1차 평가과제 피드백- 강나리님

안녕하세요! 강나리님

수업은 잘 듣고 계실까요? 수업을 들으시면서 이해하기 어렵거나, 따라가기 어려운 부분이 있다면 편하게 질의응답 토픽에 질문을 올려보세요. 혹시 너무 기초적인 질문인 것 같더라도, 다른 동료분들도 똑같이 어려워 하고 궁금해 하는 부분일거에요. 많이 질문하고 많이 궁금해 하는 것이 실력을 쌓고 성장할 수 있는 지름길이랍니다 :)

그리고 개발자는 무엇을 만드는 일을 하는 사람들이 아닌, 문제를 해결하는 사람들이에요. 문제를 해결하는 방식에는 하나의 답만 있는 것이 아니고, 여러가지 답이 있을 수 있습니다. 내가 맞다고 믿는 것들이 어쩌면 맞지 않거나, 더 나은 방법이 존재할 수도 있구요.

여러 동료들과 문제를 적극적으로 이야기 하면서, 내가 생각하지 못한 접근법이나 더 나은 해결책을 적극적으로 공유해 보시길 권장 드립니다!!

과제 리뷰에 대해서 조금 더 빨리 피드백을 드리려고 했으나, 여러분들이 얼마나 잘 이해하고 계신 지 전체 그림을 보고 현 시점에서 적절한 의견을 드리는 것이 좋을 것 같아 조금 늦어진 점 양해부탁드려요.

멘토의 리뷰 일지 1편 - 첫번째 리뷰를 마치며..

좋은 앱 개발자가 어떤 개발자 일까? 라는 질문은 현업 실무자분들도 매일하는 고민들이라고 생각해요. 그리고 하나의 답으로 귀결되기 쉽지 않는 어려운 질문이구요.

공부를 하시는 분들의 입장에서 조금 각도를 바꿔서 표현해 본다면, 어떤 앱 개발자가 되어야 기업에서 모셔가는(채용) 개발자가 될 수 있을까? 라고 표현해 볼 수 있을 것 같아요.

개발 직군에는 Backend, Frondend, iOS app, Android app, Dev-ops, D.S 등 다양한 직군이 존재하고 각 직군별로 지향해야 하는 방향성이 조금씩은 다르다고 생각해요.

앱 개발자의 경우, Frontend와 함께 유저와 가장 가깝게 마주하는 직군이기 때문에 UI/UX, 비즈니스 로직 등 여러 측면에서 고려해야하는 부분들이 많은 직군중에 하나구요.

그렇기 때문에, 앱개발자는 기획적 요구사항과 디자인적 요구사항 + 비즈니스적 요구사항을 잘 반영하는 Product를 만들어야 하는 숙명(🤣🤣)을 가지고 있답니다.

이번 과제를 통해 명시적으로 가이드 드린 부분은 수업과 관련하여 학습이해도를 확인하기 위함이었고, 어느 정도 자율성을 드린 부분은 개발자 관점에서의 UX와 로직설계 부분을 보고자 함이었습니다.(학습한 내용을 기반으로 적용과 활용 관점)

개발자 관점에서의 UX는 iOS 전문가로서 iOS 플랫폼에 기반한 사용자 편의성과 시스템 안정성을 위한 앱 개발자의 배려라고 표현 할 수 있을 것 같아요.

사용자 입력 값에 숫자만 들어가야 하는 경우, keyboard를 number pad를 사용하여 사용자 편의성과 string type이 입력되지 않도록 방지하는 효과를 주고, 로직적으로도 값이 string type으로 들어온 경우를 체크하여 alert으로 "숫자 값을 입력해주세요!" 가이드를 주는 부분들이 개발자 관점의 UX라고 생각해요.

예를 들어 '밥 먹기', '물먹기'을 입력하는 곳도 숫자 제한에 대한 요구 사항이 없었더라도, 여러분들이 스스로 제한을 두셔야 해요. 만약 해당 다마고치의 정보가 서버와 연동되는 앱인데, 서버에 숫자 값이 아닌 string 데이터를 전송해 버리면 서버에서 처리를 못하고 서버가 다운될 수도 있어요.

물론 서버에서도 iOS에서 전송하는 데이터를 숫자 데이터만 받을 수 있도록 처리를 해줘야 되구요. iOS에서는 유저가 실수할 수 있는 부분이나 서비스 의도에 맞게 사용할 수 있도록 시스템적으로 적절히 제한 및 예외처리를 해주고, 서버는 앱에서 놓칠 수 있는 부분을 한번 더 체크해서 안정으로 서비스가 운영될 수 있도록 노력 해야해요. 개발 전반적으로 그러한 시각을 갖는 것 또한, 개발자의 중요한 역량 중 하나이기 때문에 고려하지 못한 부분이 없는지 한번 더 생각해 보시면 좋을 것 같습니다.

조건 분기처리를 확인하기 위해, 다마고치에서 한번에 먹을 수 있는 밥알과 물양에 대한 제한을 드렸는데요. 만약 이 제한이 없다면 어떤 일이 발생할 수 있을까요?

