



TP : JDMI
Journée Découverte du
Métier d'Ingénieur


EpiMac 

Mars 2019



Ce TP a été conçu par :
Jean Saintavit – @LittleStranger
Tom-Eliott Herfray – @Find3r
Samy Hussaein – @Mysa

Version 2019.1¹

 *Apple, Mac, iPhone, iPad, iPod, Apple TV, Apple Watch* sont des marques déposées d'Apple Inc. Copyright © 2019 Apple Inc. Tous droits réservés.

Powered by  \LaTeX

1. Si vous remarquez une erreur ou un soucis lors de la compilation de votre projet, vous pouvez nous contacter directement à contact@epimac.org

Table des matières

1	À propos	4
2	🔌 Introduction	5
2.1	Swift : un langage moderne	5
2.2	Xcode : plus qu'un simple IDE	6
2.3	UX/UI : ce qui fait la différence	7
3	🎓 Les bases du Swift	8
3.1	Les variables	8
3.2	Les tableaux	8
3.3	Les conditions	9
3.4	Les boucles	10
4	</> Travail personnel	11
4.1	Introduction	11
4.1.1	Fonctionnement général de l'application	11
4.1.2	Explications sur la grille de jeu	11
4.2	Étape 1 : Ouvrir le projet	13
4.3	Étape 2 : Ajout de la grille et du label	15
4.4	Étape 3 : Ajout des 9 boutons	16
4.5	Étape 4 : La fonction play	17
4.6	Étape 5 : La fonction checkColumn	17
4.7	Étape 6 : La fonction checkLine	18
4.8	Étape 7 : Changement de la fonction play (1)	18
4.9	Étape 8 : La fonction isFull	18
4.10	Étape 9 : Changement de la fonction play (2)	18
5	🚀 Un peu plus loin	19
5.1	Déploiement du projet sur Git et utilisation des commandes UNIX	19
5.1.1	\$ git clone : Préparons le terrain	19
5.1.2	\$ git add : Prêt pour le lancement	20
5.1.3	\$ git commit : Quelques commentaires	20
5.1.4	\$ git push : Vers l'infini, et au-delà !	20
5.2	Conclusion et perspective	21

1 À propos



EpiMac est l'association Apple de *IONIS Education Group*. Depuis 1999, nous proposons des formations de développement informatique, des conférences sur le thème des nouvelles technologies, des ateliers pour maîtriser vos produits Apple, des soirées dédiées et plus encore.

“L’innovation, c’est une situation qu’on choisit parce qu’on a une passion brûlante pour quelque chose.”

— Steve Jobs

Venez découvrir le monde Apple avec des formations professionnelles Swift pour créer vos applications sur les plateformes d’Apple (macOS, iOS, tvOS et watchOS). Si vous êtes étudiant au sein d’une école partenaire², vous pouvez également bénéficier de réductions exclusives sur les produits Apple et d’un accès gratuit au portail Apple Developer. Si vous êtes intéressé, n’hésitez pas à nous suivre sur les réseaux sociaux ou sur notre site web.

 epimac.org
 [@EpiMac.org](https://www.facebook.com/EpiMac.org)
 [@EpiMac](https://twitter.com/EpiMac)
 [@epimacorg](https://www.instagram.com/epimacorg)
 [@epimac](https://www.linkedin.com/company/epimac)

2. Écoles partenaires : **EPITA**, **Epitech**, **ISG**, **ISEG**, **Moda Domani Institute**, **ISTH**, **ICS Bégué**, **ISEFAC**, **XP School**, **ESME Sudria**, **IPSA**, **SupBiotech**, **Webcadémie**, **e-artsup**, **IONIS-STM**, **Sup'Internet**, **etna** et **Coding Academy**. Liste complète : www.ionis-group.com



2 Introduction

Voici une petite introduction au Swift, à Xcode et à la notion d'UX/UI dans l'univers de la programmation.

