AGENDA 11

Background degradê, Grid, Estilos e aprofundamento do manipulador de eventos

GEEaD - Grupo de Estudos de Educação a Distância

Centro de Educação Tecnológica Paula Souza

GOVERNO DO ESTADO DE SÃO PAULO

EIXO TECNOLÓGICO DE INFORMAÇÃO E COMUNICAÇÃO

CURSO TÉCNICO EM DESENVOLVIMENTO DE SISTEMAS

PROGRAMAÇÃO MOBILE I

Expediente

Autores:

GUILHERME HENRIQUE GIROLLI

TIAGO ANTONIO DA SILVA

Revisão Técnica:
Kelly Cristiane de Oliveira Dal Pozzo

São Paulo – SP, 2024

MERGULHANDO NO TEMA

Vamos agora desenvolver o aplicativo "**Jogo da Velha**" proposto pelo Gustavo. Esse aplicativo conta com alguns recursos básicos de customização da interface do usuário. Essas ferramentas são utilizadas para que a interface seja agradável para quem utiliza o sistema. Em um primeiro momento vamos elencar alguns pontos importantes nesse processo.

O .NET MAUI oferece uma estrutura poderosa e flexível para o desenvolvimento de interfaces gráficas em aplicativos multiplataforma. Neste contexto, recursos visuais como backgrounds degradê, o controle Grid, e a utilização de estilos são fundamentais para criar uma experiência de usuário atraente e consistente.

- ✓ **Background degradê**: Com o LinearGradientBrush e o RadialGradientBrush, é possível aplicar transições suaves de cores, dando profundidade e modernidade ao design. Esses recursos permitem criar fundos dinâmicos e customizados que se adaptam a diferentes resoluções e dispositivos.
- ✓ **Grid**: O controle Grid é uma ferramenta essencial para o layout de interfaces, permitindo organizar elementos de forma estruturada através de linhas e colunas. Sua flexibilidade facilita a criação de interfaces responsivas e ajustáveis, sendo um dos pilares do layout em MAUI.
- ✓ **Estilos**: Para garantir consistência e reutilização de definições visuais, os **estilos** em MAUI são utilizados para padronizar elementos como botões, textos e layouts. Estilos podem ser aplicados tanto localmente quanto globalmente, tornando a manutenção e atualização visual mais eficiente.
- ✓ **Manipulador de eventos**: O manejo de interações e ações do usuário é feito através dos manipuladores de eventos, que conectam eventos como cliques e toques aos métodos responsáveis por executar a lógica do aplicativo. O MAUI permite um controle avançado sobre esses eventos, oferecendo mecanismos para capturar e manipular interações de forma detalhada e eficiente.

A escolha das cores é fundamental para o desenvolvimento da interface do usuário. Todos nós já sabemos que a escolha e combinação de cores não deve ocorrer por acaso. É um processo que requer um estudo sobre o significado de cada cor. Você deve se lembrar das agendas de Design Digital, que estudou no módulo I! O vermelho, por exemplo, é considerado um tom quente, porque transmite energia e coragem, facilitando até mesmo uma ação do usuário. Já o verde, é considerado uma cor fria, por transmitir segurança.

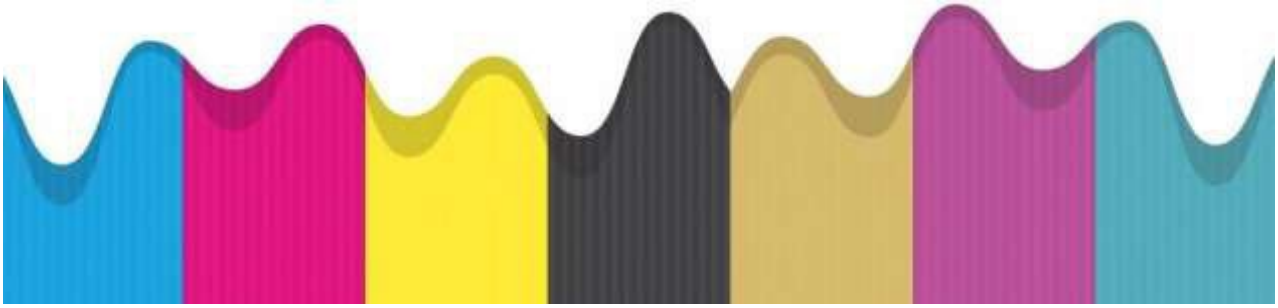


Figura 1 - Fonte: www.freepik.com

Utilizar algumas técnicas para escolher as cores certas é fundamental para que o projeto fique harmonioso e atrativo, contudo, a validação por parte do cliente é também muito importante. Imagine o desastre que seria, caso o desenvolvedor de um aplicativo da Coca-Cola utilizasse a cor azul, da Pepsi, sua principal concorrente!

