

RAPPORT DE PROJET - EP 24 MASTER CIM VCIEL

DEVELOPPEMENT D'APPLICATIONS MOBILES IOS

MASTERMIND FRUITY

Projet réalisé par : Aymane DANIEL



Enseignement : Serge Miguet

REMERCIEMENTS

Je tiens à remercier ma fille de cinq ans, Serena DANIEL qui m'a donnée l'idée de faire faire le jeu MasterMind Fruity. C'est en regardant ces dessins que j'ai décidé de faire le Mastermind en utilisant des fruits au lieu des pions classique.



Table des matières

REMERCIEMENTS	2
INTRODUCTION.....	4
<i>1- Objectif technique.....</i>	<i>4</i>
<i>2- Constraint du projet.....</i>	<i>4</i>
<i>3- Prérequis technique</i>	<i>4</i>
LE DESIGN DE L'APPLICATION	5
<i>Les ressources graphiques et sonores</i>	<i>5</i>
Les images	5
Font et Police d'écriture.....	6
L'élément sonores	6
<i>Les écrans de l'application</i>	<i>6</i>
ARCHITECTURE DE L'APPLICATION	8
<i>1-Organisation du code.....</i>	<i>8</i>
Le Modele (Model).....	8
La Vue (View).....	8
La Vue-Model (ViewModel)	8
<i>2-Le flux de donnée (DataFlow)</i>	<i>9</i>
<i>3-Diagramme de class.....</i>	<i>10</i>
ALGORITHME ET REALISATION DU MASTERMIND	11
<i>Présentation des pions : les fruits et le panier.....</i>	<i>11</i>
<i>Déroulement d'une Partie de MasterMind Fruity</i>	<i>12</i>
<i>Explication de l'algorithme du jeu.....</i>	<i>13</i>
A- La méthode checkValueEnteredByUser () :	13
B- La méthode generateSecret() :	14
C- La méthode isDuplicate() :	15
D- La méthode scoreManger() :	15
CONCLUSION ET BILAN	16
<i>Bilan personnel.....</i>	<i>16</i>
<i>Analyse critique.....</i>	<i>16</i>
REFERENCE.....	16
<i>Ressources d'apprentissages :.....</i>	<i>16</i>
<i>Liens des resources graphiques :.....</i>	<i>16</i>

INTRODUCTION

Depuis l'apparition des premiers smartphones de nombre applications ont vu le jour. Les applications mobiles deviennent importantes et incontournables dans le quotidien des humains. La nécessité pour les entreprises de se positionner sur ce marché s'est alors agrandit.

Dans le cadre du Master 2 VCIEL, nous devons réaliser une application mobile pour iPhone. Cette application est une mise en pratique des connaissances acquises dans le cadre du module « EP24 - Développement d'Applications Mobiles ». Ainsi l'application choisie est le Mastermind Fruity, une version du Mastermind avec des fruits à la place des pions classique.

1- Objectif technique

L'objectif du projet est de développer une application iOS pour iPhone. L'application sera développée avec le langage Swift. Ce dernier est un langage de programmation puissant, fiable mis au point par Apple.

J'ai choisi de réaliser l'application en utilisant les nouveaux composants SwiftUI à la place du StoryBoard. SwiftUI n'étant pas au programme de l'EP24, je m'engage toutefois à respecter les contraintes définies pour la validation du module.

Plusieurs algorithmes peuvent être mis en place pour gérer la complexité du jeu, l'objectif pour cette version est d'implémenter le jeu avec le niveau de difficulté facile du MasterMind. Toutefois si le temps nous le permet, on pourra mettre en place les algorithmes pour gérer les autres complexités : niveau de difficulté moyen et difficile.

2- Constraint du projet

La première constraint est le temps dont nous disposons pour l'apprentissage et la réalisation du projet.

L'application devra intégrer au moins deux écrans différents et une navigation entre ces écrans. Au moins l'une des vues devra être associée à un contrôleur, dans lequel on doit gérer une partie interactive, par programmation en Swift.

3- Prérequis technique

Afin de réaliser le développement avec SwiftUI, il est nécessaire de disposer d'un MacOs version Mojave + ou Catalina et de Xcode 11. Il est préférable d'utiliser la version 12 de Xcode.