2.1 Swift : un langage moderne



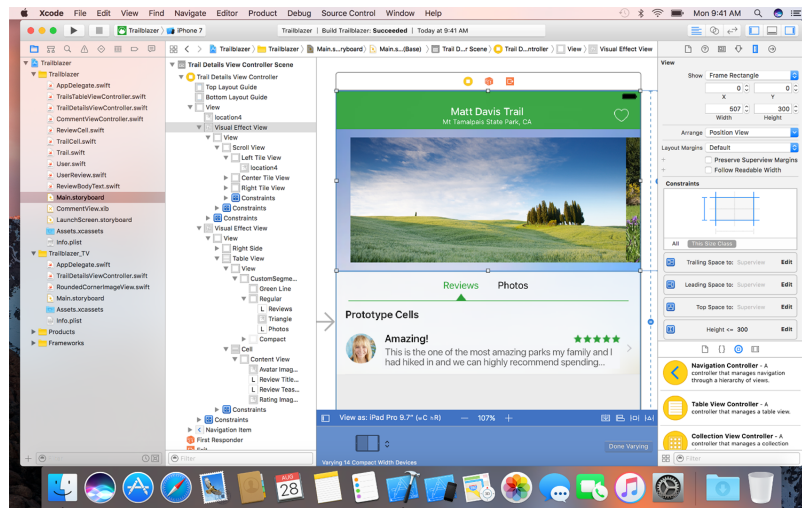
Conçu en 2014, le **Swift** est un langage de programmation objet compilé développé en open-source, et actuel langage principal des plateformes *macOS* (Macintosh), *iOS* (iPhone, iPad et iPod), *tvOS* (Apple TV) et *watchOS* (Apple Watch). Swift est également supporté sur des systèmes UNIX/Linux comme *Ubuntu*, *Debian* ou *Fedora*.

Chris Lattner, créateur du langage et ancien-ingénieur chez Apple, expliquait qu'il avait conçu Swift comme un langage capable de tout faire, du noyau d'un système d'exploitation aux scripts écrits par l'utilisateur, en passant par les apps ou le firmware d'un matériel. Le principal avantage de ce langage repose sur une très large bibliothèque de frameworks et librairies, qui sont parfois même compatibles entre les diverses plateformes,³, facilitant le développement d'applications.

En 2019, iOS est la plateforme où les développeurs gagnent le plus d'argent, largement devant Android. Avec plus de 1,2 milliards d'utilisateurs dans le monde, iOS n'est pas une plateforme négligeable et il est donc indispensable pour une entreprise ou un développeur de maîtriser la programmation d'applications sur cette plateforme. De plus, le Swift est un excellent langage pour commencer à programmer, il est très utilisé dans l'éducation et rivalise depuis quelques années avec l'éternel langage des "débutants", le Python.

3. Il s'agit du projet *Marzipan*, qui vise à "fusionner" les applications macOS et iOS.

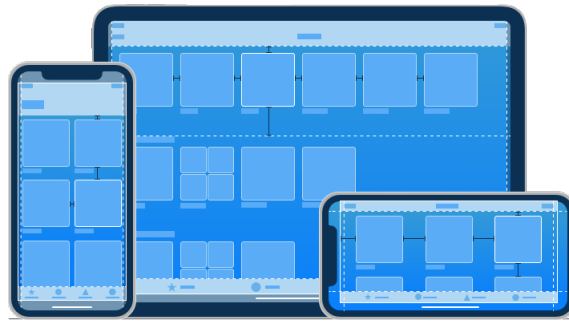
2.2 Xcode : plus qu'un simple IDE



Xcode est en environnement de développement (qu'on appelle souvent en anglais un *IDE* qui fonctionne sous macOS et qui permet de programmer et compiler des projets en Swift, en Objective-C (ancêtre du Swift mais toujours utilisé sur iOS et macOS) et en Ruby. L'avantage de cet IDE est qu'il gère nativement le déploiement d'une application sur l'App Store (sur les plateformes macOS, iOS, tvOS et watchOS) : c'est-à-dire que vous pouvez vous connecter avec votre compte *Apple Developer* et directement compiler/programmer votre projet puis l'envoyer à Apple pour vérification avant publication⁴. Vous retrouverez un guide d'utilisation d'Xcode dans la partie *Travaux pratiques* de ce sujet.

4. Pour gérer votre application après son déploiement, Apple utilise désormais le portail *App Store Connect* qui permet de faire un suivi régulier de votre application.