Por isso, é muito importante levar em consideração as aulas de Design Digital, estudadas no módulo I, para saber como utilizar as cores e combinações a fim de trazer harmonia para os seus aplicativos. Ao final dessa agenda você encontra um vídeo sobre cores. Vale a pena conferir!

Outro ponto importante diz respeito a utilização de botões. É fundamental destacar com cores ou movimentos os botões responsáveis por desempenhar ações importantes em seu aplicativo.

Background degradê com Lineargradientbrush

O LinearGradientBrush é uma ferramenta poderosa em XAML (Extensible Application Markup Language) para criar gradientes de cores lineares em elementos de interface do usuário. Ele permite que você defina uma transição suave entre duas ou mais cores ao longo de uma direção específica. Aqui está uma explicação sobre como usar o LinearGradientBrush para definir o background de um elemento em XAML:

1. **Definindo um LinearGradientBrush:** Para utilizar o LinearGradientBrush, você precisa definir um novo recurso (resource) dentro do escopo do elemento que deseja estilizar. Este recurso representa o gradiente de cores que você deseja aplicar. Você pode definir a direção do gradiente usando as propriedades StartPoint e EndPoint, que especificam onde o gradiente começa e termina.

2. **Definindo as Cores:** Você pode definir as cores do gradiente usando a propriedade GradientStops, que aceita uma coleção de GradientStop. Cada GradientStop define uma cor e uma posição relativa dentro do gradiente, permitindo que você crie transições suaves entre as cores.

3. **Aplicando o Background:** Depois de definir o LinearGradientBrush, você pode aplicá-lo como o background de um elemento usando a propriedade Background. Por exemplo, você pode aplicá-lo a um elemento como uma Grid, um StackLayout ou qualquer outro elemento que aceite um background.

   Segue um exemplo de utilização do LinearGradientBrush para definir o background de um elemento em XAML.

```
<ContentPage.Resources>
    <ResourceDictionary>
        <LinearGradientBrush x:Key="MeuGradiente" StartPoint="0,0" EndPoint="1,1">
            <GradientStop Color="Yellow" Offset="0.0"/>
            <GradientStop Color="Red" Offset="1.0"/>
        </LinearGradientBrush>
    </ResourceDictionary>
</ContentPage.Resources>

<Grid Background="{StaticResource MeuGradiente}">
    <!-- Conteúdo da Grid aqui -->
</Grid>
```

No exemplo apresentado, um gradiente linear é definido entre as cores amarela e vermelha, indo do canto superior esquerdo (StartPoint) ao canto inferior direito (EndPoint). Esse gradiente é então aplicado como o background de uma Grid, resultando em um background com um gradiente suave entre as cores especificadas.

O LinearGradientBrush oferece uma maneira flexível e poderosa de estilizar elementos de interface do usuário com gradientes de cores lineares em XAML, permitindo criar visuais atraentes e dinâmicos em seus aplicativos .NET MAUI.

Layout em Grid

O layout em Grid é uma das opções de layout disponíveis no .NET MAUI para organizar elementos de interface do usuário em linhas e colunas, proporcionando um alto nível de controle sobre o posicionamento e o dimensionamento dos elementos. Algumas informações sobre o layout em Grid e principais propriedades:

- ✓ **Organização em Linhas e Colunas:** O layout em Grid permite organizar os elementos em linhas e colunas, fornecendo flexibilidade para posicionar os elementos precisamente onde você deseja na interface do usuário.
- ✓ **Definição de Linhas e Colunas:** É possível definir o número e o tamanho das linhas e colunas no layout em Grid usando as propriedades RowDefinitions e ColumnDefinitions. Isso permite criar uma grade flexível que se adapta às necessidades do seu layout.
- ✓ **Posicionamento de Elementos:** Os elementos em Grid são posicionados usando as propriedades Grid.Row e Grid.Column, que especificam em qual linha e coluna o elemento deve ser colocado. Isso permite criar layouts complexos com elementos em sobreposição e em diferentes áreas da grade.
- ✓ **Alinhamento de Elementos:** O layout em Grid oferece várias opções para alinhar elementos dentro das células da grade, incluindo HorizontalOptions e VerticalOptions, que determinam como os elementos se comportam em relação ao espaço disponível na célula.
- ✓ **Spanning (Expansão):** um elemento pode ocupar mais de uma linha ou coluna na grade usando as propriedades Grid.RowSpan e Grid.ColumnSpan, permitindo criar layouts mais complexos com elementos que abrangem várias células.
- ✓ **Margens e Espaçamento:** Assim como em outros layouts do .NET MAUI, é possível definir margens e espaçamentos entre os elementos no layout em Grid para controlar o espaço em branco ao redor dos elementos e entre as linhas e colunas da grade.