Important à savoir le live preview sur l'IDE Xcode 11 n'est disponible que sur MacOs Catalina.

LE DESIGN DE L'APPLICATION

Les ressources graphiques et sonores

Les images

Xcode nous permet d'intégrer des images dans le projet, pour cela j'ai récupéré des images sur le site <https://fr.pngtree.com/>. J'ai ensuite découpé les images avec Photoshop et utiliser adobe XD pour générer les ressources à partir d'un Template que j'ai créé. J'ai scanner et intégrer aussi les images de ma fille. Pour gérer les différent taille d'écran Apple a mis en place un système qui se base sur trois tailles d'image.



Figure 1 -Image-fruity-Serena

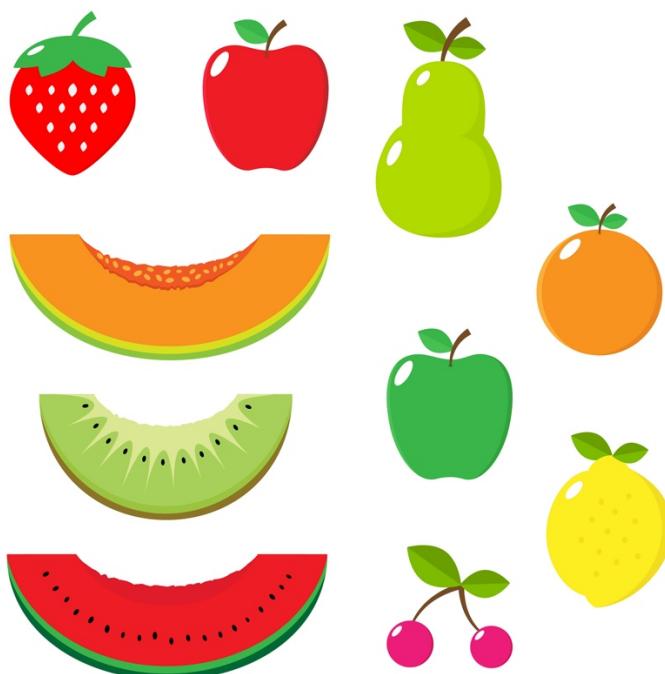


Figure 2 - source : <https://fr.pngtree.com/>

Font et Police d'écriture

Le projet utilise la font **Juicy Fruity** télécharger depuis le site <https://www.dafont.com>.
L'insertion de celui-ci se fait via le fichier « info.plist ».

Key	Type	Value
Localization native development region	String	\$(DEVELOPMENT_LANGUAGE)
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	\$(PRODUCT_BUNDLE_PACKAGE_TYPE)
Bundle version string (short)	String	1.0
Bundle version	String	1
Application requires iPhone environment	Boolean	YES
Application Scene Manifest	Dictionary	(1 item)
Application supports indirect input events	Boolean	YES
Launch Screen	Dictionary	(2 items)
Background color	String	
Image Name	String	fraise-1
Required device capabilities	Array	(1 item)
Supported interface orientations	Array	(3 items)
Supported interface orientations (iPad)	Array	(4 items)
Fonts provided by application	Array	(1 item)
Item 0	String	Juicy Fruity.otf

L'élément sonores

Deux ressources sonores sont ajoutées dans le projet. Il s'agit des fichiers des fichiers mp3 :

- 1- *Ta Da-SoundBible.com-1884170640.mp3*
- 2- *TunePocket-Access-Denied-Error-Buzz-Preview.mp3*

J'utilise aussi le Service **AudioServices** pour lire les son système de l'iPhone disponible à l'adresse suivante : <http://iphonedevwiki.net/index.php/AudioServices> . Pour utiliser ses le service **AudioServices** il faut importer les libraires : **AVFoundation** et **AudioToolbox**.

Les écrans de l'application

L'application est composée de deux écrans. Le premier est l'écran d'accueil « **HomeView** » et le deuxième écran est celle du lancement et l'exécution du jeu « **GameView** ». Le troisième écran et celle de la fin de partie « **GameOverView** » .



