

## Module LPS: Language de Programmation Spécialisés (*Swift*)

### Travaux Pratiques 4

Principe d'application basée sur *SwiftUI*  
(illustrations basées sur la version *Xcode* 15.2)

---

**Objectif:** Cette feuille de TP a pour objet de donner un aperçu de *SwiftUI*. Il s'agira dans un premier temps, de montrer les principes d'affichage et d'interaction avant de réaliser l'implémentation du jeu du nombre mystère en utilisant. En nous verrons comment à partir d'une source de code décliner l'application allant de IOS, MacOS ou Vision OS

---

#### Partie I: Premier pas avec *SwiftUI*

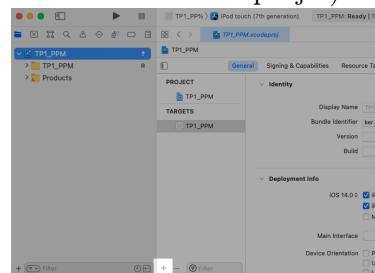
*SwiftUI* a été introduit afin de répondre aux limitations des modèles classiques du type *Model-View-Controller* (que nous avons vus dans les TP précédent) et pour proposer une nouvelle façon moderne de concevoir une interface dynamique et adaptable sur différents supports. Dans cette partie, nous allons explorer les bases pour concevoir la vue principale d'une interface simple.

1.1 Pour réaliser ce TP, vous pouvez soit commencer un nouveau projet (voir [A]) soit rajouter une nouvelle cible dans votre projet actuel (voir [B]).

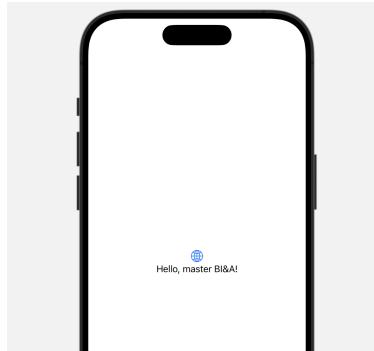
- [A] Pour créer un nouveau projet basé *SwiftUI*, sélectionner un type d'application *IOS*, avec un style d'interface de type "*SwiftUI*" et un type de cycle de vie (*life cycle* aussi associé à *SwiftUI App*).



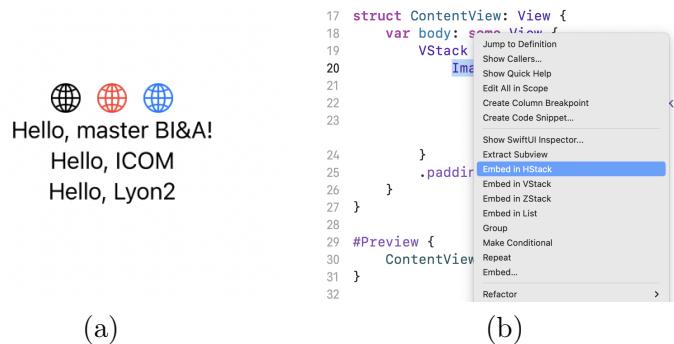
- [B] L'autre possibilité consiste à rajouter une cible dans votre projet initial. Ce rajout s'effectue en sélectionnant la racine de votre projet puis en rajoutant une cible (image (a)) ou à partir du menu Editor (attention ce menu est contextuel et l'ajout d'une nouvelle cible n'apparaît qu'après avoir sélectionné la racine du projet).



1.2 La vue principale de votre application est représentée par défaut dans la classe **ContentView**. Ouvrez cette classe, et modifiez le code de façon à faire apparaître le message "Hello master BI&A". En dessous de votre éditeur vous devez pouvoir voir un rendu de votre application quoi s'afficher en direct.



1.3 Pour comprendre le fonctionnement, de la conception d'une vue, vous pouvez commencer à éditer la vue à partir de la vue du code et en modifiant le code de façon à obtenir le résultat de l'image (a) ci-dessous. **Indication** : ces images ont été obtenues en utilisant des HStack et VStack (accessible à partir d'un simple clic souris droit sur le composant cible (voir image (b) ci-dessous)).



