Introduction aux animations dans .NET MAUI

Les animations constituent un élément essentiel dans le développement d'interfaces utilisateur modernes. Elles permettent d'améliorer l'expérience utilisateur en fournissant des retours visuels sur les interactions et en rendant l'application plus dynamique et engageante.

.NET MAUI offre un système d'animation puissant et flexible qui peut être facilement utilisé dans le code-behind pour créer des interfaces utilisateur interactives et dynamiques.

Théorie des animations dans .NET MAUI

Types d'animations de base

.NET MAUI prend en charge plusieurs types d'animations fondamentales :

1. **Fade (Opacité)** : Modifications de la propriété `Opacity` pour faire apparaître ou disparaître progressivement un élément.
2. **Rotate (Rotation)** : Utilisation de la propriété `Rotation` pour faire pivoter un élément selon un angle spécifié.
3. **Scale (Mise à l'échelle)** : Modification de la propriété `Scale` pour agrandir ou réduire un élément.
4. **Translate (Translation)** : Déplacement d'un élément en modifiant ses propriétés `TranslationX` et `TranslationY`.
5. Et d'autres encore

Méthodes d'animation

.NET MAUI fournit plusieurs méthodes d'extension pour animer des éléments visuels :

- `FadeTo()` : Anime la propriété d'opacité d'un élément.
- `RotateTo()` : Anime la rotation d'un élément.
- `ScaleTo()` : Anime la taille d'un élément.
- `TranslateTo()` : Anime la position d'un élément.

Ces méthodes retournent des `Task<bool>`, ce qui permet de les utiliser avec `async/await` pour créer des séquences d'animations.

Gestion des animations avec le code-behind

Dans une approche code behind, toute la logique de l'application et la gestion des animations sont centralisées dans le fichier xaml.cs de la page.

Les éléments clés de cette approche sont :

1. **Gestionnaires d'événements** : Les contrôles UI (boutons, switches, sliders) utilisent des événements pour déclencher des actions dans le code-behind.
2. **Accès direct aux éléments UI** : Le code-behind peut directement accéder aux éléments nommés (via `x:Name`) pour appliquer des animations.
3. **Logique métier dans le code-behind** : Toute la logique (état de rotation, calcul d'angles, gestion de vitesse) est implémentée directement dans le code-behind ou éventuellement dans une classe de service dédiée.

Architecture simplifiée

Dans cette approche, le flux d'exécution est direct :

1. L'utilisateur interagit avec un contrôle UI (par exemple, active un Switch).
2. Un gestionnaire d'événements dans le code-behind est déclenché.
3. Le gestionnaire met à jour l'état de l'application et déclenche les animations nécessaires.
4. Les méthodes d'animation .NET MAUI sont appelées directement sur les éléments UI : `box.RotateTo(angle)` .

Exemple pas à pas : Animation d'un BoxView

Suivons maintenant un exemple concret en nous basant sur le code fourni.

Étape 1 : Création de la Vue XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiAnimationDemo.Views.AnimatePage"
             Title="Animations MAUI avec Code-Behind">

    <Grid Padding="20">
        <VerticalStackLayout Spacing="10">

            <!-- BoxView qui sera animé -->
            <BoxView x:Name="box"
                     Margin="0,30,0,0"
                     Color="Aqua"
                     WidthRequest="150"
                     HeightRequest="150"
                     HorizontalOptions="Center" />

            <!-- Switch de rotation -->
            <HorizontalStackLayout HorizontalOptions="Center" Margin="0,25,0,0">
                <Label x:Name="rotationLabel"
                       Text="Rotation inactive"
                       VerticalTextAlignment="Center" />
                <Switch x:Name="rotationSwitch"
                        Toggled="OnRotationToggled" />
            </HorizontalStackLayout>

            <!-- Déplacement avec les boutons -->
            <HorizontalStackLayout HorizontalOptions="Center" Margin="0,25,0,0">
                <Button Text="&lt;&lt;"
                        Clicked="OnMoveLeft" />
                <Button Text=">>"
                        Margin="10,0,0,0"
                        Clicked="OnMoveRight" />
            </HorizontalStackLayout>

            <!-- Ajustement de la vitesse -->
            <StackLayout Orientation="Horizontal" HorizontalOptions="Center">
                <Label x:Name="speedLabel"
                       Text="Speed 10" />
```

```
                <Slider x:Name="speedSlider"
                        WidthRequest="150"
                        Value="10"
                        Minimum="1"
                        Maximum="50"
                        ValueChanged="OnSpeedChanged" />
            </StackLayout>

        </VerticalStackLayout>
    </Grid>
</ContentPage>
```

Étape 2 : Création du Code-Behind

```
namespace MauiAnimationDemo.Views
{
    public partial class AnimatePage : ContentPage
    {
        // Variables d'état
        private double _currentX = 0;
        private int _currentAngle = 0;
        private bool _isRotating = false;
        private int _speed = 10;

        public AnimatePage()
        {
            InitializeComponent();