밥알 먹기에 99개 제한이 없다면 유저 중 누군가는 한번에 10000000000000000개의 밥알을 먹일 수 있고, 먹은 밥알을 표시하는 곳이 해당 숫자를 표현하기 위해 많은 영역을 사용하게 되면서 UI가 깨질 수 있어요. 로직 에러가 발생할 확률도 높아질 수 있구요.

한명의 개발자가 모든 개발을 담당한다면 문제가 적겠지만, 2~3명 혹은 그 이상의 개발자가 함께 개발하는 앱이라면 충분히 발생할 수 있는 문제라고 볼 수 있을 것 같아요.

같이 개발하는 다른 개발자분이 해당 값을 가져다가 쓰시는데, 메모리 효율성을 고려하는 개발자라 type에 int16을 사용하셨다면 에러가 발생할 수 있겠죠?

또, 나중에 사용자 편의성을 위해 서버가 연동되어 사용자 입력값을 서버로 전송해야 할 때도, 서버 개발자는 '다마고치가 한번에 밥알 100000개씩 먹을 일은 없겠지?' 라고 생각하며 작은 int형 type을 사용하셨을 수도 있구요.

개발하는 팀에서 이런 부분들에 대해 커뮤니케이션을 하는게 매우 중요하지만, 사람마다 당연하게 생각하는 부분이나 기준이 다르기 때문에 실무에서는 정말 다양한 문제들이 발생할 수 있어요.(특히 사용자 입력값에서요!)

그런 부분까지 하나하나 신경 써야 하나요? 라고 생각하실 수 있겠지만, 나 혼자 만들어보고 사용하는 앱과 사용자가 사용하는 앱은 많은 곳에서 차이가 발생해요.

서비스를 운영하면서 마주하는 error log를 보면, 정말 내가 생각하지 못한 다양한 use case가 존재하는 것을 경험하게 되거든요.

현업에서도 사용자가 없는 프로젝트성 앱을 여러 개 만든 개발자보다, 하나라도 출시하여 사용자들이 사용하는 앱을 개발&운영해 본 개발자를 선호하는 이유도 위와 같은 이유 때문이에요.

경험적으로 UX나 로직을 구현할 때 더 많은 것들을 고려하여 안정적으로 서비스가 동작되게 개발을 하게 되거든요!

그래서 채용기업에서 '자격요건'이나 '우대사항'에서 '2년 이상의 경력' 혹은 '서비스 운영 경험이나 앱을 배포하여 운영해 본 경험'을 선호하는 부분이구요!(배포 자체보다는 운영 경험이 더 중요하다고 생각해요)

개발자이기 때문에 시스템과 플랫폼을 이해하고, 내가 사용하는 언어와 프레임워크의 특징을 잘 고려하여 개발하는 것도 중요하지만, 위에서 말한 부분들도 실무에서는 매우 중요한 영역이기 때문에 과제를 하시면서도 다양한 측면에서 고민하고 적용해 보세요!

현 시점에서는 이제 막 개발학습을 시작하셨기 때문에, 앱이 잘 동작되는 관점에서 리뷰를 드렸고, 이후 평가 과제에서는 점진적으로 효율적인 코드와 설계관점, 유지보수성, 성능 등을 고려하여 리뷰를 드릴 예정입니다.

역량 범위에서 최선을 다해 코드를 제출해 주시면, 더 좋은 방향의 리뷰를 드릴 수 있기 때문에 다음 과제에서는 좀 더 많은 고민과 시간을 투입해서 제출해 주시면 좋을 것 같습니다 😊

마지막으로, 과제를 만드실 때 가급적 과제와 UI디자인을 최대한 동일하게 만들려고 노력해 보세요.

특히 다양한 비율의 기종에서 Autolayout 적용이 잘 될 수 있도록 신경써 보세요.

로직 뿐만 아니라 UI요소들을 의도에 맞게 잘 배치하는 것도 연습이 필요합니다!

(만들어 보신 후 개선하거나 자신만의 개성을 녹이시는건 더 좋습니다🤗)

과제 공통 피드백

여러분, 고생 많으셨어요 :) 그리고 대부분 과제를 잘 구현해주셨습니다!

배운 내용 뿐만 아니라 스스로 추가 학습하신 개념을 활용한 분도 많으셔서 앱 개발을 어떻게 공부해나가면 될 지, 어떻게 하면 효율적으로 코드를 구현할 수 있을 지에 대한 고민도 많이 보였습니다.

이번 평가 과제의 목적은 가독성 좋은 코드, 효율적인 코드, 구조적 설계가 기반이 된 코드 등 현업의 코드리뷰처럼 디테일한 리뷰를 드리는 것은 아니었습니다. 가장 중요하게 확인하고자 했던 부분은 학습 내용을 얼마나 이해하고 있는 지, 배운 개념을 적절한 곳에 잘 적용하고 있는 지를 보고 싶었습니다.

학습 초반에 배우는 개념을 탄탄히 하지 않고 복습을 소홀히 할 경우, 추후 배우게 될 내용을 이해할 때나 협업이 진행이 될 때 스스로 많이 힘들 수 있기 때문에, 디테일한 코드 리뷰는 드리지 않고, 전체적인 관점에서 배운 개념을 이해하고 구현을 한 것인지, 어떤 상황에 어떤 문법을 적용하는 것이 적합한 지를 알고 있는 지 등을 체크하게 되었습니다 :)

그럼에도 50가지의 다양한 코드와 구현에서, 공통적으로 보완하셨으면 좋을 것 같은 지점이나 부족하다고 느꼈던 부분이 있어 몇 가지 케이스를 소개해드리려고 합니다.

