

# Trabalho Prático

## Fase 2 - UI

LESI PL

Programação  
Orientada a Objetos

Docente: Ernesto Carlos Casanova Ferreira

Autor: Nuno Rodrigo Rebelo Sá Silva, 28005

Data: 21/12/2024

Repositório: [Github](#)

## Índice

|  |    |
|--|----|
| Objetivos.....                             | 3  |
| Estutura da Solução.....                   | 3  |
| Singleton (RPG/Classes/SQL.cs).....        | 4  |
| Menu (RPGUI/menu/Menu.cs).....             | 5  |
| Scoreboard (RPGUI/menu/Scoreboard.cs)..... | 6  |
| Login (RPGUI/menu/Login.cs).....           | 7  |
| Login Form .....                           | 10 |
| SignUp (RPGUI/menu/SignUp.cs).....         | 12 |
| Team (RPGUI/menu/Team.cs).....             | 14 |
| Conclusão.....                             | 23 |

## Índice de Imagens

|   |    |
|---|----|
| Figura 1 Solution.....                    | 3  |
| Figura 2 Menu, no Logged in players ..... | 5  |
| Figura 3 Menu Login Player1 .....         | 8  |
| Figura 4 Menu Ready to Start Game .....   | 9  |
| Figura 5 Login Forms.....                 | 11 |
| Figura 6 SignUp Form.....                 | 12 |
| Figura 7 Team Selection .....             | 14 |
| Figura 8 MVC Model .....                  | 16 |
| Figura 9 Battle View.....                 | 20 |

## Objetivos

O objetivo deste projeto é implementar um sistema UI para um jogo RPG com a utilização de Objetos, este sistema foi feito em Windows Forms (net8.0-windows).

O formato deste sistema tem 2 partes:

1. Menu, Login, SignUp, Scoreboard e Team: Todos este Windows Forms têm a sua lógica contida neles mesmo, toda ela excluindo lógica de base de dados SQL;
2. Sistema de batalha: Realizado em Windows Forms de acordo com o formato MVC (Model, View, Controller), estes Forms encontram-se em projetos separados: BattleModel, BattleView, BattleController).

Este relatório tem como objetivo documentar a utilização dos objetos e a interface gráfica projetadas para a solução deste projeto.

## Estutura da Solução

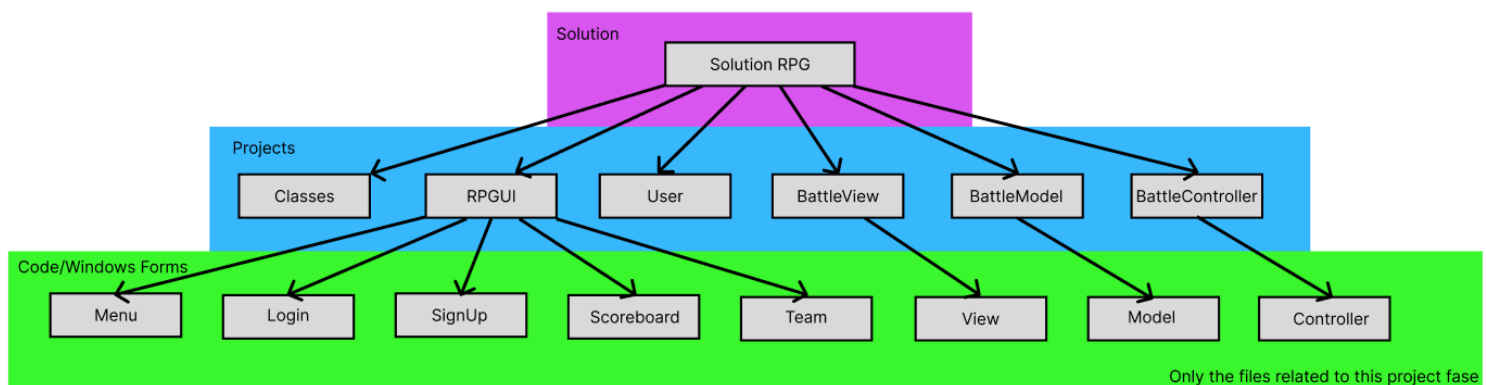


Figura 1 Solution

## Singleton (RPG/Classes/SQL.cs)

No projeto, foi implementado o padrão de design Singleton na classe SQL para garantir que apenas uma instância desta classe seja criada e utilizada durante a execução da aplicação.