            // Démarrer la tâche de rotation en arrière-plan
            StartRotationTask();
        }

        // Gestionnaire d'événement pour le changement d'état du Switch
        private void OnRotationToggled(object sender, ToggledEventArgs e)
        {
            _isRotating = e.Value;
            rotationLabel.Text = e.Value ? "Rotation active" : "Rotation inactive";
        }

        // Gestionnaire d'événement pour le bouton de déplacement vers la gauche
        private async void OnMoveLeft(object sender, EventArgs e)
        {
            _currentX -= 50; // Déplacement de 50 pixels vers la gauche
            await box.TranslateTo(_currentX, 0, 250, Easing.SpringOut);
        }

        // Gestionnaire d'événement pour le bouton de déplacement vers la droite
        private async void OnMoveRight(object sender, EventArgs e)
        {
            _currentX += 50; // Déplacement de 50 pixels vers la droite
            await box.TranslateTo(_currentX, 0, 250, Easing.SpringOut);
```

```
        }

        // Gestionnaire d'événement pour le changement de valeur du Slider
        private void OnSpeedChanged(object sender, ValueChangedEventArgs e)
        {
            _speed = (int)e.NewValue;
            speedLabel.Text = $"Speed {_speed}";
        }

        // Tâche en arrière-plan pour gérer la rotation continue
        private async void StartRotationTask()
        {
            while (true)
            {
                if (_isRotating)
                {
                    _currentAngle = (_currentAngle + _speed) % 360;
                    await box.RotateTo(_currentAngle, 0); // Rotation sans animation
(instantanée)
                }

                // Attendre pour contrôler la fréquence des mises à jour
                await Task.Delay(50);
            }
        }
    }
}
```

Explication détaillée de l'exemple

Animations

1. **Vue (AnimatePage.xaml)** : Définit la structure de l'interface utilisateur, y compris le BoxView à animer et les contrôles pour déclencher les animations (boutons, switch, slider). Chaque contrôle interactif est nommé avec `x:Name` et possède des gestionnaires d'événements.

2. **Code-behind (AnimatePage.xaml.cs)** : Contient toute la logique de l'application :

   - Les variables d'état (`_currentX` , `_currentAngle` , `_isRotating` , `_speed`)
   - Les gestionnaires d'événements pour répondre aux interactions utilisateur
   - La logique d'animation et les appels aux méthodes d'animation .NET MAUI
   - La tâche en arrière-plan pour la rotation continue

Gestion des événements

La communication entre l'interface utilisateur et la logique se fait via des événements standard :

1. **Événement Toggled** : Déclenché lorsque l'utilisateur active ou désactive le Switch. Le gestionnaire `OnRotationToggled` met à jour l'état `_isRotating` et le texte du Label.

2. **Événements Clicked** : Déclenchés lorsque l'utilisateur clique sur les boutons de déplacement. Les gestionnaires `OnMoveLeft` et `OnMoveRight` calculent la nouvelle position

et appliquent l'animation de translation.
3. **Événement ValueChanged** : Déclenché lorsque l'utilisateur ajuste le Slider. Le gestionnaire `OnSpeedChanged` met à jour la vitesse et le texte du Label.

Types d'animations implémentés

Dans cet exemple, nous avons implémenté deux types d'animations :

1. **Rotation (RotateTo)** : Le BoxView tourne en fonction de la valeur de l'angle fournie par le ViewModel.
   - Contrôlée par un Switch qui active/désactive la rotation continue.
   - La vitesse de rotation est ajustable via un Slider.
2. **Translation (TranslateTo)** : Le BoxView se déplace horizontalement en réponse aux boutons de direction.
   - Les boutons "<<" et ">>" déplacent le BoxView vers la gauche ou la droite.
   - L'animation utilise un effet d'élasticité (`Easing.SpringOut`) pour un rendu plus naturel.

Résumé des concepts clés

1. **Gestionnaires d'événements** : Les interactions utilisateur déclenchent des événements standard (Toggled, Clicked, ValueChanged) qui sont traités par des méthodes dans le code-behind.
2. **Accès direct aux éléments UI** : Le code-behind accède directement aux éléments nommés (via `x:Name`) pour appliquer les animations et mettre à jour l'interface.
3. **Animation continue** : Utilisation d'une tâche en arrière-plan (`StartRotationTask`) pour créer une animation continue (rotation) basée sur un état booléen et un paramètre de vitesse.
4. **Animation ponctuelle** : Déclenchement d'animations à la demande (déplacement) en réponse aux clics sur les boutons.
5. **Méthodes d'animation .NET MAUI** : Utilisation des méthodes d'extension d'animation intégrées (`RotateTo` , `TranslateTo`) qui retournent des `Task` et peuvent être utilisées avec `async/await` .