혹시 확인해보시면서, 나도 이렇게 구현했을 것 같은데? 나도 이렇게 구현했는데? 라는 부분이 있다면 관련 개념을 추가적으로 학습해보시면 좋을 것 같아요

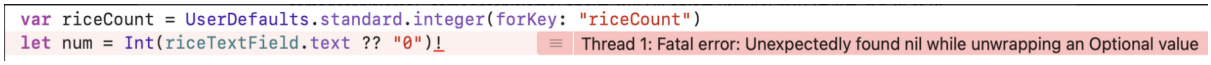
1. 옵셔널 강제 해제와 런타임 에러

- 앱을 최초 실행했을 경우 런타임 에러가 발생하는 경우가 많았습니다. (물론 최초 실행 뿐만 아니라 강제 해제 연산자가 사용된 지점에서 문제가 생긴다면 항상 런타임 오류가 발생하게 됩니다)

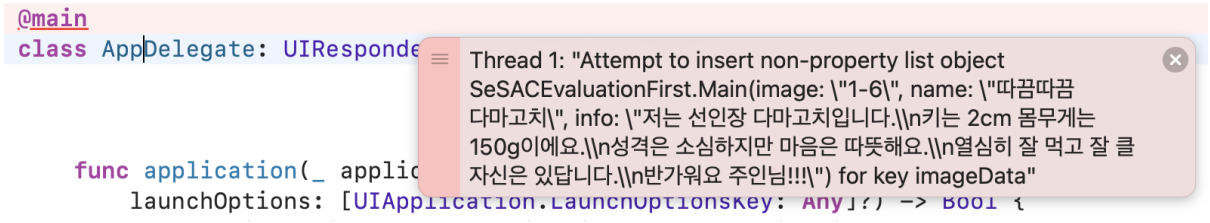
  아마 앱을 계속 빌드하면서 작업을 하셨기에 유저디폴트 키값이 남아 있어 실행이 잘 되고 테스트 시에 문제가 없으셨을 거에요. 하지만 앱 최초 실행 시에는 문제가 발생할 수 있습니다.

  최종 테스트나 배포, Remote Push 전에는 조금 번거로우시더라도 정상적으로 앱이 동작하는 지 클린한 상태에서 테스트 해보시면 좋을 것 같습니다.

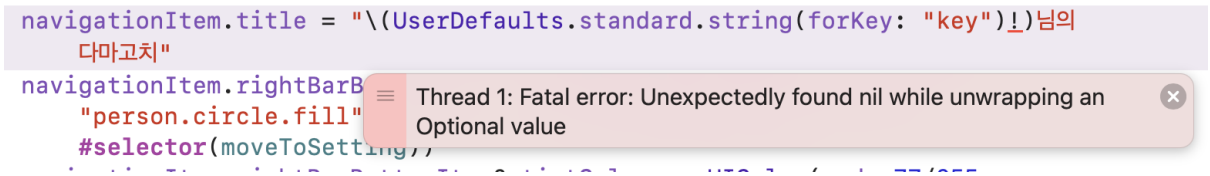
- Case1



- Case2



- Case3



- 옵셔널을 처리하는 코드가 작성되어 있음에도 불구하고, 1가지 케이스만 처리를 하셔서 다른 케이스는 처리가 안된다거나 하는 이슈가 아직 곳곳에서 발생하고 있어요 :)

  혹은 guard, if let 구문을 활용해서 옵셔널 바인딩 처리를 하시고 계시지만, 옵셔널 바인딩 구문 내에 강제 해제 연산자를 함께 사용하는 경우도 있었습니다.

보다 적절한 문법적 형태로 개선을 해보시면 좋을 것 같아요.

또한 String을 Int로, Int를 Double 등으로 형변환 하는 과정에서도 관련 이슈가 많았어요. 값이 없는 상태에서 버튼을 클릭하거나 올바른 값을 넣지 않고 로직을 진행했을 때 역시 잦았습니다.

- 옵셔널이 아직 익숙하시지 않은 부분이기 때문에, 익숙하지 않은 측면을 해결하기 위해서는 어떤 케이스로 문제가 발생할 수 있는 지 한 번 다양한 관점에서 고려를 해보시고, 옵셔널을 해제하기 위한 **옵셔널 바인딩, 타입 캐스팅 구문** 등을 학습해보시면 개선이 될 것 같습니다! :)

2. 뷰컨트롤러의 생명주기

- 간단하게 말씀드리면 부모클래스에 선언된 메서드를 오버라이딩하여 사용할 경우, 특히 viewWillAppear를 포함한 뷰컨트롤러의 생명주기 메서드를 사용할 때, super.viewWillApper 도 같이 작성해주셔야 하세요.

  super.viewWillAppear는 해당 클래스에서 메서드를 실행하기 전에 부모 클래스 메서드를 먼저 수행하도록 합니다. 가끔 쓰고 안써도 상관없는 상황이있지만 불필요한 실수를 피하기 위해서 이 문장을 추가해주시는게 좋습니다.

