APLICAÇÃO DO JOGADOR

ALEXANDRE TOMMASI
SILAS RODRIGUES
GABRIEL PIVOTO
LUIZ HENRIQUE PINA
MATHEUS AUGUSTO DE FARIA

Aplicação do Conceito Arquitetural: SPA

- Interface Amigável:
  - Navegação rápida e sem recarregamentos de página.
  - Design intuitivo e fácil de usar, garantindo boa usabilidade.

INTERFACE DO FRONT FUNCIONAL

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.SceneManagement;
using UnityEngine;

0 references
public class Menu_Manager : MonoBehaviour
{
    1 reference
    [SerializeField] private string nome;
    5 references
    [SerializeField] private GameObject painel_menu;
    3 references
    [SerializeField] private GameObject painel_rank;
    3 references
    [SerializeField] private GameObject painel_historico;

    0 references
    public void Start(){
        painel_rank.SetActive(false);
        painel_historico.SetActive(false);
        painel_menu.SetActive(true);
    }

    0 references
    public void Historico(){
        painel_historico.SetActive(true);
        painel_menu.SetActive(false);
    }
    0 references
    public void Jogar(){
        SceneManager.LoadScene(nome);
    }
    0 references
    public void Rank(){
        painel_rank.SetActive(true);
        painel_menu.SetActive(false);
    }
```

Aplicação do Conceito Arquitetural: SPA

- Pontuação e Ranking:
  - Atualização da pontuação em tempo real, sem delays.
  - Exibição clara e contínua do ranking, mantendo o jogador sempre informado sobre sua posição.

```
3 references
private int posicao;
3 references
private string nome;
3 references
private int pontuacao;

0 references
void Start()
{
    posicao = 0;
    nome = "Player";
    pontuacao = 0;

    AtualizarRank();
}

2 references
public void AtualizarRank()
{
    Posicao.text = posicao.ToString();
    Nome.text = nome;
    Pontuacao.text = pontuacao.ToString();
}

0 references
public void ConfigurarRank(int novaPosicao, string novoNome, int novaPontuacao)
{
    posicao = novaPosicao;
    nome = novoNome;
    pontuacao = novaPontuacao;

    AtualizarRank();
}
```

Aplicação do Conceito Arquitetural: SPA

- Histórico de Partidas:
  - Carregamento dinâmico dos dados, sem necessidade de recarregar a página.
  - Acesso rápido e interativo ao histórico de partidas, com dados atualizados em tempo real.

```
0 references
public class Historico : MonoBehaviour
{
    1 reference
    public TMP_Text Placar_1;
    1 reference
    public TMP_Text Placar_2;
    1 reference
    public TMP_Text Oponente;
    // Start is called before the first frame update
    0 references
    void Start()
    {
        Placar_1.text = "x";
        Placar_2.text = "y";
        Oponente.text = "" + "vs.";
    }
```

Design Patterns Escolhidos

Criação:

- Singleton

Estrutural:

- Adapter
- Composite

Comportamental:

- Observer
- Command

Criação

Singleton

- Uma instância única: Garante que Partida ou Menu tenham apenas uma instância ativa.
- Consistência no estado: Evita conflitos e garante que o estado seja consistente.
- Ponto único de acesso: Facilita o gerenciamento da instância de forma eficiente.

```
public static Menu Instance
{
    get
    {
        if (_instance == null)
        {
            Debug.Log("Silas: Criando instância única do Menu");
            _instance = FindObjectOfType<Menu>();
            if (_instance == null)
            {
                GameObject obj = new GameObject("Menu");
                _instance = obj.AddComponent<Menu>();
            }
        }
        return _instance;
    }
}

private void Awake()
{
    if (_instance != null && _instance != this)
    {
        Destroy(gameObject);
        Debug.Log("Silas: Instância duplicada do Menu destruída.");
        return;
    }
    _instance = this;
    DontDestroyOnLoad(gameObject); // Mantém a instância entre cenas
    Debug.Log("Silas: Menu configurado como Singleton.");
}
```

Estrutural

Adapter

- Adapta o formato de dados: Converte o JSON do backend para o formato necessário no frontend.
- Criação das cartas: Organiza os dados para exibição das cartas.
- Integração facilitada: Permite a integração do “backend” com o frontend sem impacto estrutural.