Figure 3- L'écran d'accueil : HomeView

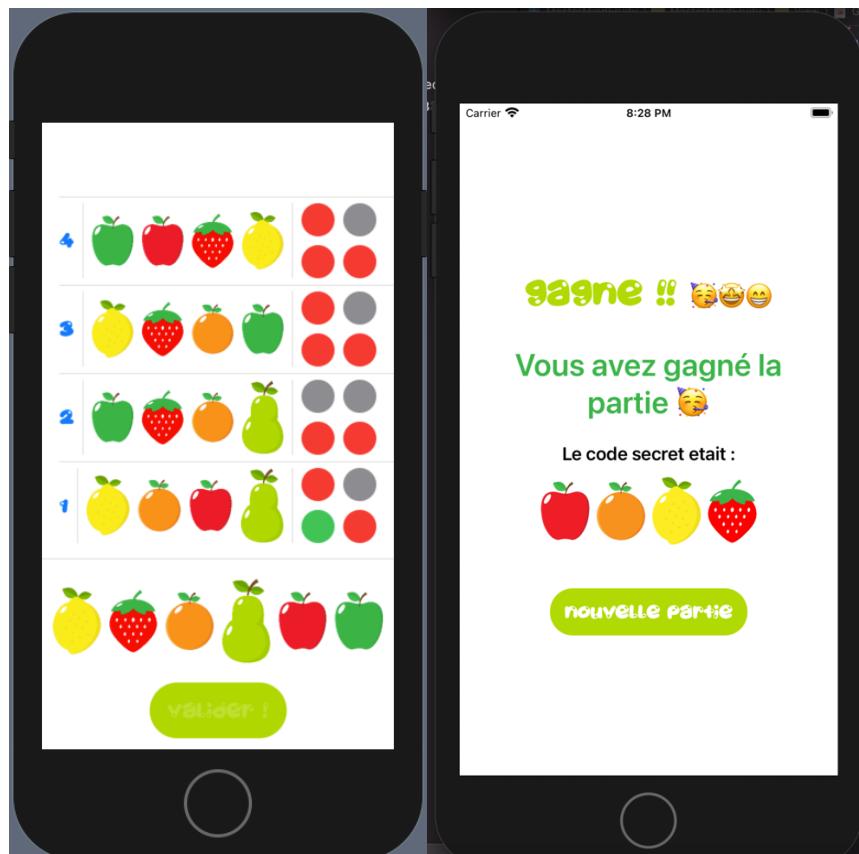


Figure 4 - L'écran du jeu : GameView

L'écran de fin de partie : GameOverView

ARCHITECTURE DE L'APPLICATION

Après avoir écrit plusieurs lignes de code dans un seul fichier **ContentView**, on se rends compte de l'importance de séparer la logique et la présentation, les vues et le model. Pour ce projet j'ai donc choisi d'implémenter l'architecture MVVM (Model View ViewModel).

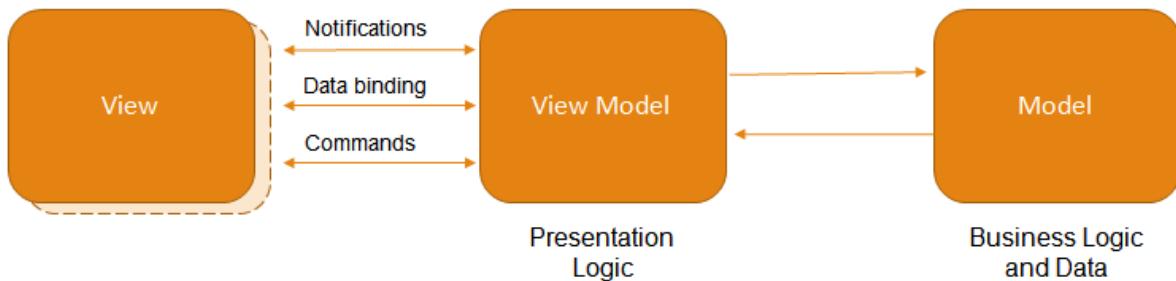


Figure 5 - Source - <https://docs.devexpress.com/WPF/15112/mvvm-framework>

1-Organisation du code

Pour répondre au besoin défini, j'ai choisi de structurer le code dans des dossiers (Group dans l'IDE Xcode) : Models, View, ViewModel. Chacun des groupes contiendra des classes du projet.