- Case1

```
override func viewWillAppear(_ animated: Bool) {
    messageLabel.text = showProperMessage(currentLV: defaults.integer(forKey:
        "level"))
    navigationItem.title = "\(defaults.string(forKey: "userName") ?? "대장")님의 다마고치"
}
```

- Case2

```
override func viewWillAppear(_ animated: Bool) {
    randomTalk()
    resetData()
}
```

- Case3

```swift
/// 유저 이름 reload 관련 메서드
override func viewWillAppear(_ animated: Bool) {
    UserDefaults.standard.string(forKey: "userName")
    tableView.reloadData()
}
```

- Case4

```swift
override func viewDidLoad() {
    setLayout()
    self.navigationItem.setHidesBackButton(true, animated: true)
    damagotchiFoodTextField.delegate = self
    damagatchiWaterTextField.delegate = self
    super.viewDidLoad()
}
```

- 세부적으로 말씀드리면, super.method를 꼭 넣어줘야 하는 상황도 있고 선택적인 상황도 존재하는데, 이런 경우 **개발자 문서**의 Discussion을 확인해보시면 도움이 되실 거에요. → 문서 링크

- 그럼에도 특히 뷰컨트롤러의 생명주기 같은 경우, 뷰컨트롤러의 뷰가 메모리에 로드될 때, OS에 의해 호출되는 것이기 때문에 습관적으로 호출을 해주시는 것이 좋습니다.

- 그리고 호출 위치도 중요해요. 만약 super.method를 메서드 가장 하단에 작성하게 된다면, 부모 클래스의 메서드가 호출되기 전에 특정 작업들이 이루어질 수 있기에 경우에 따라서는 의도하지 않은 버그가 발생할 수 있습니다. **뷰컨트롤러의 생명주기**에 대해 조금 더 공부를 하시면 개선이 될 지점일 것 같습니다!

3. 레이아웃 대응

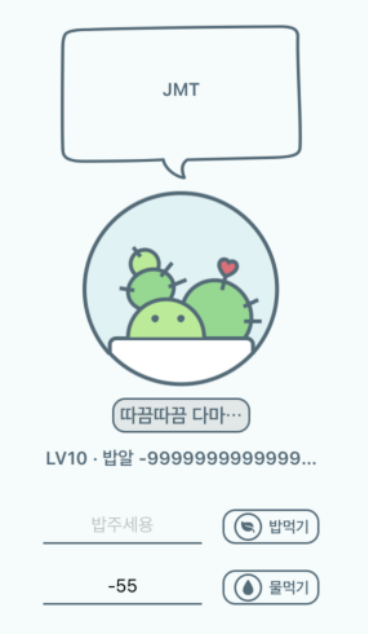
- 다마고치 평가과제 앱에서 크게 해당 사항이 없고 문제가 생길 가능성이 매우 낮다고 하더라도, 다른 앱을 만드신다고 생각하고 레이아웃 관점에서 오류가 생길 수 있는 지점을 말씀드리려고 합니다.

- 디바이스 기준으로 레이아웃을 오류없이 잘 잡아주셨으나, 무한정 긴 문자일 경우 레이아웃이 깨져버립니다. 이를 의도하신 것은 아니시겠죠? 여백의 Constant 설정에서 항상 같다고(=) 설정하지 않고, **최소 여백(≥, ≤)** 을 설정하시면 이를 해결해보실 수 있습니다.

- 디바이스가 작은 경우 또는 뷰 객체의 크기에 비해 과도하게 레이아웃 값을 주었을 경우 의도하지 않은 결과가 나올 수 있어요. 이런 지점 역시 **Preview 기능**을 통해 간편하게 파악을 해보시면 추후 앱 레이아웃 대응 시 어떤 지점을 더 고려해야 하는 지 점점 잘 파악하실 수 있으실거에요!

  보통 레이아웃 문제는 iPhoneSE, iPhone Pro Max, iPhone8 등에서 자주 발생하니 꼭 확인해보는게 좋아요 :)

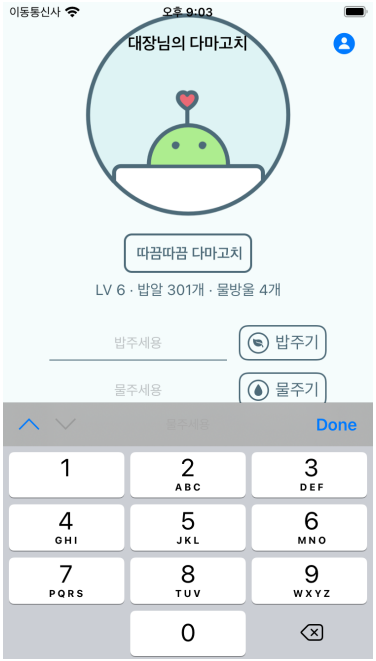
- Case1.



- Case2.

- Case3.

- Case4.
- Case5.
- Case6.



4. 네이밍 컨벤션

- 지금 단계에서는 네이밍 컨벤션이 크게 중요하지 않다고 생각해요. 회사의 스타일 가이드에 따라 조금씩 컨벤션이 달라질 수도 있고, 보통 협업하는 팀원들끼리 컨벤션을 정하고 개발을 시작하기 때문이에요.