A classe SQL possui duas propriedades principais:

1. `_instance` (do tipo SQL): É uma propriedade estática que armazena a instância única da classe SQL.
2. `Connection` (do tipo string): Serve para guardar a string de conexão com a base de dados, contendo o Data Source.

O padrão Singleton é implementado de forma que, ao chamar a instância da classe SQL pela primeira vez, um novo objeto é criado e armazenado em `_instance`. Nas chamadas subsequentes, a mesma instância armazenada é retornada. Desta forma, evita-se a criação de múltiplos objetos da classe SQL, garantindo que apenas uma instância seja utilizada em todo o projeto.

```
private static SQL _instance;
private string connection { get; set; }

private SQL()
{
    //desktop
    this.connection = "Server=DESKTOP-2J6JLCD\\SQLEXPRESS;Database=RPG;User
Id=rpg_admin;Password=1234;Encrypt=True;TrustServerCertificate=True;";

    //laptop
    //this.connection = "Server=...\\SQLEXPRESS02;Database=RPG;User
Id=rpg_admin;Password=1234;Encrypt=True;TrustServerCertificate=True;";
}
public static SQL Instance
{
    get
    {
        if (_instance == null)
        {
            _instance = new SQL(); // Lazy initialization
        }
        return _instance;
    }
}
```

## Menu (RPGUI/menu/Menu.cs)

Menu é a página inicial do projeto onde são apresentadas 3 botões:

1. Login1: botão presente no lado esquerdo da tela para aparecer a janela de login para o jogador 1;
2. Login2: botão presente no lado direito da tela para aparecer a janela de login para o jogador 2;
3. Scoreboard: botão presente no meio da tela que cria uma pequena página com uma tabela dos jogadores e as suas estatísticas (numero de vitórias e partidas).

O estado inicial do menu contém apenas os botões indicados previamente com o display de equipas escondidos:

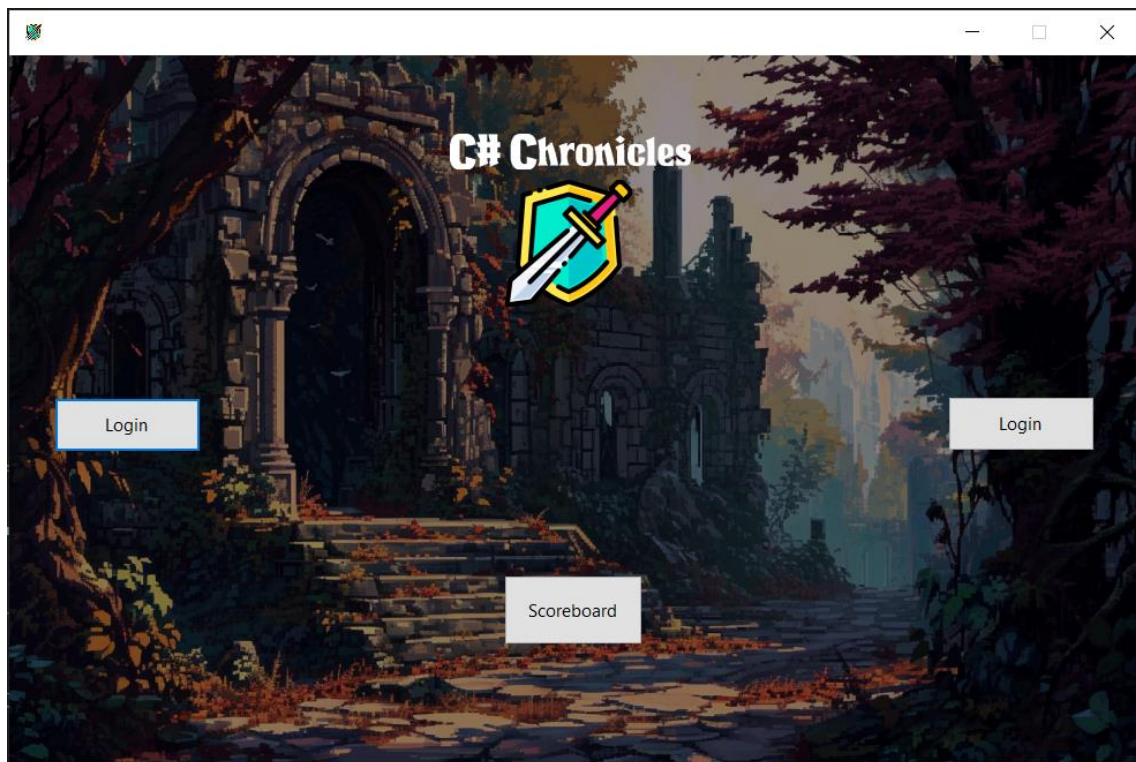


Figura 2 Menu, no Logged in players

Para passar o menu para um próximo estado existem 2 opções:

- Login
- Scoreboard

## Scoreboard (RPGUI/menu/Scoreboard.cs)

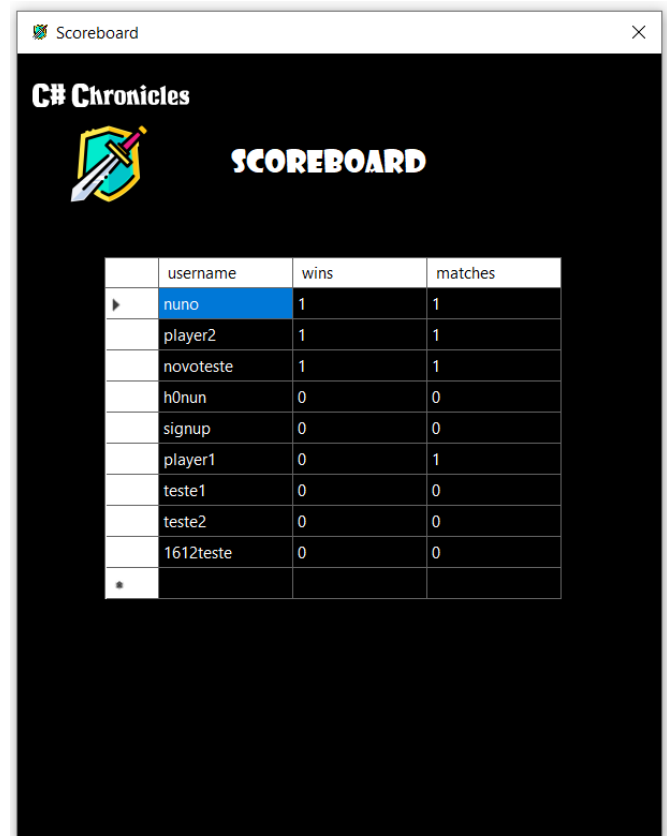
A Scoreboard é uma tabela simples dos dados carregados da base de dados para uma dataGridView que é chamada através do menu em forma de “popup” com a seguinte representação:

Visto que a tabela é feita com dados diretamente da base de dados é utilizado o Singleton SQL que return a dataGridView já preenchida e apenas a coloca dentro da dataGridView presente no Windows Forms Scoreboard.

- Representação em código do uso de singleton e processo de inicialização/carregamento de dados para o Forms

```
public Scoreboard()
{
    InitializeComponent();
    LoadScoreboard();
}
private void LoadScoreboard()
{
    try
    {
        SQL db_connection = SQL.Instance;
        // Get the scoreboard data from the SQL class
        DataTable scoreboardData = db_connection.GetScoreboard();

        dataGridView1.DataSource = scoreboardData;
        Controls.Add(dataGridView1);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}");
    }
}
```



## Login (RPGUI/menu/Login.cs)

A lógica de Login está toda realizada dentro do Menu.cs na função `LogInOut(object sender, EventArgs e, int i)`.

Esta função é chamada sempre que o botão de login 1 ou 2 é pressionado, mas dependendo do lado o "int i" pode alternar entre 1 e 2 representando o objeto User1 e User2 da seguinte forma:

```
button1.Click += (sender, e) => LogInOut(sender, e, 1);
button2.Click += (sender, e) => LogInOut(sender, e, 2);
```

Estes botões após ser realizado o Login com sucesso servem para Logout e o seu texto é alterado de acordo.