2.3 UX/UI : ce qui fait la différence



L'**UX** et l'**UI** sont deux choses très différentes dans la conception d'un projet, comme une application. L'*UX Design* se réfère au terme *Expérience Utilisateur*, tandis qu'*UI Design* signifie *User Interface Design*, interface utilisateur. Les deux éléments sont essentiels à un produit et travaillent en étroite collaboration. Pour se donner une idée, si vous imaginez un produit comme un corps humain par exemple, les OS représentent le code qui lui confère une structure. Les organes, eux, représentent la conception UX, c'est-à-dire mesurer et optimiser pour supporter les fonctions de vie. Et la conception de l'interface utilisateur représente les cosmétiques du corps : sa présentation, ses sens et ses réactions.

Vous le savez probablement, Apple est une entreprise que l'on peut qualifier de *minimaliste*. Lors du processus de création d'une application (ou plus généralement d'un projet), vous devez répondre à des critères et des attentes sur la façon dont vous gérez l'expérience utilisateur de votre projet. Alors bien sûr, nous n'allons pas nous focaliser sur l'UX/UI durant cette séance de TP, mais sachez que ces deux notions sont souvent essentielles pour le succès d'un projet, pour ne pas dire qu'elles sont *obligatoires*.

Apple délivre à ses développeurs la *Human Interface Guidelines*, un guide qui explique la façon dont vous gérez la conception visuelle de l'application (parcours utilisateur, aspect visuel, optimisation marketing). Il peut être intéressant, afin d'approfondir votre maîtrise de gestion de projets, d'y jeter un coup d'oeil⁵.

5. HIG Apple



3 Les bases du Swift

Passons maintenant à une partie importante : les bases du Swift. Pour ce TP, ce qui va nous intéresser principalement, ce sont les variables, les tableaux, les conditions et les boucles.

3.1 Les variables

Dans un langage de programmation on a besoin de stocker des éléments afin de les utiliser dans les différentes parties de notre programme.

Pour cela il existe les variables dont la déclaration est assez simple.

Une variable peut prendre quatre types différents : chaîne de caractères "string", nombre entier "int", booléen (vrai ou faux) "bool" et nombre à virgules "float".

```
var test = 42 // Type int
var hello = "Hello World" // Type string
var cond = true // Type bool
var life = 4.2 // Type float
```

3.2 Les tableaux

Afin de pouvoir stocker plusieurs éléments à la fois, nous utilisons des tableaux. Un tableau peut contenir un seul type de donnée et la taille du tableau ne change pas après sa déclaration.

```
var grades = [20, 18, 12, 8, 4, 2] // Tableau de taille 6
```

Pour accéder à un élément du tableau nous utilisons son index. L'index commence à 0 et peut aller jusqu'à la taille du tableau - 1

```
grades[0] // Accès à l'élément dont l'index est 0 (ici 20)
grades[3] // Accès à l'élément dont l'index est 3 (ici 8)
grades[6] // Erreur: l'index se trouve en dehors du tableau.
```


3.3 Les conditions

Afin de pouvoir interagir avec les données, nous disposons de deux possibilités : Les conditions et les boucles. On aimerait faire certaines choses si une condition est remplie. Imaginons que l'on ait une variable dont on ne connaît pas la valeur, et on souhaite savoir si elle est supérieure à 10. Pour cela nous utilisons la boucle if/else.

```
var resultat = "42"

if (inconnu > 10)
{
    resultat = "La variable est supérieure à 10"
}
else
{
    resultat = "La variable est inférieure ou égale à 10"
}
```

Comme nous avons pu le constater dans la condition du-dessus, pour comparer si un nombre est supérieur à un autre nous utilisons l'opérateur logique "<".

Ce n'est pas le seul opérateur logique qui existe. Voici un tableau avec tous les opérateurs logiques et leurs significations respectives.

Opérateur Logique	Signification
==	Egal
!=	Différent
>	Supérieur
<	Inférieur
>=	Supérieur ou égal
<=	Inférieur ou égal
==	Supérieur ou égal

3.4 Les boucles

Il existe deux boucles principales en Swift : le `for` et le `while`. La boucle `for` permet de répéter une même action autant de fois que souhaité et est très utilisée notamment pour parcourir les tableaux.

```
var grades = [20, 18, 12, 8, 4, 2]

for i in 0...5
{
    print(grades[i])
}

// Cette boucle affiche:
// 20
// 18
// 12
// 8
// 4
// 2
```

Ici la boucle déclare une variable `i` qui va prendre les valeurs successives suivantes : 0, 1, 2, 3, 4, et 5.