  그리고 학습 초기에 네이밍까지 구체적으로 신경을 쓰는 것 보다 큰 개념과 동작 원리를 이해하는 것이 더 중요하다고 판단하고 있기 때문입니다.

- 하지만 보편적으로 사용이 되는 것들이나 일관성이 필요한 부분은 지금부터 지켜주시면 추후에 좋은 습관이 될 거에요.

  메서드나 프로퍼티는 소문자로 시작하고 의미 단위로 대문자를 사용하는 카멜 케이스를 보편적으로 사용합니다. 열거형, 클래스, 구조체를 선언하실 때는 대문자로 시작한다는 것을 알아두셔요 :)

- Case1. 메서드/프로퍼티는 Lower Camel Case를 사용합니다. backgroundColor, detailDesignView 등과 같은 형태입니다.

```
class ColorName{
    static var backgroundcolor = UIColor(red:245/255, green: 252/255,
        blue:252/255,alpha: 1)
    static var fontcolor = UIColor(red:77/255, green: 106/255, blue:120/255, alpha:
        1)
}


extension UIView{
    func DetailCenterDesignView(){
        self.backgroundColor = .systemGray3
    }
    func DetailbackDesignView(){
        self.backgroundColor = ColorName.backgroundcolor
        self.layer.borderWidth = 1
        self.layer.cornerRadius = 4
    }

}
```

- Case2. 클래스/구조체/열거형 등은 대문자로 시작하는 것이 애플이 가이드하는 컨벤션입니다.

```
class growingTamagochiViewController: UIViewController {


enum myDama {
    case first
    case second
}
```

- Case3. 크게 의미가 없는 네이밍은 사용을 지양해주시는 것 역시 추후에 좋은 습관이 되실 거에요!

  훨씬 복잡한 앱을 여러 명이서 협업한다고 가정을 하고, 1년 뒤에 코드를 다시 본다면 축약해두거나 숫자로 기입해둔 네이밍을 과연 쉽게 이해할 수 있을까요?

```
    var level: Int {
        let temp = (riceCount / 5 + waterCount / 2) / 10
        if temp >= 10 {
            return 10
        } else if temp < 10 && temp >= 1 {
            return Int(temp)
        } else {
            return 1
        }
    }
```

```
let a = UIStoryboard(name: "SettingStoryboard", bundle: nil)
let b = a.instantiateViewController(withIdentifier: "SettingTableViewController")
    as! SettingTableViewController
```

- Case4. 타입 어노테이션 주위의 띄워쓰기 역시 컨벤션에 해당하니 참고해주셔요! 그리고 띄워쓰기나 함수 호출 연산자 (), Brace { } 도 한 칸 씩 띄워쓰는 형태가 권장되는 편입니다 :)

```
@objc func savefunc(){

    if nameTextField.text!.count > 7 || nameTextField.text!.count < 2{
```

```
struct Damagochi{
    var image : String?
    let name: String?
    var explain = "준비중이에요"
    var damagochiLevelImage: [String]?

}
```

- 그 외의 다양한 컨벤션을 확인해보고 싶으시다면, **스위프트 스타일 가이드**와 **레퍼런스**가 되는 회사의 스타일 가이드를 참고해보시면 좋으실 것 같아요. 세부적으론 모두 다르지만, 대부분 비슷한 컨벤션을 사용하고 있어 참고하기에 좋습니다.

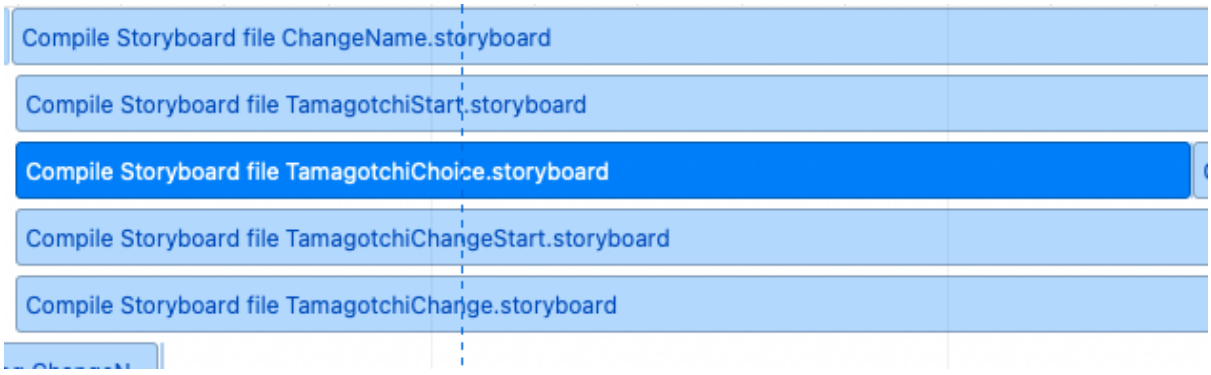
- https://www.swift.org/documentation/api-design-guidelines/

- https://google.github.io/swift/
- https://github.com/raywenderlich/swift-style-guide
- https://github.com/airbnb/swift