A utilização deste `inteiro` serve para verificações tal como:

1. Se o User do respetivo número é null, isto serve para saber se é para ser realizado um Login ou um Logout:
  - 1.1. se for null: é realizado o login;
  - 1.2. se não for null: o display no menu com o nome do utilizador e a equipa do mesmo são escondidas e o User passa a ter um valor null.
2. Durante o Login também é verificado se o outro User não é o mesmo que está a tentar fazer login novamente:
  - 2.1. Condição se `i==1`, se estiver a ser realizado o Login do User1

```
if (user2 != null && user2.loggedIn && login.user.username == user2.username)
{
    MessageBox.Show("This user is already logged in.");
    this.Show();
    return;
}
```

2.2. Condição se `i==2`, se estiver a ser realizado o Login do User2

```
if (user1 != null && user1.loggedIn && login.user.username == user1.username)
{
    MessageBox.Show("This user is already logged in.");
    this.Show();
    return;
}
```



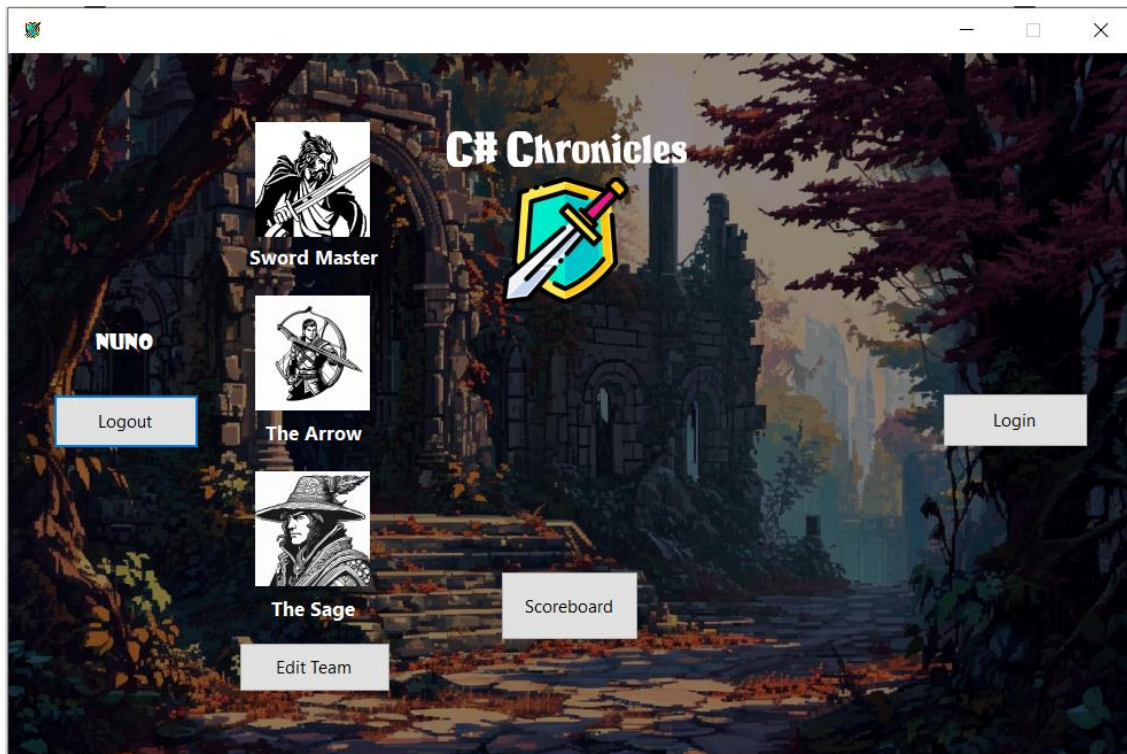


Figura 3 Menu Login Player1

Após cada Login e Logout também é verificado se ambos User1 e User2 estão presentes ao qual é finalmente visível o botão para começar o jogo.