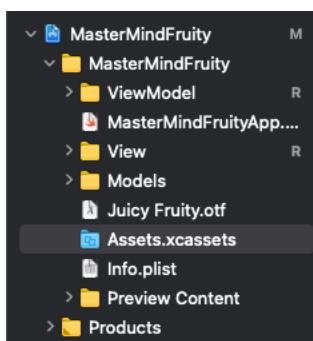


Figure 6 – structure des dossiers

Le Modele (Model)

Le model représente la structure des données et n'est pas lié à aucune vue.

La Vue (View)

Elle contient la définition structurelle de ce que les utilisateurs auront à l'écran. On peut y mettre du contenu statique et dynamique (animations et les états de changements). Elle ne doit contenir aucune logique applicative.

La Vue-Model (ViewModel)

Ce composant fait le lien entre le modèle et la vue. Il s'occupe de gérer les liaisons de données et les éventuelles conversions. C'est ici qu'intervient le binding.

2-Le flux de donnée (DataFlow)

Le flux de donnée nous permet de faire la communication entre le Model et la VueModel et du VueModel vers la Vue. Comme nous pouvons le constater sur la **figure 5** le model ne communique jamais avec la vue directement.

Pour mettre en place la communication entre les différents composants Swift et SwiftUI nous propose plusieurs outils. Ces derniers font partie intégrante du Framework. Ici nous n'allons pas détailler le fonctionnement mais juste les présentés. Le site d'Apple fournit beaucoup d'information. Ces outils sont :

Les outils de communication basiques :

- a- Propriété = Lecture de la donnée
- b- @State = Lecture et écriture
- c- @Binding = Lire et modifier une propriété hors de sa vue.
- d- @Environment

Les outils avancés : Les Objets

- a- ObservableObject,
- b- @Published,
- c- @ObservedObject
- d- @EnvironmentObject