4 </> Travail personnel

4.1 Introduction

Parler, c'est bien. Agir c'est mieux. Nous allons passer à la partie la plus intéressante du TP, c'est-à-dire... le TP lui-même.

Le but de cette séance de TP est de vous familiariser avec l'interface Xcode et le Swift : vous allez concevoir une application iOS : un jeu du Morpion.

Rassurez-vous, nous n'allons pas vous laisser vous débrouiller seul. Si vous avez compris la partie *Les bases du Swift*, vous n'aurez pas de mal à faire ce TP.

Le Staff EpiMac reste bien sûr à votre disposition durant la séance.

4.1.1 Fonctionnement général de l'application

L'application est relativement simple. Nous connaissons tous le jeu du Morpion. Le jeu va se jouer à deux, chacun va jouer à son tour.

Le joueur concerné (1 ou 2 donc) voit une grille, et quand il clique sur l'une des cases (dans laquelle il y a un bouton), la fonction principale (la fonction `play`) se déclenche. La fonction `play` permet de placer une croix ou un rond si la case est vide, de vérifier si la partie est finie et de changer de tour.

La fonction `play`, ainsi ses fonctions associées, sont stockées dans le fichier `ViewController.swift`. L'interface utilisateur de notre jeu est stockée dans le fichier `Main.storyboard`.

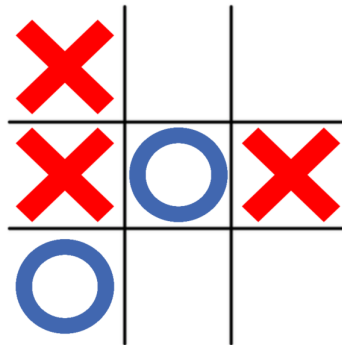
4.1.2 Explications sur la grille de jeu

Il est assez difficile de savoir si une case est vide, si elle est occupée par une croix ou un rond. Par conséquent, nous stockons les états des cases dans une liste que nous avons appelé `grid`. Chaque case de la liste contient l'état d'une case.

0	1	2
3	4	5
6	7	8

Rappel : En informatique, on commence à compter à partir de 0, et pas de 1.

Exemple :



On définit notre tableau⁶ avec :

```
grid = ["x", " ", " ", "x", "o", "x", "o", " ", ""]  
grid[0] = "x"  
grid[3] = "x"  
grid[4] = "o"  
grid[5] = "x"  
grid[6] = "o"
```

6. grid[i] désigne l'élément en place i de la grille.

4.2 Étape 1 : Ouvrir le projet

Pour commencer, ouvrez le logiciel **Xcode** qui se trouve dans votre dock (le lanceur d'application, situé dans la partie inférieure de votre écran).

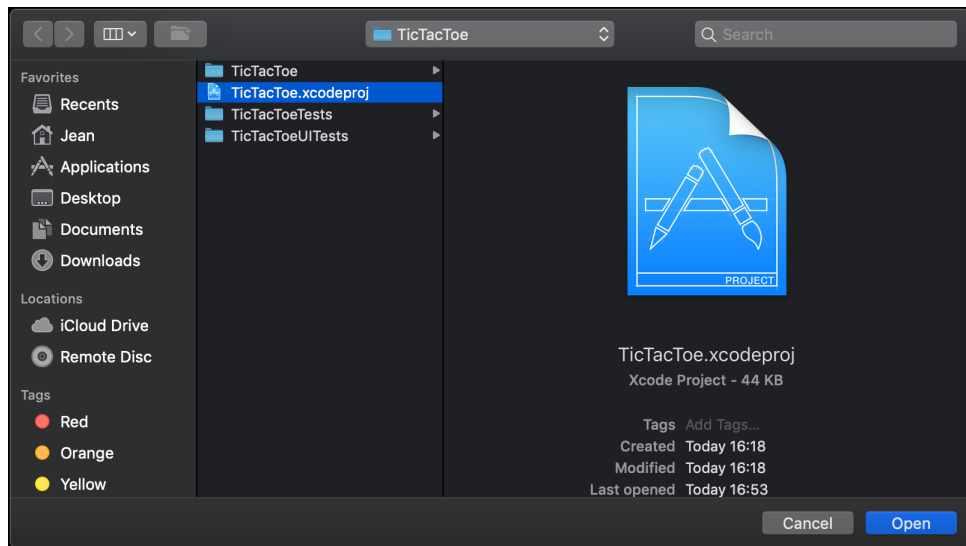


*Logo d'**Xcode** (si vous ne trouvez pas l'app dans le Dock, tapez cmd+espace pour lancer Spotlight et recherchez Xcode).*

