Programacao Orientada a Objetos

GCT052 - Aula 2.2 - Jogo de Combinação (Animal Matching Game)

Bento Rafael Siqueira

Universidade Federal de Lavras (UFLA)

1 de setembro de 2025

Sumario

Conteudo da aula:

- Introducao ao Jogo de Combinacao
- Estrutura do Projeto
- Passo a Passo da Implementacao
- Detalhes da Grade 4x4
- Conceitos Aplicados
- Referencias aos Codigos
- Recursos e Referencias
- Resumo

O que vamos aprender hoje?

Objetivos da aula:

- **Implementar** o Jogo de Combinação completo
- **Aplicar** conceitos de WPF e XAML
- **Desenvolver** lógica de jogo interativo
- **Praticar** programação orientada a eventos

Base: Head First C# 4th Edition - Animal Matching Game

O que e o Jogo de Combinacao?

Descrição do jogo:

- **Objetivo:** Encontrar pares de animais iguais
- **Interface:** Grade 4x4 simples
- **Mecânica:** Clicar em duas cartas para verificar
- **Vitória:** Todos os 8 pares encontrados

Conceitos básicos: TextBlock, Button, Eventos

Estrutura do projeto WPF

Arquivos principais:

- **MainWindow.xaml** - Interface simples do jogo
- **MainWindow.xaml.cs** - Lógica básica (100 linhas)
- **App.xaml** - Configuração mínima
- **App.xaml.cs** - Inicialização padrão

Organização: Interface (XAML) + Lógica (C#)

Interface XAML - Estrutura basica

Layout principal:

- **Grid** como container principal (3 linhas)
- **TextBlock** para título do jogo
- **TextBlock** para status (StatusText)
- **Grid** para grade 4x4 das cartas (GameGrid)
- **Button** para novo jogo (RestartButton)

Design: Interface simples e clara

Passo 1: Criar o projeto WPF

Configuração inicial:

1. **File** → **New** → **Project**
2. Selecionar **"WPF Application"**
3. Nome: **"AnimalMatchingGame"**
4. Framework: **.NET 9.0**
5. Clicar em **Create**

Verificação:

- Projeto criado com estrutura básica
- MainWindow.xaml e MainWindow.xaml.cs presentes
- Ambiente pronto para desenvolvimento

Passo 2: Configurar a interface XAML

Elementos da interface:

- **Título:** "Animal Matching Game"
- **Contador:** "Pares encontrados: 0/8"
- **Grade:** 4x4 para os emojis de animais
- **Botão:** "Novo Jogo"para reiniciar

Layout implementado:

- Grid com 3 linhas principais
- UniformGrid para organização dos emojis
- Controles posicionados adequadamente
- Estilo visual consistente

Passo 3: Definir os dados do jogo

Lista de animais:

- **8 pares** de emojis de animais
- **16 posições** na grade 4x4
- **Emojis:** Cachorro, Gato, Rato, Hamster, Coelho, Raposa, Urso, Panda
- **Distribuição:** Cada animal aparece 2 vezes

Estrutura: Arrays, Listas e Dicionários

Passo 4: Definir controles estáticos no XAML

Grade estática 4x4:

- **Grid.RowDefinitions** e **Grid.ColumnDefinitions** (4x4)
- **16 TextBlocks** predefinidos (TextBlock00 a TextBlock33)
- **Evento MouseLeftButtonDown** associado a cada TextBlock
- **Posicionamento** fixo no Grid

Sistema de numeração: [Linha][Coluna]

Grade 4x4 simplificada:

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

Convenção: Primeiro dígito = Linha, Segundo dígito = Coluna

Detalhes da Grade 4x4 - Sistema de Coordenadas

Explicação detalhada da numeração:

Posição	TextBlock	Linha	Coluna	Coordenadas
00	TextBlock00	0	0	(0,0)
01	TextBlock01	0	1	(0,1)
02	TextBlock02	0	2	(0,2)
03	TextBlock03	0	3	(0,3)
10	TextBlock10	1	0	(1,0)
11	TextBlock11	1	1	(1,1)
12	TextBlock12	1	2	(1,2)
13	TextBlock13	1	3	(1,3)
20	TextBlock20	2	0	(2,0)
21	TextBlock21	2	1	(2,1)
22	TextBlock22	2	2	(2,2)
23	TextBlock23	2	3	(2,3)
30	TextBlock30	3	0	(3,0)
31	TextBlock31	3	1	(3,1)
32	TextBlock32	3	2	(3,2)
33	TextBlock33	3	3	(3,3)

Características dos TextBlocks:

- **FontSize:** 24
- **Background:** LightBlue
- **Margin:** 2

Passo 5: Implementar a lógica do jogo

Funcionalidades principais:

- **Seleção de cartas** (máximo 2 por vez)
- **Verificação de pares** (animais iguais)
- **Controle de estado** (primeira/segunda carta)
- **Atualização do status** de pares

Lógica: TextBlock_Click, verificação simples

Passo 6: Gerenciar eventos de clique

Estados das cartas:

- **Oculto:** LightBlue, Text="?"(estado inicial)
- **Revelado:** LightYellow, Text=animal (selecionado)
- **Encontrado:** LightGreen, Text=animal (par completo)

Controle: Verificação simples se Text != "?"