```json
{
    "card":{
        "foto": "string",
        "descricao": "string",
        "tipo":"string",
        "poder":"integer"
    }
}
```

```
public class Card
{
    public string Foto { get; set; }
    public string Descricao { get; set; }
    public string Tipo { get; set; }
    public int Poder { get; set; }

    public Card(string foto, string descricao, string tipo, int poder)
    {
        Foto = foto;
        Descricao = descricao;
        Tipo = tipo;
        Poder = poder;
    }

    public override string ToString()
    {
        return $"Foto: {Foto}, Descricao: {Descricao}, Tipo: {Tipo}, Poder: {Poder}";
    }
}
```

Estrutural

Composite

- Gerenciamento de menus e submenus: Organiza e estrutura menus de forma simples e intuitiva.
- Estrutura hierárquica: Organiza objetos em árvores, representando as hierarquias dos itens.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.SceneManagement;
using UnityEngine;

0 references
public class Menu_Manager : MonoBehaviour
{
    1 reference
    [SerializeField] private string nome;
    5 references
    [SerializeField] private GameObject painel_menu;
    3 references
    [SerializeField] private GameObject painel_rank;
    3 references
    [SerializeField] private GameObject painel_historico;

    0 references
    public void Start(){
        painel_rank.SetActive(false);
        painel_historico.SetActive(false);
        painel_menu.SetActive(true);
    }

    0 references
    public void Historico(){
        painel_historico.SetActive(true);
        painel_menu.SetActive(false);
    }
    0 references
    public void Jogar(){
        SceneManager.LoadScene(nome);
    }
    0 references
    public void Rank(){
        painel_rank.SetActive(true);
        painel_menu.SetActive(false);
    }
```

Comportamental

Observer

- Uso na classe Partida: Notifica mudanças como o fim da partida ou atualização de pontuação.
- Notificação para Jogadores: As instâncias de Jogador são notificadas quando um evento importante ocorre.

```
public void RegistrarObservador(IObservador observador)
    {
        observadores.Add(observador);
        Debug.Log("Silas: Observador registrado.");
    }

    public void RemoverObservador(IObservador observador)
    {
        observadores.Remove(observador);
        Debug.Log("Silas: Observador removido.");
    }

    private void NotificarObservadores(string evento)
    {
        foreach (IObservador observador in observadores)
        {
            observador.Atualizar(evento);
        }
    }

    public void FimDePartida()
    {
        Debug.Log("Silas: Partida finalizada.");
        NotificarObservadores("Fim da partida");
    }

    public void AtualizarPontuacao(int placarP1, int placarP2)
    {
        Debug.Log($"Silas: Placar atualizado. P1: {placarP1}, P2: {placarP2}");
        NotificarObservadores($"Atualização de placar - P1: {placarP1}, P2: {placarP2}");
    }
```

```
public interface IObservador

void Atualizar(string evento);



public void Atualizar(string evento)
{
    Debug.Log($"Silas: Jogador {Nome} recebeu notificação - {evento}");
}
```

Comportamental

Command

- Uso na classe Menu: Encapsula ações como iniciarPartida() e solicitarHistorico().
- Facilidade de expansão: Permite adicionar novas ações no menu sem modificar sua estrutura.