O layout em Grid no .NET MAUI oferece uma maneira poderosa e flexível de criar layouts complexos e responsivos em aplicativos multiplataforma. Com o layout em Grid, é possível criar interfaces de usuário ricas e visualmente atraentes que se adaptam a uma variedade de tamanhos de tela e dispositivos.

Estilos Style e Setter

É possível estilizar elementos de interface do usuário usando a classe Style em conjunto com a classe Setter. O Style define um conjunto de propriedades de estilo que serão aplicadas a um ou mais elementos, enquanto o Setter define os valores das propriedades individuais que compõem o estilo.

Como usar Style e Setter para estilizar elementos?

1. **Definindo um Estilo**: usando a classe Style é possível definir um estilo atribuindo um ou mais elementos de interface do usuário. Um estilo é um conjunto de propriedades de estilo que podem incluir qualquer número de Setters, cada um dos quais define o valor de uma propriedade específica.
2. **Definindo Setters:** Setter é usado para definir o valor de uma propriedade de estilo específica. É possível especificar o nome da propriedade que deseja definir e o valor que deseja atribuir a essa propriedade. Por exemplo, você pode definir a cor de fundo de um elemento usando um Setter para a propriedade BackgroundColor.

3. **Aplicando o Estilo:** Depois de definir um estilo com os Setters desejados, aplicamos a um elemento de interface do usuário usando a propriedade Style. Isso permite a reutilização dos estilos em vários elementos, mantendo uma aparência consistente em todo o seu aplicativo.

4. **Hierarquia de Estilos:** Em .NET MAUI, os estilos podem ser definidos localmente em um arquivo XAML, em um arquivo de recursos compartilhados (como um arquivo ResourceDictionary) ou globalmente em nível de aplicativo. A hierarquia de estilos permite que você aplique estilos em diferentes níveis, com estilos definidos em níveis mais baixos substituindo estilos definidos em níveis mais altos.

```
<ContentPage.Resources>
    <ResourceDictionary>
        <Style x:Key="MeuEstilo" TargetType="Label">
            <Setter Property="TextColor" Value="Red" />
            <Setter Property="FontSize" Value="20" />
        </Style>
    </ResourceDictionary>
</ContentPage.Resources>

<StackLayout>
    <Label Style="{StaticResource MeuEstilo}" Text="Este é um label estilizado" />
</StackLayout>
```

No exemplo apresentado foi definido um estilo chamado MeuEstilo, para elementos do tipo Label, onde a cor do texto é definida como vermelho e o tamanho da fonte é definido como 20. Esse estilo é então aplicado a um Label dentro de um StackLayout, resultando em um Label estilizado de acordo com as propriedades definidas no estilo.

Manipulador de Eventos: Sender

O parâmetro sender em um método que manipula o evento Click é uma convenção comum em muitas estruturas de desenvolvimento de software, incluindo .NET MAUI. Ele se refere ao objeto que acionou o evento, ou seja, o elemento de interface do usuário (como um botão) no qual o clique ocorreu.

Quando um evento como o Click é acionado em um elemento de interface do usuário, o sistema de eventos passa informações sobre esse evento para o método associado a ele. O parâmetro sender é uma maneira de acessar o objeto que gerou o evento dentro desse método.

Algumas situações em que o parâmetro sender pode ser útil:

1. **Identificação do Elemento:** O sender permite a identificação de qual elemento específico de interface do usuário acionou o evento. Isso é especialmente útil se tiver vários elementos que compartilham o mesmo manipulador de eventos.

2. **Acesso às Propriedades do Elemento:** Com o sender, é possivel acessar as propriedades e métodos do elemento de interface do usuário que acionou o evento. Isso pode ser útil para fazer alterações dinâmicas com base na interação do usuário.