Passo 7: Implementar verificação de pares

Algoritmo de verificação:

- **Primeiro clique:** Revela a carta (primeiraCarta)
- **Segundo clique:** Revela e compara (segundaCarta)
- **Se iguais:** Marca como encontrado (verde)
- **Se diferentes:** Oculta após 1 segundo (timer)

Timer: DispatcherTimer simples com delay de 1 segundo

Passo 8: Adicionar funcionalidades extras

Recursos adicionais:

- **Botão "Novo Jogo"** para reiniciar (RestartButton_Click)
- **Status de pares** encontrados (StatusText)
- **Mensagem de vitória** quando completar (MessageBox)
- **Embaralhamento** das cartas (Random + LINQ)

Melhorias: Interface simples e funcional

Conceitos de C# aplicados

Elementos da linguagem:

- **Arrays e Listas** para armazenar dados
- **Loops** para criar controles dinamicamente
- **Condicionais** para verificar pares
- **Métodos** para organizar o código

Conceitos avançados:

- **Eventos** para interação do usuário
- **Delegates** para callbacks
- **Timers** para controle de tempo
- **Threading** para operações assíncronas

Conceitos de WPF aplicados

Controles utilizados:

- **Grid** para layout principal
- **TextBlock** para grade de cartas
- **TextBlock** para texto e contadores
- **Button** para interações

Recursos do WPF:

- **Event Handling** para interações
- **Layout** responsivo com Grid
- **Properties** para configuração visual
- **Eventos** MouseLeftButtonDown

Conceitos de POO aplicados

Princípios utilizados:

- **Encapsulamento** em classes e métodos
- **Separação de responsabilidades** (UI vs lógica)
- **Reutilização** de código
- **Manutenibilidade** do código

Padrões: Event-driven programming, State management

Estrutura de Dados - Arrays e Listas

Definição dos dados do jogo:

Elementos principais:

- **Array:** animais[] com 8 animais únicos
- **Lista:** cartas com 16 elementos (8 pares)
- **Variáveis de estado:** primeiraCarta, segundaCarta
- **Array 2D:** grade[4,4] para mapear TextBlocks
- **Timer:** timer para controle de tempo

Código implementado:

- **Arquivo:** MainWindow.xaml.cs (linhas 15-25)
- **Array:** private string[] animais
- **Lista:** private List<string> cartas
- **Variáveis:** private TextBlock primeiraCarta, segundaCarta
- **Array:** private TextBlock[,] grade
- **Timer:** private DispatcherTimer timer

Método SetupGame - Configuração Inicial

Configuração de TextBlocks estáticos e embaralhamento:

Funcionalidades implementadas:

- **Inicialização:** InitializeGameTextBlocks() mapeia TextBlocks XAML
- **Array 2D:** TextBlock[,] grade para acesso
- **Lista de cartas:** cartas com duplicatas
- **Embaralhamento:** Random + LINQ OrderBy
- **Distribuição:** Animais nos TextBlocks estáticos
- **Configuração:** Text="?", Tag=animal, Background=LightBlue

Código implementado:

- **Arquivo:** MainWindow.xaml.cs (linhas 27-60)
- **Método:** private void SetupGame()
- **Inicialização:** InitializeGameTextBlocks()
- **Embaralhamento:** cartas.OrderBy(x => random.Next())
- **Distribuição:** grade[row, col].Tag = animal

Método InitializeGameTextBlocks - Mapeamento Estático

Mapeamento de TextBlocks XAML para array:

Características do mapeamento:

- **Array 2D:** TextBlock[,] grade[4,4]
- **Mapeamento:** TextBlock00 → grade[0,0]
- **Acesso:** grade[row, col] para cada posição
- **Configuração:** Todos os 16 TextBlocks mapeados
- **Referência:** Acesso direto aos controles XAML
- **Organização:** Estrutura lógica por linha/coluna

Código implementado:

- **Arquivo:** MainWindow.xaml.cs (linhas 27-45)
- **Método:** private void InitializeGameTextBlocks()
- **Array:** grade = new TextBlock[4, 4]
- **Mapeamento:** grade[0,0] = TextBlock00
- **Organização:** Todos os 16 TextBlocks mapeados