Esta função também serve para esconder o botão em caso de Logout da seguinte forma:

- Fim da função LogInOut():

```
// Check if both users are logged in before showing a start game button
buttonStart();
```

- Na função buttonStart():

```
private void buttonStart()
{
    // Check if both users are logged in
    if (user1 != null && user2 != null)
    {
        StartButton.Visible = true; // Show "Start Game" button
        StartButton.Enabled = true; // Makes the button "usable"
    }
    else
    {
        StartButton.Visible = false;
        StartButton.Enabled = false;
    }
}
```



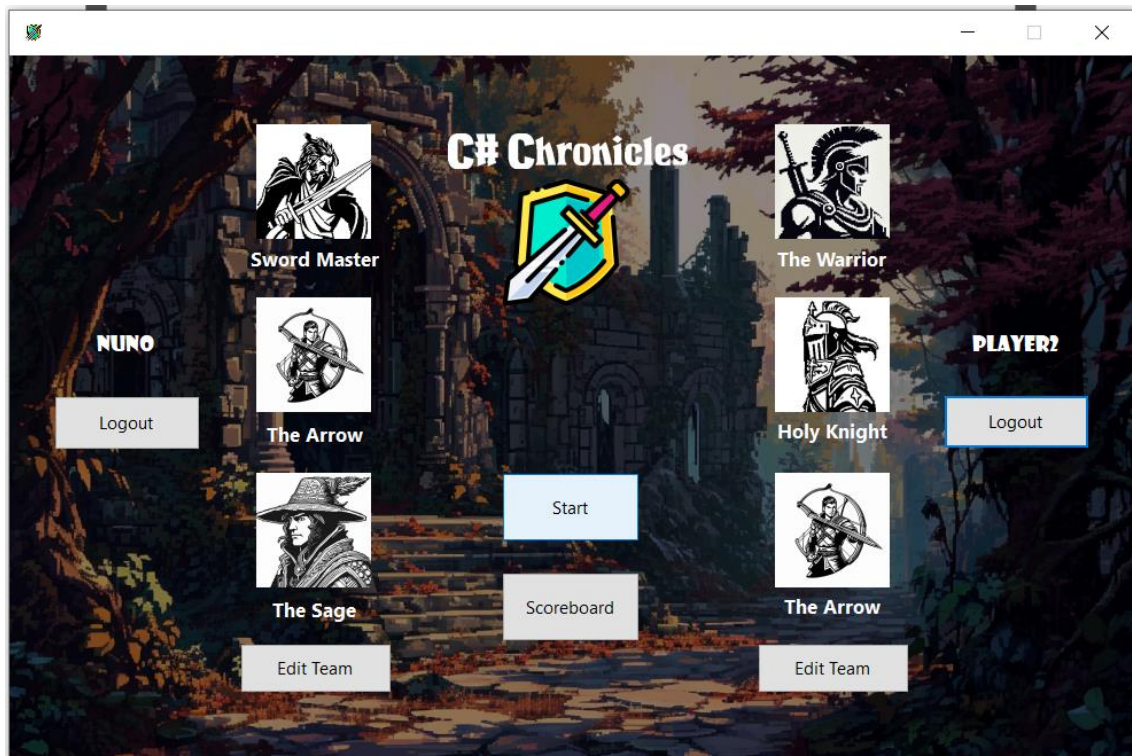


Figura 4 Menu Ready to Start Game

## Login Form

O Windows Forms de Login contém o seu próprio objeto do tipo User e as suas verificações quanto ao preenchimento do mesmo, estas verificações incluem:

- Campo “username” tem que estar preenchido;
- Campo “password” também tem de estar preenchido;
- O conjunto dos campos têm que pertencer a algum utilizadores na base de dados.