3. **Ações Diferentes para Diferentes Elementos:** Tendo vários elementos que compartilham o mesmo manipulador de eventos, o Sender permite que execute ações diferentes com base no elemento que

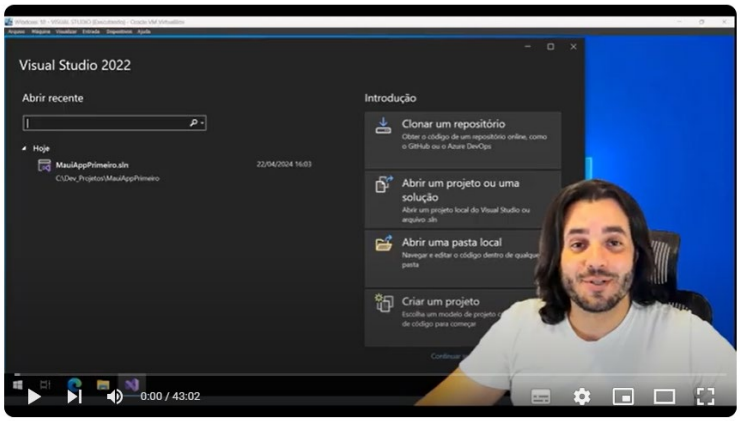
acionou o evento. Você pode usar condições ou técnicas de conversão de tipo para determinar qual ação deve ser executada.

```
4. private void MeuManipuladorDeEvento(object sender, EventArgs e)
5. {
6.     // Converte o sender para o tipo correto (por exemplo, Button)
7.     Button botao as sender;
8.
9.    // Acesso às propriedades ou métodos do botão que acionou o evento
10.   botao.Text = "Clicado!";
11.}
```

No exemplo apresentado, o método MeuManipuladorDeEvento manipula o evento Click e usa o parâmetro sender para acessar o botão específico que acionou o evento. Ele então altera o texto do botão para "Clicado!" em resposta ao clique.

Criando o Projeto

Assista a vídeo aula explicativa sobre o projeto:



Disponível em: https://www.youtube.com/watch?v=cuNAjx4B-0o

Codificação do Projeto:

Código 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="MauiAppJogoDaVelha.MainPage">

    <ContentPage.Background>
        <LinearGradientBrush StartPoint="0,1" EndPoint="1,0">
            <GradientStop Color="Purple" Offset="0.3" />
            <GradientStop Color="Red" Offset="1.1" />
        </LinearGradientBrush>
```

```
    </ContentPage.Background>

    <ContentPage.Resources>
        <Style TargetType="Button">
            <Setter Property="BackgroundColor" Value="Transparent" />
            <Setter Property="CornerRadius" Value="15" />
            <Setter Property="TextColor" Value="White" />
            <Setter Property="FontSize" Value="80" />
            <Setter Property="BorderWidth" Value="1" />
            <Setter Property="BorderColor" Value="White" />
        </Style>
    </ContentPage.Resources>

    <Grid RowDefinitions="*, *, *, *"
          ColumnDefinitions="*, *, *"
          ColumnSpacing="10" RowSpacing="10" Margin="10">

        <Label Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="3"
               Text="Jogo da Velha" />

        <Button x:Name="btn10" Grid.Row="1" Grid.Column="0" Clicked="Button_Clicked" />
        <Button x:Name="btn11" Grid.Row="1" Grid.Column="1" Clicked="Button_Clicked" />
        <Button x:Name="btn12" Grid.Row="1" Grid.Column="2" Clicked="Button_Clicked" />

        <Button Grid.Row="2" Grid.Column="0" Clicked="Button_Clicked" />
        <Button Grid.Row="2" Grid.Column="1" Clicked="Button_Clicked" />
        <Button Grid.Row="2" Grid.Column="2" Clicked="Button_Clicked" />

        <Button Grid.Row="3" Grid.Column="0" Clicked="Button_Clicked" />
        <Button Grid.Row="3" Grid.Column="1" Clicked="Button_Clicked" />
        <Button Grid.Row="3" Grid.Column="2" Clicked="Button_Clicked" />

    </Grid>

</ContentPage>
```

Codificação Back-end, C#:

```
namespace MauiAppJogoDaVelha
{
    public partial class MainPage : ContentPage
    {
        string vez = "X";

        public MainPage()
        {
            InitializeComponent();
```

```
        }

        private void Button_Clicked(object sender, EventArgs e)
        {
            Button btn = (Button)sender;

            btn.IsEnabled = false;

            if(vez == "X")
            {
                btn.Text = "X";
                vez = "O";
            } else
            {
                btn.Text = "O";
                vez = "X";
            }

            /* Verificando se o X ganhou na 1ª linha */
            if(btn10.Text == "X" && btn11.Text == "X" && btn12.Text == "X")
            {
                DisplayAlert("Parabéns!", "O X ganhou!", "OK");
                Zerar();
            }


        } // Fecha método

        void Zerar()
        {
            btn10.Text = "";
            btn11.Text = "";
            btn12.Text = "";

            btn10.IsEnabled = true;
            btn11.IsEnabled = true;
            btn12.IsEnabled = true;
        }

    } // Fecha Classe
} // Fecha Namespace
```

CÓDIGO-FONTE COMPLETO

https://github.com/tiagotas/MauiAppJogoDaVelha