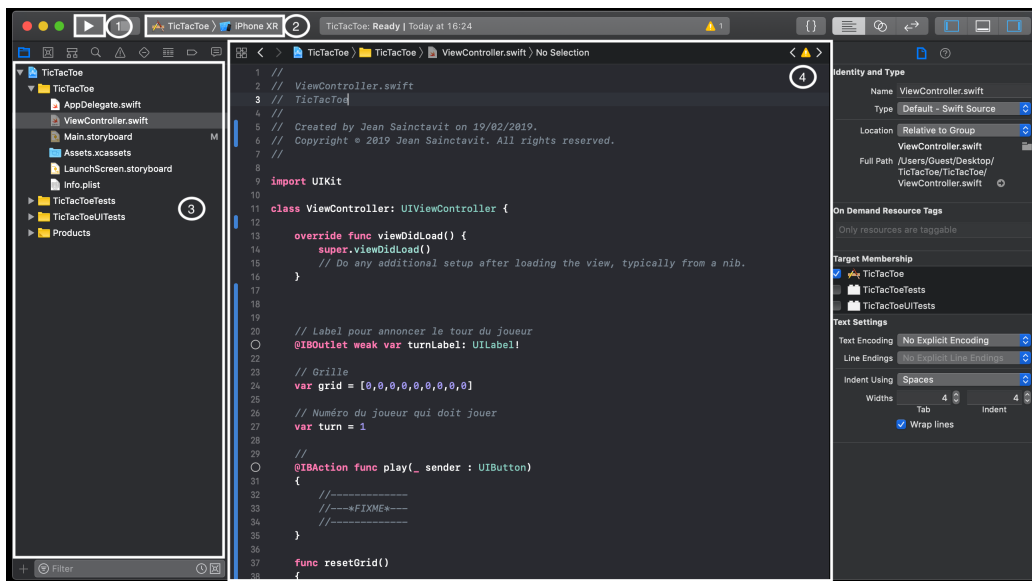
Maintenant, il ne vous reste plus qu'à ouvrir le projet déjà existant. Pour cela, cliquez sur *Open Another Project...* en bas à droite de la fenêtre.

Une boîte de dialogue va ensuite s'ouvrir pour vous proposer de choisir l'emplacement du dossier. Une fois que vous avez trouvé le bon chemin, sélectionnez le dossier, sans entrer dedans, puis appuyez sur *Open* en bas à droite de la boîte de dialogue.





Tada ! Vous voilà maintenant dans votre projet !



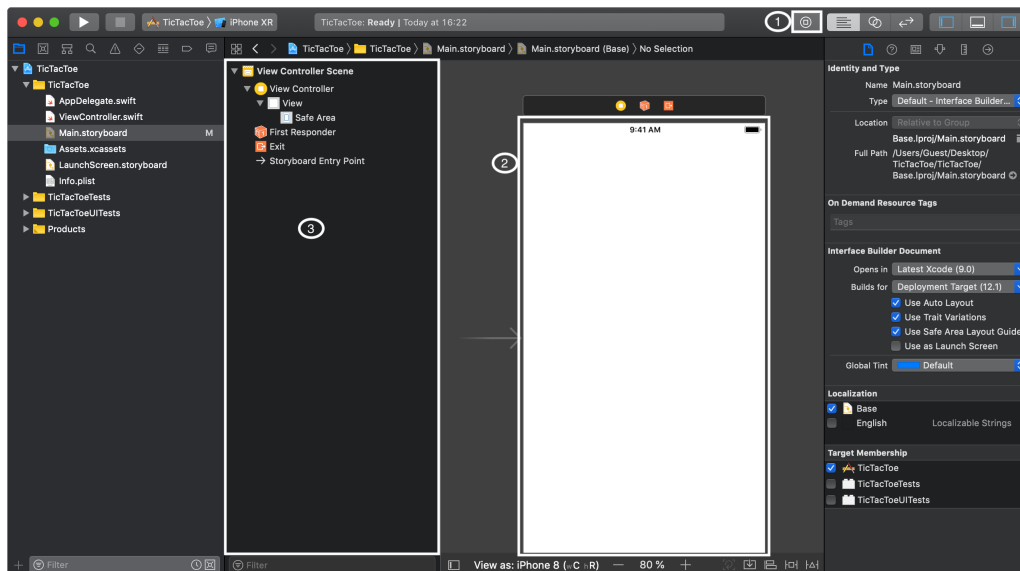
1. Bouton permettant de compiler le projet
2. Choix du simulateur de notre application
3. Affichage du fichier que nous traitons