5. 필요 없는 코드나 사용하지 않는 라이브러리의 설치

- 사용하지 않는 코드가 작성이 되어 있거나, 사용하지 않는 라이브러리가 설치가 된 경우가 많았어요. 아마 학습하시다 필요 없다고 판단되어 주석 처리 하셨거나, 라이브러리를 삭제하지 않으셨던 거 같아요.
- 지금은 코드가 주석 처리된 부분이나 고민이 되는 지점을 주석에 작성하신 것을 보고, 여러분들의 학습 이해도를 파악 하는 데 도움이 되었어요. 어떤 라이브러리를 사용하려다 포기(?)를 하게 되었는 지, 어떤 이슈로 라이브러리가 아닌 코드로 구현을 하게 되었는 지 등도 판단을 할 수 있었습니다.
- 그럼에도 사용을 하지 않거나 필요가 없는 코드 등은 삭제를 해주시는 습관을 조금씩 연습해보시는 것이 좋습니다.
- 사용하지 않는 뷰컨트롤러나 스토리보드 파일, 씬 등도 마찬가지입니다. 스토리보드는 코드에 비해 비교적 속도가 느리다는 것을 배우셨을 텐데요. 따라서 가능한 가볍게 만드는 것이 좋습니다. 과제 하시면서 수십번을 빌드 하셨을 텐데, 여러분들의 빌드 시간을 야금 야금 갉아먹었을 거에요 :)

  그래서 불필요한 스토리보드 파일은 최대한 삭제하는 것이 좋습니다. 참고로 아래 이미지는 Xcode14 버전부터 확인해볼 수 있는 빌드 타임라인입니다. 추후에 배우게 되실 거에요 .



6. 공통적인 값 처리하기

- 뷰컨트롤러 식별자나 유저디폴트 키값, 반복적으로 사용되는 문자열 또는 숫자값, 인터페이스 빌더 네임 등 키 값을 문자열로 하드코딩 하는 것도 실수 발생 가능성이 높고 민첩한 코드는 아닙니다.

  이런 방법은 실수 가능성도 높고 수정할 때 엄청난 보일러플레이트 발생시킵니다. 문자열 상수를 묶어서 조직화 해볼 수 있지 않을까요?

```
@IBAction func startButtonClicked(_ sender: UIButton) {
    UserDefaults.standard.set(tamagotchiNum, forKey: "tamagotchiNum")
    UserDefaults.standard.set(UserDefaults.standard.integer(forKey:
        "changeTamagotchi"), forKey: "choiceTamagotchi")
    let sb = UIStoryboard(name: "Main", bundle: nil)
    let vc = sb.instantiateViewController(withIdentifier: "MainViewController")
        as! MainViewController
    let nav = UINavigationController(rootViewController: vc)
    nav.modalPresentationStyle = .fullScreen
    self.present(nav, animated: true)
}
```

- 컬러, 폰트 등도 마찬가지입니다. 만약 정책이 바뀌어서 컬러 변경된다면, 그리고 같은 코드를 쓰고 있는 UI가 많다면 어떻게 유지 보수를 하고 수정할 수 있을까요?

  아마 모든 코드를 찾아다니면서 바꿔야 할 거에요. 한 군데 놓치는 지점이 있거나 고치지 말아야 할 곳을 수정한다면 전체 앱을 다시 살펴보는 테스트를 무수히 반복해야 할 거에요.

  지금 코드만 봐도 이게 무슨 색상인지 사실 알 수가 없습니다. 현업에서는 코드를 어떻게 하면 민첩하고 유연하게 짜는지도 굉장히 중요한 요소라, 한번쯤 생각해보시면 좋을 것 같습니다 :)

```
//이름 속성
detailName.font = .systemFont(ofSize: 13, weight: .semibold)
detailName.textAlignment = .center
detailName.layer.cornerRadius = 5
detailName.textColor = UIColor(red: 77/255, green: 106/255, blue: 120/255,
    alpha: 1)
detailName.layer.borderWidth = 0.5
detailName.layer.borderColor = UIColor(red: 77/255, green: 106/255, blue:
    120/255, alpha: 1).cgColor

//설명 속성
detailDescription.font = .systemFont(ofSize: 13, weight: .semibold)
detailDescription.textAlignment = .center
detailDescription.textColor = UIColor(red: 77/255, green: 106/255, blue:
    120/255, alpha: 1)
detailDescription.numberOfLines = 0
```

- 예외 처리를 하시고 계신 “대장" 문자열 역시 공통된 상수로 관리를 해보시는 게 어떨까요? 어딘가에서는 대장님으로, 어딘가에선 대장이라고 사용하고 있다면, 컴파일 타임에 확인하기 어려운 버그가 될 수 있지 않을까요?

```
override func viewDidLoad() {
    super.viewDidLoad()

    userTextField.text = tamaName ?? "대장"
```

```
dummyLabels = ["\(currentName ?? "대장") 님 오늘 잘 잘 자셨나요?",
               "\(currentName ?? "대장") 님 오늘 깃허브 푸쉬 하셨어영?",
               "\(currentName ?? "대장") 님 오늘 과제 하셨어용?",
              "\(currentName ?? "대장") 님 잘 쉬셨나요?"]
```

7. 기타

1) 전역 변수 지양

- Swift 언어에서는 앱 전역에서 쓰이는 전역 함수 또는 전역 변수를 가능한 사용하지 않습니다.