3-Diagramme de class

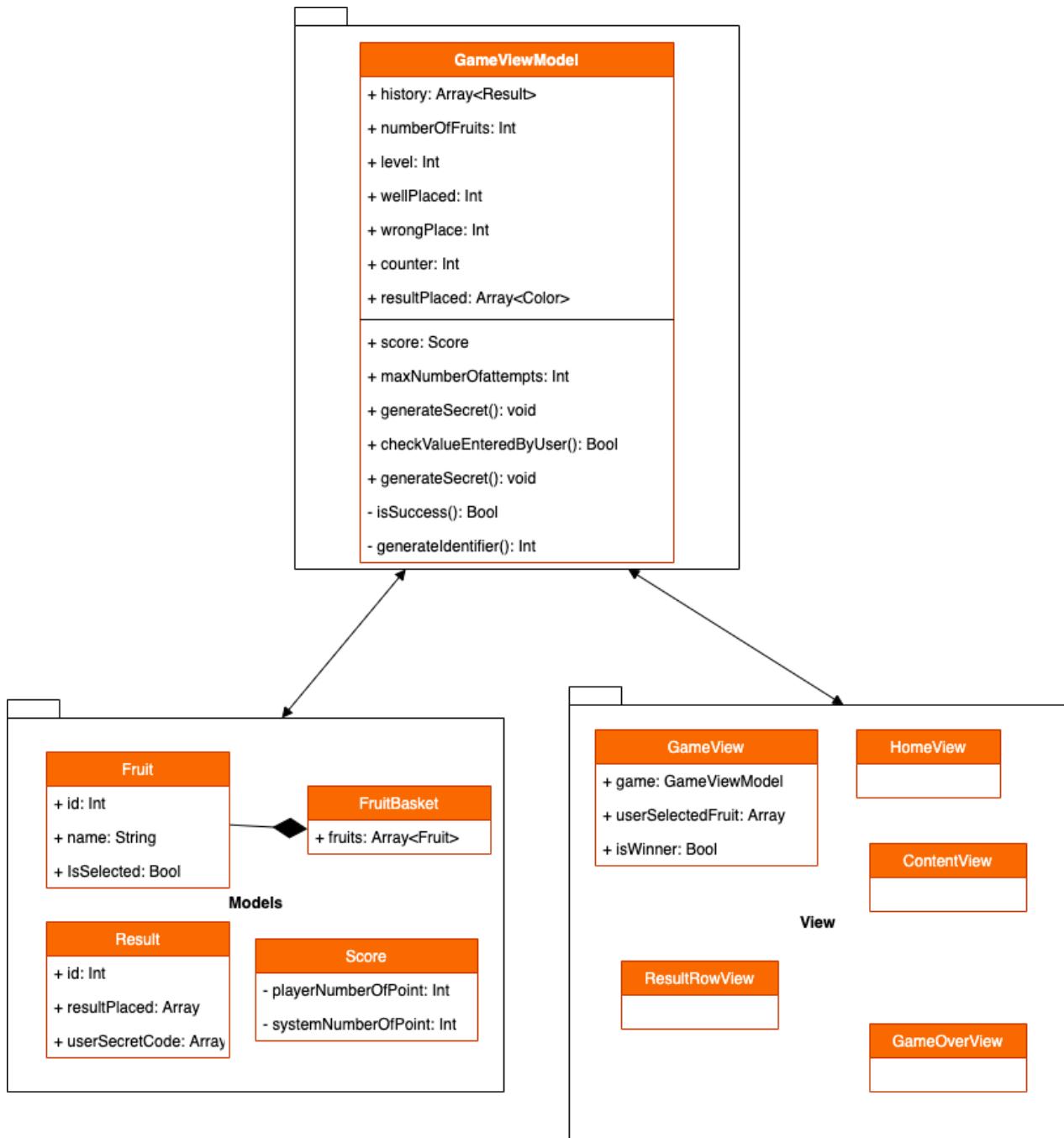


Figure 7 - Diagramme de class

ALGORITHME ET REALISATION DU MASTERMIND

Le Mastermind est jeux de logique dont le but est pour l'un des joueurs d'élaborer une combinaison un code de quatre couleurs et pour son adversaire, de deviner en un minimum de coup cette combinaison. Le nombre maximum de coup est de 15 pour le mode facile et de 12 pour les modes medium et difficile pour cette implémentation.

Présentation des pions : les fruits et le panier

Dans cette implémentation du Mastermind, nous devons devenir une combinaison de quatre fruits distincts (en mode facile) parmi les six fruits suivants :



1. Citron jaune
2. Fraise
3. Orange
4. Poire
5. Pomme rouge
6. Pomme verte

L'ordre de saisie des fruits dans le jeu est important.