(a)

(b)

1.4 Recopier la base du code ci-dessous et vérifiez que vous obtenez bien le résultat de l'image (b).

```

12 struct ContentView: View {
13     var body: some View {
14         VStack() {
15             VStack {
16                 Text("Hello, master BI&A!")
17                 Text("Bertrand Kerautret")
18             }
19             VStack(alignment: .center) {
20                 Text("Jeu n° Mystère
21                     again!")
22             }
23             .background(Color(
24                 .init(srgbRed: 0.9, green:
25                     0.9, blue: 0.9, alpha: 1)))
26         }
27     }

```

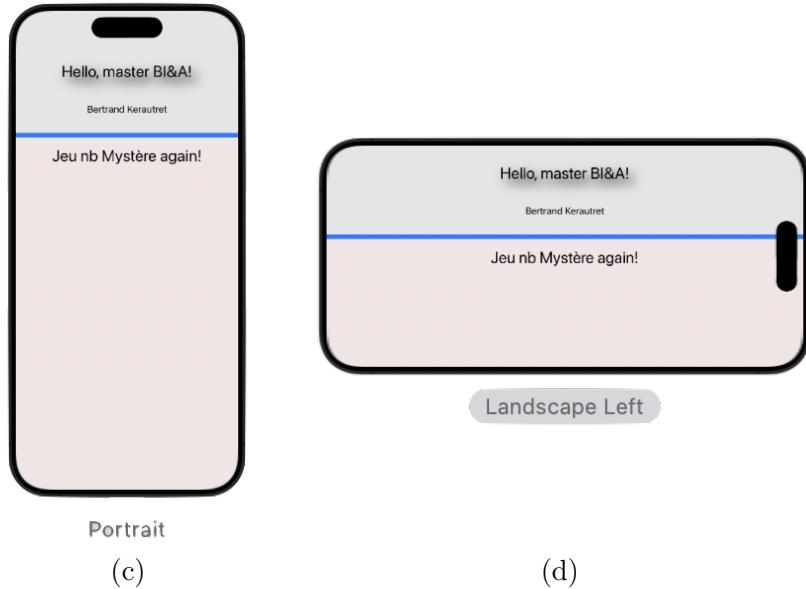
(a)



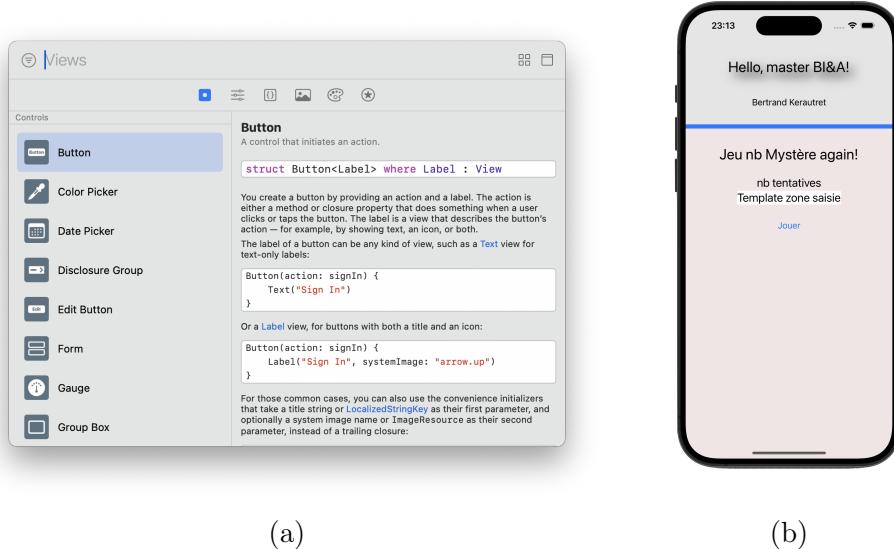
(b)

1.5 En utilisant les attributs ci-dessous, tentez d'obtenir les visualisations des images (c) et (d).

- **padding()** : permet de rajouter des marges sur l'élément considéré.
- **frame(minWidth: , maxWidth: , minHeight: , maxHeight: , alignment: )** : spécifie le cadre de l'objet et peut s'utiliser notamment en utilisant l'attribut **infinity** pour avoir une extension maximale ;
- **shadow(color: , radius: , x: , y:)** : permet de rajouter de l'ombre à un élément.



1.6 En utilisant le bouton + (image (a)) ci-dessous, rajoutez un **Button** et d'autres éléments de façon à obtenir une interface ressemblant à la suivante :



## Partie II: Interaction avec *SwiftUI*

Dans cette partie, il s'agit d'explorer les éléments d'interaction définis à travers la liaison de variables et l'utilisation de boutons.

2.1 Dans la structure **ContentView**, rajoutez une variable **nbJoue** de type **String** qui servira à stocker le nombre saisi par l'utilisateur (initialisée à une chaîne vide). Afin que la variable puisse être modifiée automatiquement, il est nécessaire de rajouter **@State** devant le nom de variable.