- Affichage de l'arborescence du projet (Cette arborescence contient tout les fichiers de notre projet).

Nous ne modifierons que `ViewController.swift` et `Main.storyboard` pour ce TP.

4.3 Étape 2 : Ajout de la grille et du label

Allez maintenant dans le fichier `Main.storyboard`. Certains boutons sont apparus. Nous verrons en détail ce que ces boutons permettent de faire. Le storyboard représente ce que voit l'utilisateur de votre application



- Bouton permettant de créer un objet que vous pouvez mettre sur le view (l'interface que va voir l'utilisateur)
- Affichage de l'arborescence des éléments présents dans le storyboard
- Affichage de l'écran que va voir l'utilisateur.

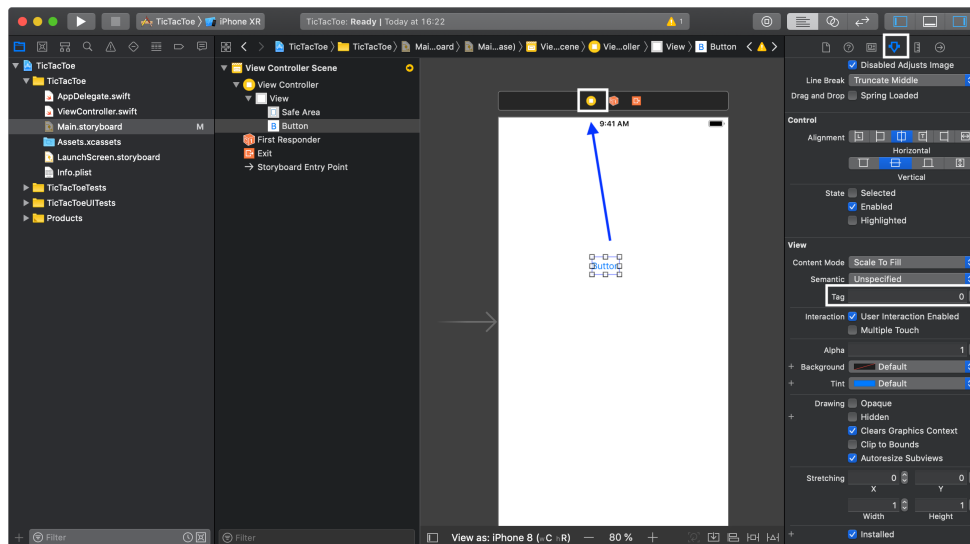
4.4 Étape 3 : Ajout des 9 boutons

Maintenant que vous avez ajouté la grille, il faut ajouter les boutons. Nous aurons besoin de 9 boutons, qui représentent les 9 cases de la grille. Commencez par ajouter 9 boutons, un pour chaque case de votre grille, puis ajustez-les et mettez les à la bonne taille. Pour ajouter un bouton, appuyez sur le bouton en haut à droite de la fenêtre, puis cherchez *button*, et faites glisser le bouton dans votre *view*.

Vous pouvez copier-coller des boutons avec cmd+c et cmd+v.

Malheureusement, il ne se passe rien quand on appuie sur les boutons. Pour remédier à ce problème, il faut relier les boutons à une fonction qui est définie dans le *ViewController*.