L'ensemble des six fruits représente le panier. Le code la classe « **FruitBasket** » implémente le panier de fruits.

```
class FruitBasket: ObservableObject{
    @Published var fruits: [Fruit]

    init(){
        self.fruits = [
            Fruit(id: 1, name: "citron-jaune", isSelected: false ),
            Fruit(id: 2, name: "fraise", isSelected: false),
            Fruit(id: 3, name: "orange", isSelected: false),
            Fruit(id: 4, name: "poire", isSelected: false),
            Fruit(id: 5, name: "pomme-rouge", isSelected: false),
            Fruit(id: 6, name: "pomme-vert", isSelected: false),
        ]
    }
}

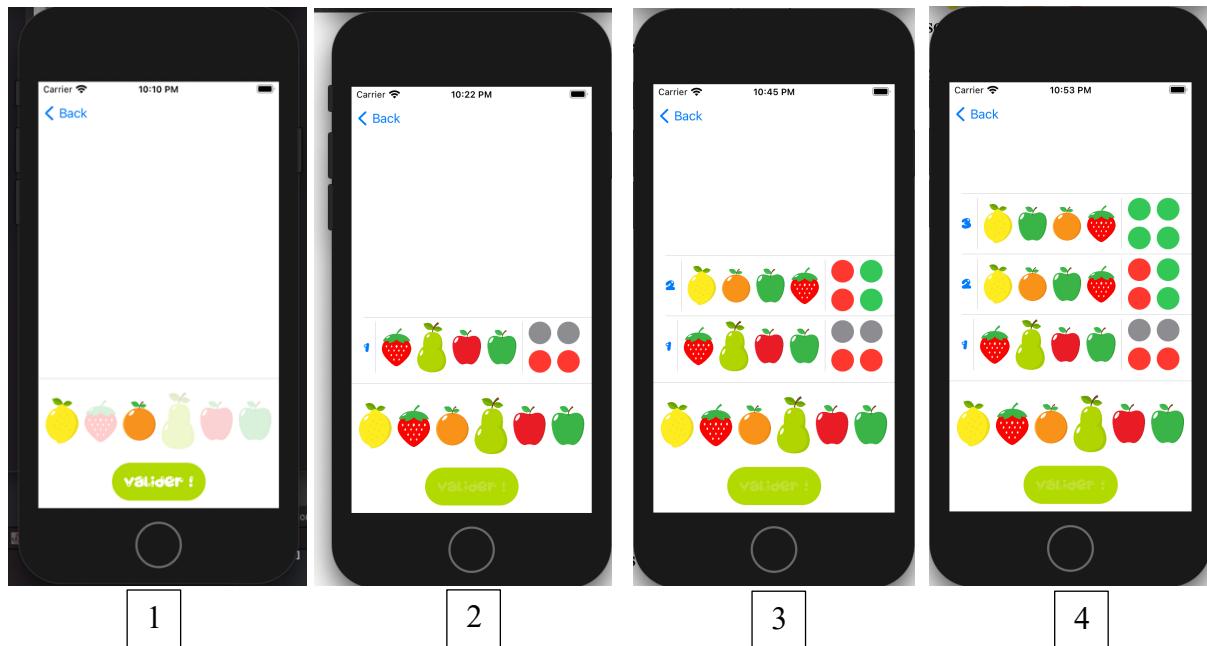
class Fruit: Identifiable ,ObservableObject{
    var id : Int
    var name: String
    @Published var isSelected: Bool

    init(id: Int ,name: String,isSelected: Bool){
        self.id = id
        self.name = name
        self.isSelected = isSelected
    }
}
```

Déroulement d'une Partie de MasterMind Fruity

Lorsque le jeu est lancé, un code de quatre chiffres (id de fruit de 1 à 6) distincts (pour la complexité facile) est généré par le système. L'ordre des chiffres est important ce qui implique que l'ordre de saisie des fruits l'est aussi.

L'utilisateur tape une combinaison fruits, celles-ci sont désactivés et une opacité à 0.5 est appliquée.



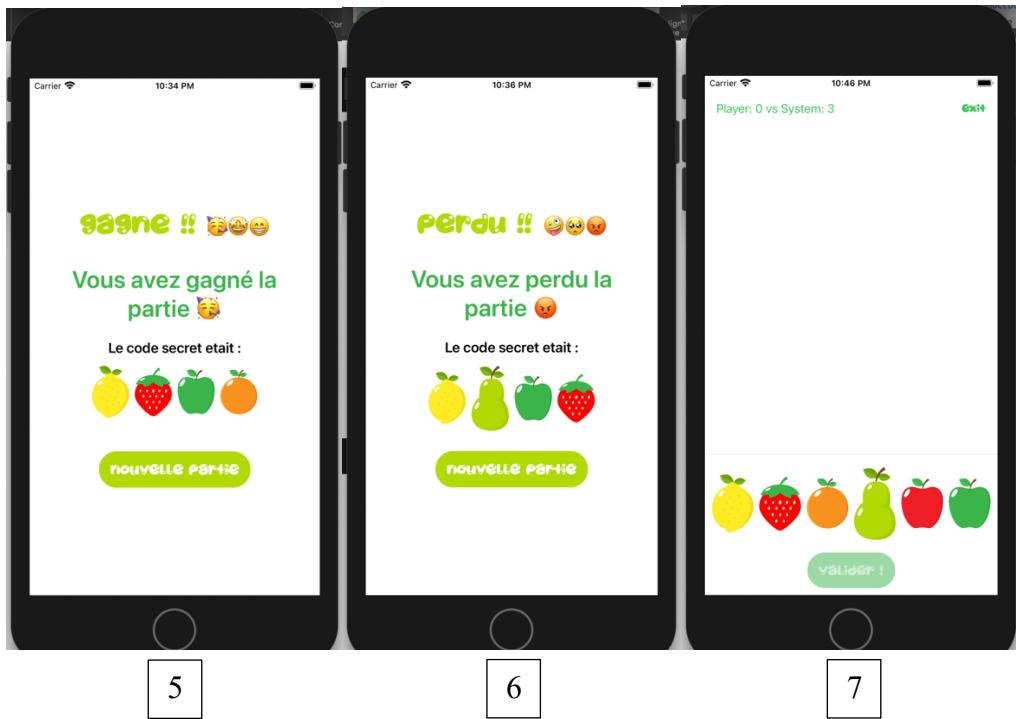
Dans notre exemple le code à deviner est [1, 6, 3, 2] qui correspond à :

