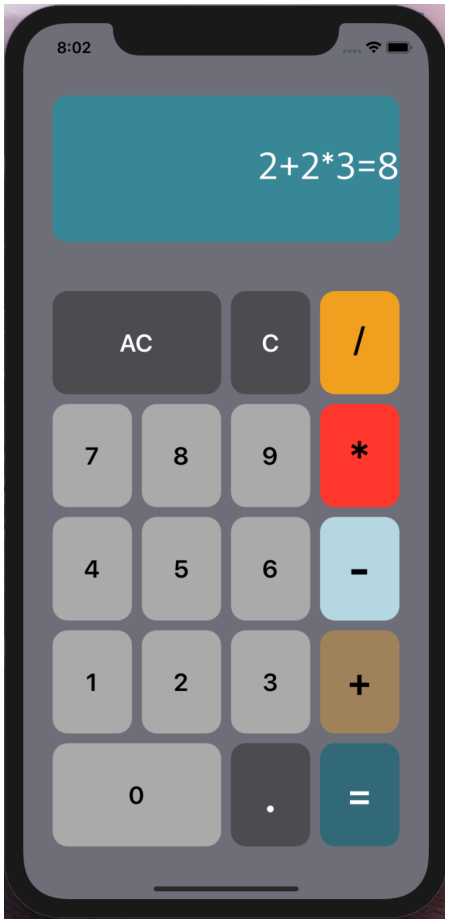
Architecture CountOnMe DA iOS Projet 5

Pour cette application, nous avons réorganisé le code selon le modèle MVC.
Tout ce qui concerne le calcul des opérations ainsi que la gestion de l'expression mathématique sont dans notre fichier Model.

Le ViewController gère les éléments et fonctions d'affichage.

Présentation générale du fonctionnement:



Les nombres et opérateurs saisis par l'utilisateur sont ajoutés dans un tableau de type String défini dans notre Model.

```
var elements: [String]
```

L'expression est récupérée et retraitée par la fonction performCalcul() qui utilise une NSExpression pour produire le résultat.
Le problème que nous avons rencontré est que la NSExpression effectue un calcul sur base de nombres entiers. Par exemple, le résultat de 3/2 donne 1.0 et non pas 1.5.

Pour contourner ce problème, nous convertissons les nombres entiers en décimaux grâce à la fonction makeCurrentNumberDecimal() qui va ajouter « .0 » aux nombres non décimaux.

L'expression est ensuite récupérée par la variable calculée display du ViewController qui enlève les « .0 » pour un affichage plus agréable.

```
private var display: String {
    return calculator.elements.joined().replacingOccurrences(of: ".0", with: "")
    }
```

Fonctionnalités ajoutées:

AC Button: lance la fonction reset() du ViewController qui efface le contenu du tableau éléments et affiche 0 à l'écran.

C Button: efface le dernier élément saisi par l'utilisateur (hors résultat)

(.) Décimal: la fonction associée n'autorise qu'une seule virgule par nombre. Si le bouton est tapé avant un nombre, on suppose que l'utilisateur souhaite un zéro devant.

Multiplication et Division.

ShowsTouchOnHighlight: les boutons s'illuminent légèrement à la pression.

Présentation du Model:

Variables:

En dehors du tableau « éléments » évoqué ci dessus, nous avons crée ces ensembles de Strings qui nous permettent d'identifier opérateurs et chiffres parmi nos éléments.

```
private var operands: Set = ["+", "-", "*", "/"]
private var numbers: Set = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
```

Propriétés calculées:

- **expressionIsCorrect: Bool**
    - Minimum 3 éléments
    - Minimum un opérateur
    - Le dernier élément n'est pas un opérateur
    - L'expression ne contient pas de division par zéro
- **expressionHasOperand: Bool**
- **canAddOperator: Bool**
    - S'il y a déjà un chiffre
    - Si le dernier élément de l'expression n'est pas déjà un opérateur ou une virgule
- **expressionHasResult: Bool**
- **lastElementIsNumber: Bool**
- **lastElementsIsOperand: Bool**

    Ces 2 variables identifient le dernier caractère de l'expression en cours.

- **currentNumber: String**

    Cette variable est calculée en identifiant l'index du dernier opérateur de l'expression, et en enlevant ce qui précède par la méthode removeSubrange()
Elle nous permet de savoir si le nombre en cours a déjà une virgule grâce à la propriété calculée suivante:

- **currentNumberHasDecimal: Bool**

Méthodes:

- **makeCurrentNumberDecimal():**

Comme évoqué au début, nous utilisons cette fonction pour ajouter « .0 » à un chiffre entier afin que la NSExpression produise un résultat décimal.

- **performCalcul():**
    - Calcul du résultat
    - Arrondissement du résultat au centième.
    - Ajout du signe égal puis du résultat à l'expression.

Présentation du ViewController:

Variables:

- **display:**

Cette variable est utilisée pour distinguer ce qui s'affiche de l'expression en elle même.
Ainsi nous utilisons le getter pour récupérer les éléments du Calculator et enlever les .0 des nombres en disposant.
Le setter est utilisé pour définir la propriété text de la textView.

Méthodes:

- **updateDisplay():**

Cette fonction est appelée dans toutes nos @IBActions, à chaque modification de l'expression pour mettre à jour l'affichage.

- **alert():**

Par simplification, nous avons crée une alerte unique qui se déclenche si l'utilisateur tente de taper une expression incorrecte.

- **reset():**

Efface l'ensemble des éléments du tableau par la méthode removeAll() et affiche 0 à l'écran.
Cette fonction est apellé en pressant le bouton reset (AC)

- **tappedCorrectionButton():**

Efface le dernier élément du tableau sauf si l'expression affiche un résultat.

- **tappedNumberButton():**

    - Si un résultat est affiché, on démarre une nouvelle expression
    - Sinon on vérifie que le nombre en cours ne commence pas par zéro
    - Puis on ajoute le nombre à l'expression

- **tappedDecimalButton():**

    - Si tapé en début d'expression, on y ajoute un zéro devant
    - Sinon on vérifie que le currentNumber n'a pas déjà une virgule
    - Puis on ajoute la virgule au currentNumber

- **tappedOperandButton():**

    - Met le currentNumber décimal si nécessaire
    - Si un résultat est affiché, celui ci devint le 1er élément de la nouvelle expression
    - Sinon, ajoute l'opérateur à l'expression si canAddOperator == True

- **tappedEqualButton():**

    - Met le currentNumber décimal si nécessaire
    - Si un résultat est déjà affiché, rien ne se passe
    - Si l'expression est correcte, on lance la fonction performCalcul()

Test Unitaires:

Nous avons effectué des tests unitaires sur les variables et fonctions du Model.
Nos tests couvrent 95% du code.

Le ViewController et ses éléments d'interface n'ont pas été testés.

GitHub:

Lien vers le dépôt distant du projet :

https://github.com/Hquart/P5OCR-CountOnMe.git