  어디에서나 부를 수 있다는 특징 때문에, 자동완성에 방해가 되고 앱이 커지면 커질수록 불편함도 비례할 거에요.

조금 번거로우시더라도 구조체나 클래스 등의 형식을 통해 해결해보세요.

```
import UIKit

func backgroundViewUI(sender: UIView) {
    sender.backgroundColor = UIColor(red: 245/255, green: 252/255, blue: 252/255,
        alpha: 1)
}

func lineViewUI(sender: UIView) {
    sender.backgroundColor = UIColor(red: 77/255, green: 106/255, blue: 120/255,
        alpha: 1)
}

func lightLineViewUI(sender: UIView) {
    sender.backgroundColor = UIColor(red: 200/255, green: 230/255, blue: 230/255,
        alpha: 1)
}
```

2) 다마고치 레벨 계산

- 다마고치 프로젝트에서 레벨 계산법은 조건이 많아 생각보다 복잡하셨을 거에요. 하지만 꼭 모든 조건에 Switch 구문이나 if 분기 처리가 필요했을까요?

  예를 들어서 30 에서 39 까지라면 10으로 나누면 몇일까요? 40부터 49까지는 10으로 나누면 무조건 4가 나오겠네요!

```
switch levels {
case 0..<20:
    UserDefaults.standard.set(1, forKey: "Level")
    return UserDefaults.standard.integer(forKey: "Level")
case 20..<30:
    UserDefaults.standard.set(2, forKey: "Level")
    return UserDefaults.standard.integer(forKey: "Level")
case 30..<40:
    UserDefaults.standard.set(3, forKey: "Level")
    return UserDefaults.standard.integer(forKey: "Level")
case 40..<50:
    UserDefaults.standard.set(4, forKey: "Level")
    return UserDefaults.standard.integer(forKey: "Level")
case 50..<60:
    UserDefaults.standard.set(5, forKey: "Level")
    return UserDefaults.standard.integer(forKey: "Level")
case 60..<70:
    UserDefaults.standard.set(6, forKey: "Level")
    return UserDefaults.standard.integer(forKey: "Level")
```

3) 인스턴스 생성

- 많은 분들이 다마고치 인스턴스를 생성하셨어요. 구조화를 시도하신 모습은 먼저 잘하셨다고 말씀드리고 싶습니다.

  하지만 class 나 struct 등 보다 더 좋은 형태는 없을까요? 여러 인스턴스를 생성할 필요도 없고 변경 되지 않는 Tamagotchis 라고 한다면 타입 프로퍼티 또는 초기화 구문이 필요 없는 열거형을 활용해보시는 게 어떨까요?

```
class TamagotchiInfo {

    var Tamagotchis: [Tamagotchi] = [
        Tamagotchi(image: "1", name: "따끔따끔 다마고치", introduce: "나는야 따끔따끔 다마고치!
            \n 나를 잘못 건들다간 아주 혼쭐날걸?? 후후... \n 하지만 마음만은 여리니깐 잘 챙겨 달라구~"),
        Tamagotchi(image: "2", name: "방실방실 다마고치", introduce: "헤헤헤~ 나는 방실방실
            다마고치~ \n 하루종일 방실방실해서 붙여진 이름이지~! \n 하하하 호호호 히히히~ 나랑 같이
            놀자~!"),
        Tamagotchi(image: "3", name: "반짝반짝 다마고치", introduce: "반짝반짝 작은별~ \n
            보다도 빛나는 나는 반짝반짝 다마고치~ ⭐ \n 내가 너를 환하게 밝혀줄게~!")
    ]
}
```

- 컬러 등도 마찬가지 일 것 같아요. 꼭 여러 메모리를 사용할 필요가 있을까요?

```
import Foundation
import UIKit

struct ProjectDesign {
    let backgroundColor: UIColor = UIColor(red: 245/255, green: 252/255, blue:
        252/255, alpha: 1)
    let fontColor: UIColor = UIColor(red: 77/255, green: 106/255, blue: 120/255,
        alpha: 1)

}
```

4) 테이블뷰 갱신

- 설정 화면에서 많은 분들이 선택해주셨던 방식입니다. 닉네임 변경을 하고 난 뒤 다시 설정화면으로 돌아왔을 때 셀의 클릭 효과를 해제하시기 위해 viewWillAppear 메서드에 reloadRows를 구현하신 것 같아요.