Cette fonction va permettre d'ajouter une croix ou un rond dans une case quand vous appuyez sur un bouton. Nous avons déjà défini cette fonction pour vous. Pour lier les boutons à la fonction, maintenez *Ctrl*, cliquez sur le bouton, puis faites glisser la flèche sur le bouton jaune situé en haut du *view*. Nous avons maintenant changé les *tags* des boutons. Les *tags* sont des identifiants présents sur tout ce que vous allez mettre dans le *storyboard*. Nous allons utiliser les *tags* pour différencier les boutons, car nous n'avons qu'une seule fonction pour les 9 boutons. Changez les *tags* des boutons. Le bouton en haut à gauche doit avoir un *tag* de 1, celui en haut doit avoir un *tag* de 2.



4.5 Étape 4 : La fonction `play`

Cette partie, ainsi que celles qui suivent, peuvent être un peu compliquées si vous n'avez jamais codé. Par conséquent, il est vivement recommandé de demander de l'aide à un Staff EpiMac si vous bloquez sur un point en particulier.

Réfléchissez bien avant d'écrire des lignes de code !

Tout d'abord, commencez par aller dans le fichier `ViewController`. Vous devez modifier la fonction `play` afin que les joueurs puissent placer des croix et des ronds. **Attention** : un joueur ne doit pas pouvoir modifier une case déjà modifiée par un autre joueur.

Astuce : `sender.setBackgroundImage(UIImage(named: "img"), for: UIControl.State())` permet de changer l'image présente sur le bouton `sender` (celui sur lequel l'utilisateur a appuyé). Remplacez `img` par `"x"` si vous voulez afficher une croix, et `"o"` si vous voulez afficher un rond.

Indice : Vous devez modifier l'apparence du bouton sur lequel vous avez appuyé, mais aussi la `grid`, le tableau qui contient l'état de la grille.

Rappels : `grid[0] = "x"` permet de changer la première valeur du tableau en `"x"`.

`grid[0] != "x"` va *return* vrai si la première valeur du tableau est différente de `"x"`.

4.6 Étape 5 : La fonction `checkColumn`

Nous allons maintenant programmer une fonction qui permet de voir si un joueur a rempli une colonne. Cette fonction doit renvoyer «
« si aucun des joueurs n'a rempli de colonne, `"x"` si le joueur 1 a rempli une colonne et `"o"` si le joueur 2 a rempli une colonne.

Indice : `return "abc"` permet de renvoyer la chaîne de caractère `"abc"`, et doit être utilisé dans votre code (en remplaçant `"abc"` par la chaîne de caractère de votre choix).

4.7 Étape 6 : La fonction `checkLine`

Cette fonction est similaire à `checkColumn`, sauf qu'elle vérifie si un joueur a rempli une ligne.

4.8 Étape 7 : Changement de la fonction `play` (1)

Nous avons programmé pour vous deux fonctions utiles : `checkWin` et `sendMessage`.

`checkWin` permet de vérifier si un des deux joueurs a gagné. `checkWin` renvoie " " si aucun joueur n'a gagné, "x" si le joueur 1 a gagné et "o" si le joueur 2 a gagné.

`sendMessage` permet d'envoyer une notification, et va prendre en paramètre le message qui va être affiché dans la notification.

Changez maintenant la fonction `play` pour afficher un message si l'un des joueurs a gagné.

4.9 Étape 8 : La fonction `isFull`

La fonction `isFull` doit retourner *True* si la grille est pleine, *False* sinon.

4.10 Étape 9 : Changement de la fonction `play` (2)

Vous devez maintenant changer la fonction `play` pour qu'elle vide la grille quand cette dernière est pleine.

5 Un peu plus loin

5.1 Déploiement du projet sur Git et utilisation des commandes UNIX

Pour cette dernière partie **facultative** du TP JDMI, nous allons travailler sur un outil extrêmement important en entreprise : `Git`. C'est est un logiciel de gestion de versions décentralisé, probablement le plus populaire de l'Internet. Il a été conçu par Linus Torvalds (créateur du noyau Linux) en 2005 et est utilisé aujourd'hui par la plupart des développeurs, ingénieurs et entreprises pour gérer "proprement" leurs projets.

Nous n'allons pas rentrer dans les détails de ce système car il ne s'agit pas ici d'un TP dédié à Git (GConfs⁷, une association épitéenne, organise par ailleurs un TP très intéressant en début d'année sur Git). Cependant, afin de garder une trace de votre projet, vous allez publier votre travail sur la plateforme populaire `GitHub`. Pour ce TP, nous avons déjà configuré la clé SSH qui vous permettra de vous identifier sur le serveur `GitHub` et d'envoyer votre travail.