Evento TextBlock_MouseLeftButtonDown - Lógica Principal

Evento principal do jogo:

Lógica implementada:

- **Verificação de estado:** Text != "?"para verificar se já foi revelado
- **Revelação:** Tag.ToString() para mostrar animal
- **Controle de seleção:** primeiraCarta, segundaCarta
- **Verificação de pares:** Comparação de Tag
- **Chamada de métodos:** Verificação simples de igualdade

Código implementado:

- **Arquivo:** MainWindow.xaml.cs (linhas 62-95)
- **Método:** private void TextBlock_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
- **Verificação:** if (clickedTextBlock.Text != "?") return;
- **Revelação:** clickedTextBlock.Text = animal
- **Comparação:** if (primeiraCarta.Tag.ToString() ==

Processamento de Pares - Lógica Simplificada

Processamento quando um par é encontrado:

Ações realizadas:

- **Mudança de cor:** LightGreen para pares encontrados
- **Incremento:** paresEncontrados++ para contador
- **Limpeza:** primeiraCarta = null, segundaCarta = null
- **Verificação de vitória:** paresEncontrados == 8
- **Chamada:** MessageBox.Show() se jogo terminou

Código implementado:

- **Arquivo:** MainWindow.xaml.cs (linhas 85-95)
- **Lógica:** Dentro do evento TextBlock_MouseLeftButtonDown
- **Cores:** primeiraCarta.Background = LightGreen
- **Contador:** paresEncontrados++;
- **Limpeza:** primeiraCarta = null; segundaCarta = null;
- **Vitória:** if (paresEncontrados == 8) MessageBox.Show("Parabéns!");

Timer - Controle de Tempo Simplificado

Timer para esconder cartas não combinadas:

Funcionalidades do timer:

- **Esconder cartas:** Text = "?", Background = LightBlue
- **Limpeza de seleção:** Reset das variáveis primeiraCarta/segundaCarta
- **Controle de estado:** Reset das variáveis de controle
- **Parar timer:** timer.Stop() para finalizar
- **Delay:** 1 segundo para visualização

Código implementado:

- **Arquivo:** MainWindow.xaml.cs (linhas 97-110)
- **Método:** private void Timer_Tick(object sender, EventArgs e)
- **Esconder:** primeiraCarta.Text = "?"; primeiraCarta.Background = LightBlue
- **Limpeza:** primeiraCarta = null; segundaCarta = null;
- **Parar:** timer.Stop();

Interface XAML - MainWindow.xaml

Estrutura da interface gráfica:

Elementos da interface:

- **Window:** Título, dimensões, centralização
- **Grid principal:** 3 linhas (Auto, *, Auto)
- **Título:** TextBlock com nome do jogo
- **Área do jogo:** Grid 4x4 com TextBlocks
- **Status:** TextBlock para mostrar pares encontrados
- **Botão:** Button para novo jogo

Código implementado:

- **Arquivo:** MainWindow.xaml (linhas 1-50)
- **Window:** Title="Jogo de Combinação"
- **Grid:** Grid.RowDefinitions com 3 linhas
- **Título:** TextBlock Text="Jogo de Combinação"
- **Área do jogo:** Grid 4x4 com 16 TextBlocks

Recursos para estudo

Materiais recomendados:

- **Livro:** "Head First C#"4th Edition
- **Repositório:** github.com/head-first-csharp/fourth-edition
- **Documentação:** Microsoft Learn - WPF
- **Vídeos:** Canais especializados em C#

Comunidade e suporte:

- **Stack Overflow** para dúvidas técnicas
- **GitHub** para exemplos de código
- **Reddit** para discussões
- **Discord** para networking

Próximos passos

O que vem depois:

- **Prática:** Implementar o jogo completo
- **Conceitos:** Herança e polimorfismo
- **Unity:** Lab 1 - Introdução ao Unity
- **Projeto:** Aplicação mais complexa

Preparação:

- Revisar conceitos de WPF e XAML
- Praticar com eventos e controles
- Explorar recursos do Visual Studio
- Estudar padrões de design

Resumo da Aula

Pontos principais:

- **Implementação completa** do Animal Matching Game
- **Conceitos práticos** de WPF e C#
- **Lógica de jogo** interativo
- **Programação orientada a eventos**

Conceitos aprendidos:

- Criação dinâmica de controles
- Gerenciamento de estado de aplicação
- Manipulação de eventos complexos
- Separação entre interface e lógica

Obrigado!

Dúvidas? **Próxima aula:** Unity Lab 1 - Introdução ao Unity

Contato:

- **Professor:** Bento Rafael Siqueira
- **Disciplina:** GCT052 - Programação Orientada a Objetos
- **Universidade:** UFLA