그런데 viewWillAppear 위치가 적합한 걸까요? 설정 화면이 뜰 때마다 불필요한 갱신이 이루어질 수도 있을 것 같아요.

didSelectRowAt 메서드에서 셀을 클릭한 직후에 바로 선택한 셀에 대한 갱신 코드를 구현해준다면, 필요한 시점에만 호출이 될 수 있지 않을까요?

```
override func viewWillAppear(_ animated: Bool) {
    tableView.reloadData()
}
```

5) 올바른 구조화

- 다마고치의 값을 각각 배열로 만들어주신 분들도 계셨습니다. 다마고치 순서가 변경되거나 삭제, 추가된다면 수정해야 할 부분이 많은 구조체입니다. 이런 경우 하나의 데이터로 개선을 해볼 수 있지 않을까요??

```
struct TamaInfo {

    let tamaName = ["따끔따끔", "방실방실", "반짝반짝"]
    let tamaBaseImage = ["1-7.png", "2-7.png", "3-7.png"]
    let tamaInformation = ["저는 선인장 다마고치 입니다. 키는 2cm몸무게는 150g 이에요. 성격은 소심하지만 마음은 따뜻해요. 열심히 잘 먹고 잘 클 자신은 있답니다. 반가워요 주인님!!!", "저는 방실방실 다마고치입니당. 키는 100km 몸무게는 150톤이에용 성격은 화끈하고 날라다닙니당~! 열심히 잘 먹고 잘 클 자신은 있답니당 방실방실!", "저는 반짝반짝 다마고치 입니다. 키는 170cm 몸무게는 70kg 이에요. 성격은 항상 밝고 반짝여요! 열심히 잘 먹고 잘 클 자신은 있답니다. 저를 키워주세요 주인님!"]

    // 버블 상태메시지
    let tamaStatus = ["안녕하세요","배가 고파요", "응애! 나 아기 다마고치 밥 줘", "공부를 더 열심히 하셔야겠어요!", "오늘 푸쉬하셨나요?", "힘내세요!", "졸리면 일어나서 공부하세요!", "이라고 부르지만 제가 주인이에요^^", "TIL도 작성해주셔야죠!", "큰 일 났어요! 목이 말라요", "오늘도 고생하셨어요:)"]
}
```

- 설정 화면에서도 유사한 부분이 있었습니다. 만약 다마고치 변경하기 셀과 내 이름 설정하기 셀의 위치가 바뀐다면, SettingTableViewController 코드는 얼마나 많은 수정을 해주어야 할까요?

  셀에 대한 정보를 열거형의 rawValue로 관리해주신다면, 셀이 추가되거나 위치가 변경되거나 삭제되더라도 뷰컨트롤러 코드는 건드리지 않고 수정을 해보실 수 있을 거에요.

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
    IndexPath) {
    switch indexPath.row {
    case 0: return moveToChangeNamePage()
    case 1: return moveToSelectTamaPage()
    case 2: return dataInitializeAlet()
    default:return
    }
}
```

개별 피드백

안녕하세요, 나리님! 평가 과제 진행해주시느라 고생 많으셨어요 🙂

사실 제출해주신 프로젝트로는 평가가 어려워요 ㅜㅠ 한 자리에서 10시간 넘게 집중하시면서 무엇인가를 한 게 처음이라고 말씀을 해주셨는데, 완성이라고 보기에는 부족한 점이 많아 리뷰를 드리기가 힘들 것 같습니다.

별도의 시간을 더 많이 투자해주셔서 개선해보신 뒤에 어느 정도 정리가 되셔서 제출을 다시 해주신다면 나중에 코드 피드백을 세세히 드리도록 하겠습니다!!!!

그래서 코드 리뷰보다는 부족한 지점들에 대한 말씀을 조금 더 드리고자 해요.

먼저 시작 화면을 분기 처리 하는 요소가 보이지 않았어요. 아마 항상 앱을 켤 때마다 다마고치를 선택하는 화면이 뜨게 될 것 같아요. SceneDelegate에서 Window RootViewController를 설정해주시는 코드가 추가가 되어야 할 것 같아요. 최초 앱 빌드 시 다마고치를 선택하면 런타임 오류가 발생합니다 :)

그리고 메인 화면에서 동작하지 않는 기능들이 있었습니다. 랜덤으로 말풍선 뜨는 곳에도 더미 데이터가 그대로 노출이 되고, 몇 개 이상 물방울이나 밥알을 먹지 못하는 상황들에 대한 예외 처리, 디바이스 해상도에 따른 대응, 레이블이 … 표시가 되면서 어떤 다마고치인 지 모르는 상태, 레벨에 따라 이미지가 변경이 되지 않는 부분, 준비중인 다마고치를 선택한 경우에도 메인 화면으로 진입 되는 부분 등이 있었습니다.

설정 화면에서는 모든 셀을 선택했을 때 닉네임 변경하는 화면이 뜨게 됩니다. 고래밥 이라고 하는 리터럴 문자열도 뜨고 있어서 이 앱을 실제 사용자가 사용했다고 한다면 사용성이 좋아 보이지 않거나 사용자가 입력하지 않은 문자열이 화면에 보이다 보니 앱을 사용하기 힘들어 질 수 있을 것 같습니다.

UI를 코드로 조절하는 형태와 메서드를 만들어서 기능을 분리하는 것은 전반적으로 적용을 잘 해주신 것 같습니다.

하지만 개발이라는 것은 다양한 경우의 수를 생각하고 통제하는 것에서 더 큰 의미를 가지고 있습니다. 부족했던 지점들은 평가 과제 기간 이후에 새롭게 학습한 개념도 많기 때문에 지속적으로 개선해보실 수 있지 않을까 싶어요 :)

소중한 주말을 코딩과 함께 해주셔서 감사합니다. 다시 한 번 고생많으셨어요!!