Método com as verificações mencionadas:

```
private void login(object sender, EventArgs e)
{
    try
    {
        string? username = textBoxUser.Text;
        string? passwd = textBoxPass.Text;
        if (string.IsNullOrEmpty(username))
        {
            throw new Exception("Make sure to fill the username.");
        }
        if (string.IsNullOrEmpty(passwd))
        {
            throw new Exception("Make sure to fill the password.");
        }
        if (database.CheckLogin(username, passwd))
        {
            // Login successful - proceed to the next step
            int[] data = database.UserData(username);
            this.User = new User(username, data[0], data[1]);
            Close();
        }
        else
        {
            throw new Exception("Password or Username incorrect.");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

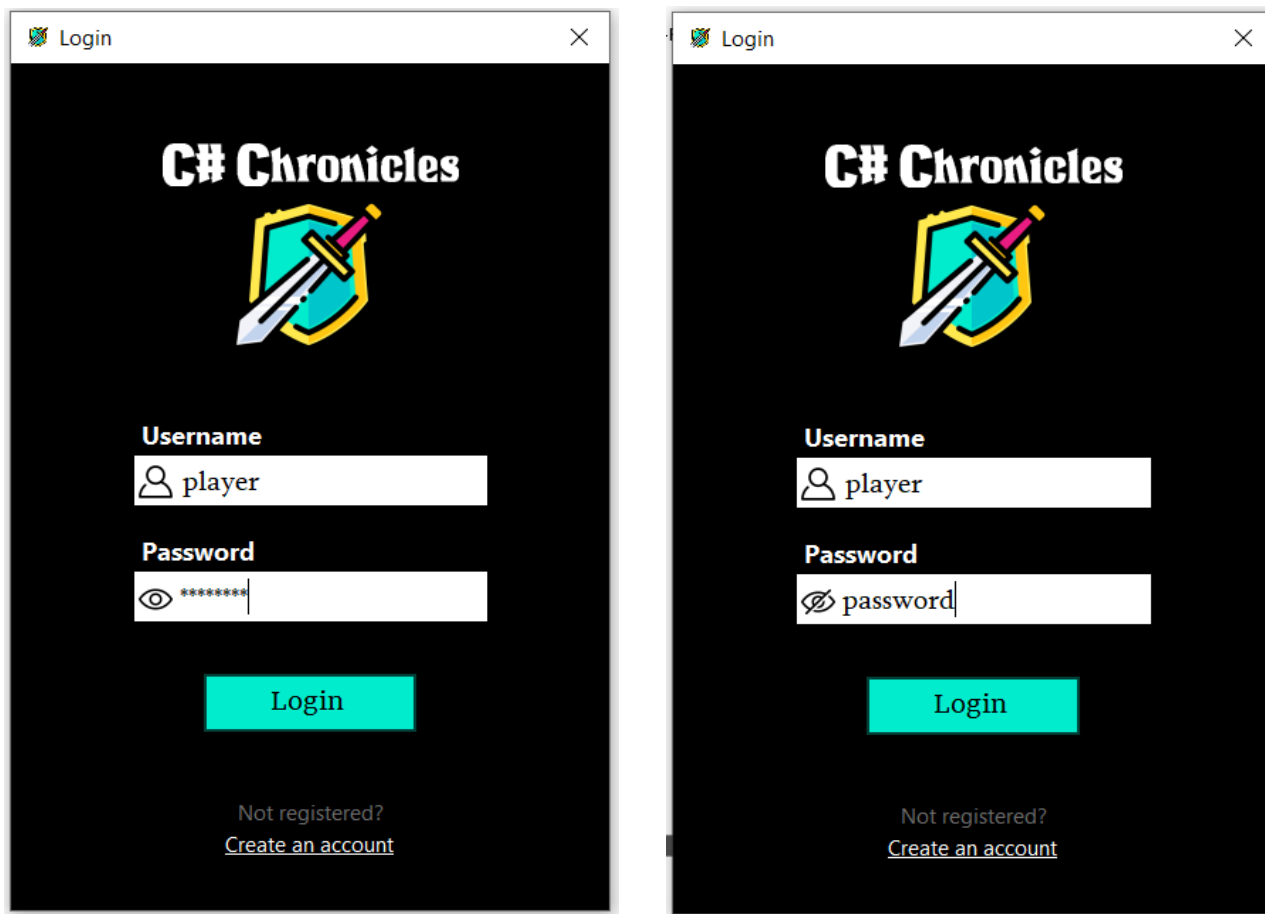


Figura 5 Login Forms

Através desta página é possível criar uma nova conta ao clicar na LinkLabel presente na parte de baixo da página é chamada a função:

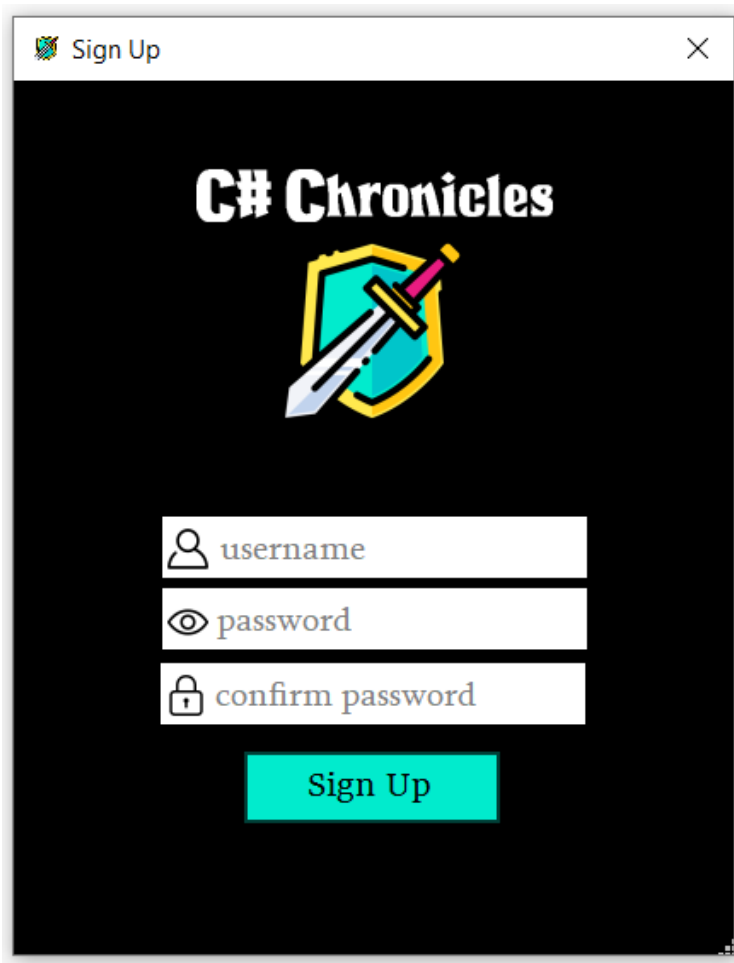
```
private void sign(object sender, EventArgs e)
```

Onde o Windows Forms SignUp é inicializado.

## SignUp (RPGUI/menu/SignUp.cs)

O Windows Forms de SignUp contém as suas verificações quanto ao preenchimento do mesmo, estas verificações incluem

- Campo “username” tem que estar preenchido;
- Campo “password” e “confirmar password” também tem de estar preenchido;
- Campo “password” e “confirmar password” têm de ser iguais.



The image shows a Windows Forms application window titled "Sign Up" with a close button (X) in the top right corner. The background is black. At the top center, the text "C# Chronicles" is displayed in a white, stylized font. Below the text is a logo featuring a yellow shield with a blue border, and a sword with a blue blade and a yellow hilt. Below the logo are three input fields, each with a white background and a black border. The first field has a person icon and the placeholder text "username". The second field has an eye icon and the placeholder text "password". The third field has a lock icon and the placeholder text "confirm password". Below the input fields is a red rectangular button with the text "Sign Up" in white. The bottom right corner of the window shows a small system tray icon.

Figura 6 SignUp Form

Quando pressionado o botão “Sign Up” a função `createAccount()` é chamada, a mesma faz as verificações mencionadas anteriormente e se todas as condições necessárias se reúnem então é chamada a inserção na base de dados.

### Verificações:

```
// Check if the username and password fields are empty
if (string.IsNullOrEmpty(textBoxUser.Text) || string.IsNullOrEmpty(textBoxPass.Text) ||
string.IsNullOrEmpty(textBoxConfirm.Text))
{
    throw new ArgumentNullException("Make sure to fill the username and password boxes.");
}
// Check if the passwords match
else if (textBoxPass.Text != textBoxConfirm.Text)
{
    throw new Exception("Make sure the passwords match");
}
```

Quando os dados inseridos estão de acordo com as verificações mencionadas passamos para o seguinte segmento da função, base de dados.

Na classe SQL é chamada a função `CreateAcc()` que retorna um boolean, (verdadeiro quando a inserção é concluída com sucesso, falso se houve falha na inserção), verifica se o username inserido já pertence a um jogador, se este não for o caso os dados são inseridos na base de dados e voltamos para a página de Login onde já podem ser usados os dados que foram inseridos inicialmente.

### Verificação de inserção com **try...catch**:

```
try
{
    //Verificações mencionadas anteriormente
}
catch (ArgumentNullException ex)
{
    MessageBox.Show(ex.Message, "Input Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);
}
catch (ArgumentException ex)
{
    MessageBox.Show(ex.Message, "Username Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

## Team (RPGUI/menu/Team.cs)

Ao finalizar com sucesso o Login é chamada a função `public bool TeamLoad(string username, out BattleTeams team)`, que return a equipa e um boolean indicando se a equipa existe na base de dados e foi possível carregar a mesma ou se não foi possível.

