keyboard and board different background colors:
`view.backgroundColor = .red/.blue`

Synchronous Developing Notes:

Main Architecture:

Add main view controllers to build architecture in `ViewController.swift`:

```
  private func addChildren(){
        addChild(keyboardVC)
        keyboardVC.didMove(toParent: self)
        keyboardVC.view.translatesAutoresizingMaskIntoConstraints =
false
        view.addSubview(keyboardVC.view)
        addChild(boardVC)
        boardVC.didMove(toParent: self)
        boardVC.view.translatesAutoresizingMaskIntoConstraints = false
        view.addSubview(boardVC.view)
   }
```

Constraints:

To make the keyboard and board view controller background colors visible add constraints:

```
  func addConstraints(){
  NSLayoutConstraint.activate([
  boardVC.view.leadingAnchor.constraint(equalTo: view.leadingAnchor),
  boardVC.view.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
  boardVC.view.topAnchor.constraint(equalTo: view.topAnchor),
  boardVC.view.bottomAnchor.constraint(equalTo:
keyboardVC.view.topAnchor),
```

```
   boardVC.view.heightAnchor.constraint(equalTo: view.heightAnchor,
multiplier: 0.6),
   keyboardVC.view.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
   keyboardVC.view.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
   keyboardVC.view.bottomAnchor.constraint(equalTo:
view.bottomAnchor)}
```

After adding constraints, the VCs background color displayed:

vc background colors displayed.PNG

Keyboard:

In `KeyboardViewController.swift`, create keyboard collection views:

```
 class KeyboardViewController: UIViewController {
    let letters = ["qwertyuiop", "asdfghjkl", "zxcvbnm"]
    private var keys: [[Character]] = []
    let collectionView: UICollectionView = {
        let layout = UICollectionViewFlowLayout()
        layout.minimumInteritemSpacing = 2
        let collectionView = UICollectionView(frame: .zero,
collectionViewLayout: layout)
        collectionView.translatesAutoresizingMaskIntoConstraints =
false
        collectionView.backgroundColor = .yellow
        return collectionView
    }()
```

Run the simulator to test if Keyboard UI Collection VIew working:

keyboard ui collection view works.PNG

Delegate Data Source:

Assign delegate and datasource to self in viewdidload function:

```
  collectionView.delegate = self
  collectionView.dataSource = self
```

Keys:

Create keyboard view controller extensions components.

Initialize in `KeyCell.swift`:

```
       override init(frame: CGRect) {
        super.init(frame: frame)
        backgroundColor = .systemGray5
    }
    required init?(coder: NSCoder){
        fatalError()
    }
```

Now keyboard-shaped squares displayed underneath:

keyboard squares displayed.PNG

Centering the Keyboard Cells:

Adjust keyboard cells alignment to make them centered:

```
     func collectionView(_ collectionView: UICollectionView, layout
collectionViewLayout: UICollectionViewLayout, insetForSectionAt
section: Int) -> UIEdgeInsets {
        var left: CGFloat = 1
        var right: CGFloat = 1
        let margin: CGFloat = 20
        let size: CGFloat = (collectionView.frame.size.width -
margin)/10
        let count: CGFloat =
CGFloat(collectionView.numberOfItems(inSection: section))
        let inset: CGFloat = (collectionView.frame.size.width - (size
* count) - (2 * count))/2
        left = inset
        right = inset
        return UIEdgeInsets(
            top: 2,
            left: left,
            bottom: 2,
            right: right
        )
```

Now keyboard cells are all centered:

keyboard cell centered.PNG

Board Controller:

Configure in `KeyboardViewController.swift`:

```
 let letter = keys[indexPath.section][indexPath.row]
        cell.configure(with: letter)
        return cell
```

Now all keys letters are visible:

keys letters displayed.PNG

In `BoardViewController.swift`:

Similar to Keyboard view controller, set all board characters with A to test out:

```
func collectionView(_ collectionView: UICollectionView, cellForItemAt
indexPath: IndexPath) -> UICollectionViewCell {guard let cell =
collectionView.dequeueReusableCell(withReuseIdentifier:
KeyCell.identifier, for: indexPath) as? KeyCell else {
            fatalError()
        }
        cell.configure(with: Character("A"))
        return cell
```

wordle board A character test works.PNG

To create an interaction on the keyboard cells:

In `KeyboardViewController.swift`:

```
 protocol KeyboardViewControllerDelegate: AnyObject {
    func keyboardViewController(
        _vc: KeyboardViewController,
        didTapKey letter: Character
...
weak var delegate: KeyboardViewControllerDelegate?
```

And to use delegate:

```
 func collectionView(_ collectionView: UICollectionView,
didSelectItemAt indexPath: IndexPath) {
       //
        collectionView.deselectItem(at: indexPath, animated: true)
        let letter = keys[indexPath.section][indexPath.row]
        delegate?.keyboardViewController(_vc: self, didTapKey: letter)
    }
}
```

To get actual access to the delegate, add an extension in `ViewController.swift`:

```
extension ViewController: KeyboardViewControllerDelegate {
    func keyboardViewController(_vc: KeyboardViewController, didTapKey
letter: Character) {
        print(letter)
    }
}
```

So now when we type the letter on the simulator keyboard, the letters appear on the console.

<u>Data Source:</u>

Add a protocol to get data source in `BoardViewController.swift`:

```
protocol BoardViewControllerDataSource: AnyObject {
    var currentGuesses: [[Character?]] {get}}
```

Then to get the board showing, add boardVC extension in `ViewController.swift`:

```
 extension ViewController: BoardViewControllerDataSource  {
    var currentGuesses: [[Character?]] {
        return guesses
    }
}
```

Now update these guesses and tell the board controllers to reload itself, in `BoardViewController.swift`:

```
    public func reloadData(){
        collectionView.reloadData()
    }
}
```

To make the key tap responsive and tapped letters show up on board, we need to configure data source in `BoardViewController.swift`:

```
   let guesses = datasource?.currentGuesses ?? []
        if let letter =
            guesses[indexPath.section][indexPath.row] {
            cell.configure(with: letter)
        }
```

Now when tap keyboard, the tapped letters responded and become visible on board:

tapped letters displayed on board in order.PNG

Logic:

Set up the rule: if the tapped letter exists in the word and at the correct spot, green color, if the tapped letter exists in the word and at the wrong spot, orange color, else if the tapped letter is not in the word, gray color. In `ViewController.swift`:

```
 extension ViewController: BoardViewControllerDataSource  {
    var currentGuesses: [[Character?]] {
        return guesses
    }
    func boxColor(at indexPath: IndexPath) -> UIColor? {
        let rowIndex = indexPath.section
        let count = guesses[rowIndex].compactMap({ $0 }).count
        guard count == 5 else {
            return nil
        }
        let indexedAnswer = Array(answer)
        guard let letter = guesses[indexPath.section][indexPath.row],
indexedAnswer.contains(letter) else {
            return nil
        }
        if indexedAnswer[indexPath.row] == letter {
            return .systemGreen
        }
        return .systemOrange
```

Test out with the answer word: AFTER:

wordle works with the answer AFTER.PNG

Add app name:

Other -> Main -> ViewController->View-> Editor-> Embed in-> Navigator-> Right hand bar

wordle 2.0 title displayed.PNG

Now import a bunch of A-B five-lettered words:

```
let answers = ["abuse","adult", "agent", "anger", "apple", "award",
"basis", "beach","birth", "block", "blood", "board", "brain","bread",
"break", "brown"]
```

Test word BRAIN works:

test BRAIN works.PNG