2.2 Rajoutez un **TextField** à la place du texte représentant la zone de saisie. Associez la variable **nbJoue** dans le deuxième champ du, en ajoutant le symbole \$ devant le nom de la variable (sans espace).

2.3 Pour vérifiez que la variable est bien mise à jour en temps réel en la faisant afficher dans la champ *indication*. Vous pourrez avoir une visualisation comparable à l'image suivante :



2.4 Rajoutez une action dans le bouton que vous avez rajouté afin de tester le bon fonctionnement de ce dernier. Vous pouvez, par exemple changer la valeur de la variable représentant le nombre joué par l'utilisateur.

2.5 Enfin, il s'agit de tester l'attribut *disabled* qui peut être utile pour désactiver un élément de l'interface en fonction d'une condition. Par exemple, cela pourra être utile pour désactiver le bouton "Jouer" lorsque le joueur a dépassé le nombre de tentative autorisés. Utilisez cet attribut pour désactiver le bouton jouer après une première tentative.

### Partie III: Réalisation du jeu du nombre mystère

Maintenant que nous avons exploités différents éléments d'interaction dans *SwiftUI*, il devient possible de réaliser la version graphique du jeu du nombre mystère.

3.1 Rajoutez une variable `nombreMystere` de type entier qui sera initialisé à partir d'un nombre aléatoire entre 1 et 100.

3.2 De la même façon, rajoutez une variable `indication` qui servira à afficher l'indication donné au joueur en fonction du nombre joué. Comme cette variable va être modifiée dynamiquement, il faudra rajouter `@State` devant la déclaration. Mettez à jour le champ texte pour qu'il affiche le contenu de la variable (dans le champ en bas de l'écran).

3.3 A l'intérieur de l'action du bouton *Jouer*, convertissez le contenu de la variable `nbJoue` en entier et la stocker dans une constante locale. **Rappel :** en swift vous pouvez convertir une chaîne de caractères en entier en utilisant le constructeur `Int()`.

3.4 Toujours au même endroit que précédemment, tester la valeur de votre constante et en fonction de la comparaison du nombre saisi par l'utilisateur, mettez à jour le contenu de la variable `indication`. Il s'agira de guider le joueur en indiquant "Trop grand", "Trop petit", "Gagné!", ou encore "vraiment trop grand" etc.



- 3.5 Rajoutez une variable pour compter le nombre d'essais du joueur. Affichez son contenu dans l'interface du jeu et incrémentez cette variable à chaque tentative.
- 3.6 Désactivez le bouton pour jouer lorsque le joueur a atteint 10 tentatives.
- 3.7 Rajoutez un bouton permettant au joueur de rejouer une fois qu'il a soit gagné, soit perdu. Le bouton permettra de re initialiser le jeu et de gérer les status des éléments du jeu.
- 3.8 Rajoutez un commentaire sur la performance du joueur une fois qu'il a gagné.

## Partie IV: Extension vers de nouvelles cibles : MacOS, VisionOS

Dans cette partie il s'agit de montrer l'intérêt de SwiftUI pour étendre une même application sur différentes système tels que MacOS ou VisionOS.

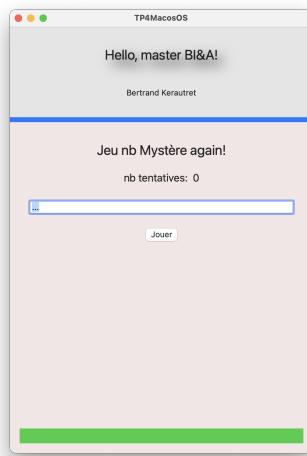
4.1 Rajoutez une nouvelle cible de façon à définir une version sur MacOS et supprimer la vue par défaut de façon à utiliser la vue de la partie précédente sans la dupliquer.

4.2 Faire de même autre cible comme VisionOS ou AppleTV.

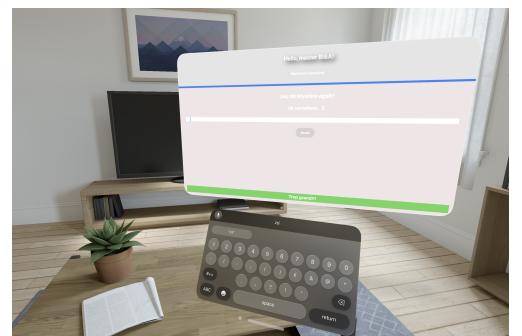
Vous pourrez obtenir les trois versions suivantes :



version IOS



version MacOS



version Vision OS