Nous pouvons constater sur l'image 2 que le code saisi contient deux fruits mal placés et deux fruits qui ne font pas parti du code secret.

Les fruits mal placés sont représentés par les cercles rouge et ceux qui ne sont pas dans le code pas les cercles gris à droite de l'écran.

Sur l'image 3 le code saisi est partiellement correct avec deux fruit bien placé représenté par les cercles verts et deux fruits mal placés représentés par les cercles rouges.

Sur l'image 4 le code est déchiffré en trois coups et le joueur gagne la manche.



Lorsqu'un joueur gagne ou perd la partie, la vue « **GameOverView** » est affichée voir image 5 et 6. Cette vue indique au joueur s'il a gagné ou perdu et affiche le code secret. Sur cette vue le bouton « **nouvelle partie** » apparaît lorsqu'on clique dessus une nouvelle partie démarre et le score du joueur ou du system est incrémenté, voir image 7. Sur l'image 7 en haut à droite le bouton « **exit** » permet de revenir sur l'écran d'accueil.

Explication de l'algorithme du jeu

A- La méthode `checkValueEnteredByUser()` :

L'algorithme principale du jeu est décrit par la class « **Game** » dans le fichier *Game.swift*. Dans cette class se trouve la méthode « `checkValueEnteredByUser()` » qui permet de vérifier le code saisi par le joueur. Avant de lancer cette méthode il faut démarrer le jeu en exécutant la méthode « `start()` ». Cette dernière permet d'initialiser les variables et exécute la méthode « `generateSecret()` ». La méthode « `generateSecret()` » prend en paramètre la complexité (*easy, medium, hard*) et rempli la variable « `secretCode` » qui est un tableau de longueur « `secretCodeLength` ». Cette longueur est définie en fonction de la complexité du jeu, ça valeur est soit 4 ou 6. Six c'est le nombre fruit total définit par la variable « `numberOfFruits` ».

Lorsque le joueur fait une proposition de code, celle-ci est évalué par la méthode `checkValueEnteredByUser`. Lors de cette évaluation certaines variables sont alimenté comme « `wellPlaced` », « `wrongPlaced` », « `history` », « `counter` » et « `resultPlaced` »

L'algorithme en langage Swift : checkValueEnteredByUser

```
/* la methode permet de verifier le code saisi */
public func checkValueEnteredByUser(userValue: [Int]) -> Bool{
    self.userSecretCode=userValue
    counter+=1
    resultPlaced.removeAll()
    var secret = self.secretCode
    self.wellPlaced=0
    self.wrongPlaced=0

    //recherche des pions bien placé
    for i in 0 ..< secretCodeLength
    {
        if(secret[i] == userValue[i] ){
            self.wellPlaced+=1
            resultPlaced.append(Color.green)
            secret[i] = -1
        }
    }

    //recherche des pions mal placé
    for i in 0 ..< secretCodeLength
    {
        if(secret.contains(userValue[i]) ){
            self.wrongPlaced+=1
            resultPlaced.append(Color.red)
            if let index = secret.firstIndex(of: userValue[i]) {
                secret[index] = -1
            }
        }
    }

    resultPlaced.shuffle()
    secret.removeAll()
    history.append(Result(id: counter,resultPlaced: resultPlaced, userSecretCode: userValue))
    scoreManger()
    return isGameOver
}
```

B- La méthode generateSecret() :

Cette méthode comme son nom l'indique elle permet de générer un nouveau code. Pour cela elle fait appelle à la méthode « **generateIdentifier()** ». Cette dernière génère un identifiant compris entre 1 et 6.