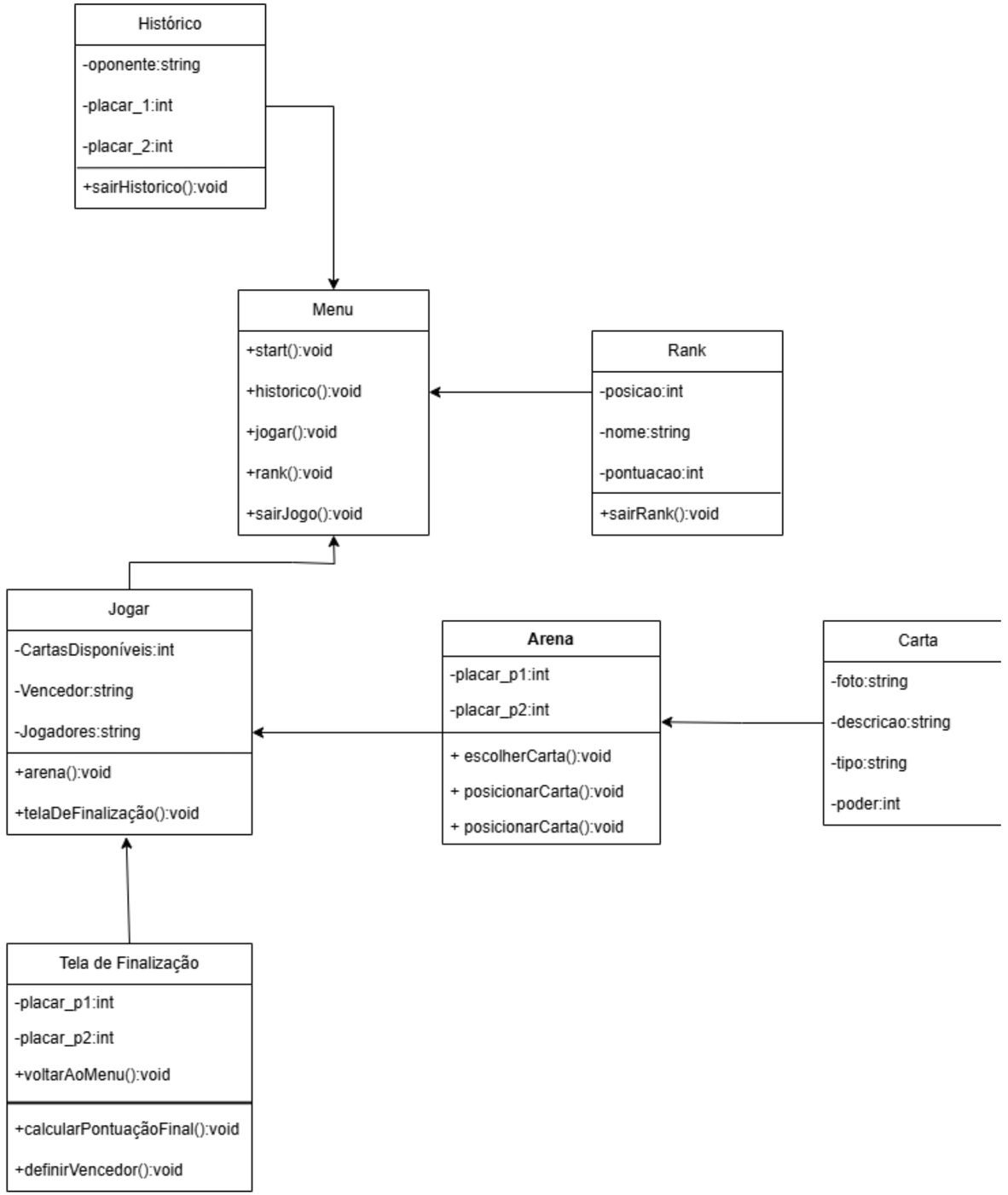
```
0 references
public void Start(){
    painel_rank.SetActive(false);
    painel_historico.SetActive(false);
    painel_menu.SetActive(true);
}

0 references
public void Historico(){
    painel_historico.SetActive(true);
    painel_menu.SetActive(false);
}
0 references
public void Jogar(){
    SceneManager.LoadScene(nome);
}
0 references
public void Rank(){
    painel_rank.SetActive(true);
    painel_menu.SetActive(false);
}
```

```
0 references
public void Sairhistorico(){
    painel_historico.SetActive(false);
    painel_menu.SetActive(true);
}
```

colo

UML de Classes



Funcionamento da Proposta

MUITO OBRIGADO!