5.1.1 `$ git clone` : Préparons le terrain

Ouvrez l'app **Terminal** et tapez les commandes suivantes :

```
$ cd Desktop/  
$ git clone git@github.com:EpiMac/JDMI_submission.git  
$ mv EpiMac_JDMI EpiMacFev19_prenom.nom  
$ mv /EpiMacFev19_prenom.nom /JDMI_submission
```

La première commande va simplement placer le Terminal dans le dossier Desktop (ou Bureau). La seconde va, elle, cloner le dossier depuis les serveurs de `GitHub`. Les troisième et quatrième commandes vont, respectivement, renommer puis déplacer votre travail dans le dossier qui sera envoyé sur les serveurs de `GitHub`.

7. Plus d'informations sur <https://gconfs.fr>

5.1.2 \$ git add : Prêt pour le lancement

Maintenant que vos fichiers sont au bon endroit, il est temps de les "préparer" au déploiement sur Git. Pour cela, rien de plus simple, tapez dans le Terminal les commandes suivantes :

```
$ cd JDMI_submission  
$ git add .
```

5.1.3 \$ git commit : Quelques commentaires

Afin de garder des "commits" propres, il est important de toujours mettre un commentaire résumant les modifications apportées à votre projet. Une commande peut suffire, et c'est la suivante :

```
$ git commit -m "prenom.nom"
```

5.1.4 \$ git push : Vers l'infini, et au-delà!

C'est l'heure du déploiement, et pour publier votre travail, c'est simple, tapez simplement cette *merveilleuse* commande magique dans le Terminal :

```
$ git push
```

Vous allez voir quelques alertes ou messages sur votre Terminal : rien d'inquiétant, il s'agit souvent de problème de langue (langage système, langage serveur, ...) ou d'une incohérence de formatage. N'hésitez pas à demander à un *Staff EpiMac* de venir voir si le commit apparaît bien sur la page GitHub.

Pour retrouver votre travail, vous pouvez vous rendre à l'adresse suivante : www.github.com/epimac/JDMI_submission. Vous retrouverez votre travail dans le dossier `EpiMacFev19_prenom.nom` (et même celui des autres). **Attention, si vous voulez refaire le projet chez vous, vous ne pourrez pas \$ git push sur ce Git car il est configuré en SSH spécifiquement sur nos machines à l'EPITA**⁸.

8. Nous vous invitons à suivre ce **tutoriel** qui explique comment configurer Git facilement chez vous

5.2 Conclusion et perspective


Félicitations, vous avez réussi à arriver jusqu'ici.

Il vous reste probablement quelques notions à voir, mais l'idée de ce TP était de vous donner un avant-goût des possibilités offertes par le *Swift*. Maintenant il ne reste qu'à personnaliser votre application.

Voici quelques idées pour personnaliser votre application :

- Concevoir un système de scores ;
- Changez les couleurs de l'interface ;
- Redesignez l'UX/UI⁹ ;
- Changez les ressources visuelles (photos, icônes, ...) utilisées.

Sachez par ailleurs que nous gérons la salle *SM15* (salle des TP JDMI) du campus Kremlin-Bicêtre de l'EPITA. Cette salle, équipée de plusieurs Mac, est ouverte H24 7J/7, donc vous pouvez venir utiliser les Mac n'importe quand. Nous possédons également plusieurs locaux au sein du campus, afin d'y stocker notre matériel et d'y organiser des séances avancées de développement Swift ou autre. De plus, nous offrons aux étudiants de l'école des licences gratuites au portail *Apple Developer* (normalement 99€/an) qui vous permet de publier, gérer et monnayer votre application Swift sur l'App Store.

Si vous avez d'autres questions ou que vous souhaitez en savoir plus sur le *formidable* monde d'Apple ou même tout simplement sur la vie à l'EPITA, les études, l'ingénierie ou autre : n'hésitez pas à nous contacter directement sur notre page Facebook ( : EpiMac). Nous serons heureux de vous répondre !

Think different. 

9. Nous vous invitons à revoir la section *UX/UI* : ce qui fait la différence au début du sujet.