L'algorithme en langage Swift : generateSecret

```
/* la methode permet de générer un code */
private func generateSecret(complexite: Level = .easy){
    secretCode.removeAll()
    switch complexite {
    case .easy:
        while(secretCode.count<secretCodeLength){
            let digit = generateIdentifier();
            if !secretCode.contains(digit){
                self.secretCode.append(digit)
            }
        }
        break
    default:
        for _ in 1 ... secretCodeLength {
            self.secretCode.append(generateIdentifier())
        }
        break
    }

    print("Nouveau code: \(secretCode)")
}
```

C- La méthode isDuplicate() :

Cette méthode permet de vérifier si la proposition de combinaison de fruit existe ou pas dans l'historique. Elle se traduit dans le jeu par une alerte qui se déclenche lorsque la combinaison testée va être en double.



Figure 8 – Alerte doublon

L'algorithme en langage Swift : isDuplicate

```
/* Detection des doublons dans les propositions de code du joueur*/
public func isDuplicate(userValue: [Int])->Bool{
    for result in history {
        if(result.userSecretCode.elementsEqual(userValue)){
            return true
        }
    }
    return false
}
```

D- La méthode scoreManger() :

Cette méthode gère le score de la partie. Une partie gagnée permet de cumuler 3 points. Elle utilise la struct score et déclenche les méthodes d'incrémentation du score.

L'algorithme en langage Swift : scoreManger

```
/* Gestion du score */
public func scoreManger(){
    if(isGameOver){
        if(isSuccess){
            score.incrementScorePlayer()
        }else {
            score.incrementScoreSystem()
        }
    }
}
```

CONCLUSION ET BILAN

Bilan personnel

Après avoir suivi les premiers cours du module EP24 – Développement d'applications mobiles iOS, je me suis lancé dans un apprentissage intensif pour acquérir les bases de la programmation Swift. Durant cet apprentissage du langage **Swift** et du **Storyboard**, je me suis rendu compte de la difficulté à réaliser des choses simples comme des listes ou des tableaux. Pour créer une simple liste scrollable il faut avoir acquis des connaissances avancées en programmation orienté objet (les protocoles etc..), créer un Controller qui va gérer la liste et qui répond aux exigences du protocole. Tout un chapitre entier rien que pour faire une simple liste d'élément.

C'est sur ce constat que j'ai donc décidé de m'orienter vers la nouvelle façon de travailler avec les interfaces graphiques en utilisant SwiftUI et j'en suis très satisfait.

Analyse critique

Dans l'ensemble l'application fonctionne bien sans bug apparent. Toutefois il reste quelques points d'amélioration :

- 1- La rotation de l'appareil en mode paysage (Je pense à le désactiver)
- 2- La gestion de la complexité, J'aurai aimé réaliser les deux complexités qui reste medium et hard.
- 3- La persistance du score dans une base de données.

Personnellement il me reste beaucoup de chose à apprendre du langage Swift et SwiftUI et je pense continuer à l'utiliser pour mes prochaines applications.

REFERENCE

L'application est disponible sur github : <https://github.com/maxson007/firstAppSwiftUI>

Ressources d'apprentissages :

- 1- Cours Université Lyon 2 : <https://moodle.univ-lyon2.fr/course/view.php?id=1194>
- 2- Introducing SwiftUI: <https://developer.apple.com/tutorials/swiftui/>
- 3- Documentation Swift : <https://swift.org/documentation/>
- 4- Learn Swift 5.2 for free: <https://www.hackingwithswift.com/>
- 5- OpenClassroom : Cours sur la programmation Swift : <https://openclassrooms.com/>
- 6- Stack Overflow: <https://stackoverflow.com/>
- 7- <https://forums.swift.org/>

Liens des resources graphiques :

- 1- Icône de l'application : <https://appicon.co/>
- 2- Font : <https://www.dafont.com/fr/juicy-fruity.font?text=fruit>
- 3- Images : <https://fr.pngtree.com/>