Esta verificação tem como finalidade saber se é um novo utilizador que ainda não tem um equipa seleciona “team preset” ou se é um utilizador que já selecionou uma equipa.

Se o utilizador acabou de realizar o Login pela primeira vez o Windows Forms Team vai ser inicializado para que o utilizador escolha a equipa, se foi possível carregar a equipa o Forms não é iniciado e a equipa é carregada para o Menu.

A unica outra forma de chegar a este Forms seria através do Menu, por baixo da equipa existe um botão “Edit Team” que abre o Forms e permite editar/escolher uma nova equipa.

Dentro da Windows Forms Team temos uma interface simples com todos os 6 personagens e checkBoxs com o nome dos personagens. Dentro da mesma Form temos algumas indicações que nos dizem que podemos selecionar exatamente 3 personagens.

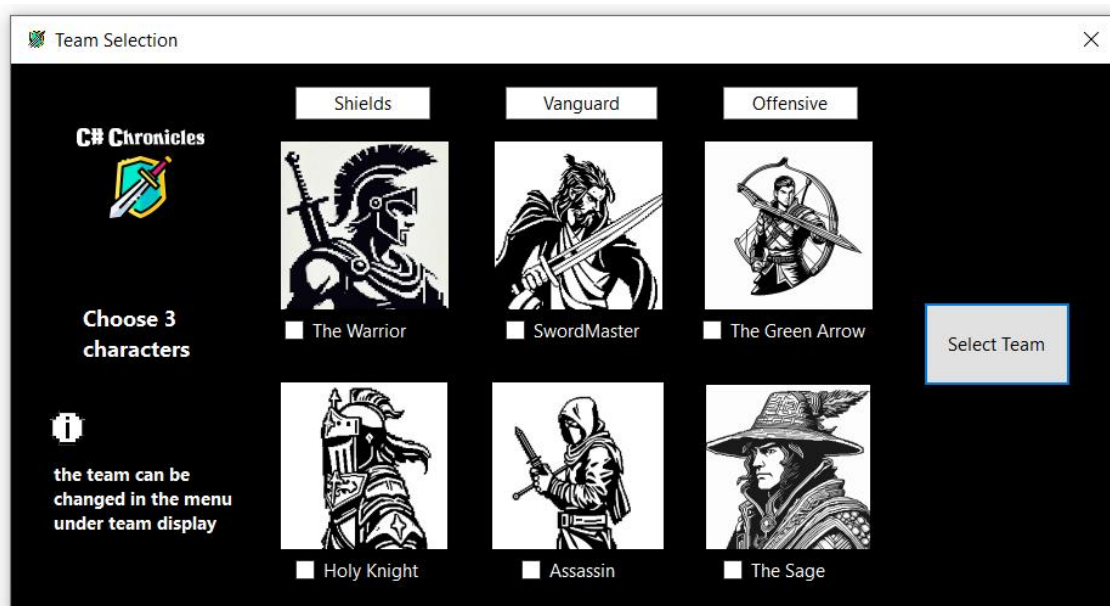


Figura 7 Team Selection

Neste Forms existem algumas verificações tais como:

- Máximo de 3 checkboxes com estado “Checked”, através de um contador;
- Se tentar selecionar menos de 3 personagens, através do mesmo contador.

A verificação de “Checked” checkboxes é realizada na função:

```
private void CheckBox_CheckedChanged(object sender, EventArgs e)
{
    // Get the currently checked checkbox count
    int checkedCount = 0;
    CheckBox[] checkboxes = { checkBox1, checkBox2, checkBox3, checkBox4, checkBox5, checkBox6 };

    foreach (var checkbox in checkboxes)
    {
        if (checkbox.Checked)
        {
            checkedCount++;
        }
    }

    // If more than MaxTeamSize checkboxes are checked, uncheck the last one
    if (checkedCount > MaxTeamSize)
    {
        // Uncheck the checkbox that triggered the event if the limit is exceeded
        CheckBox currentCheckbox = sender as CheckBox;
        if (currentCheckbox != null)
        {
            currentCheckbox.Checked = false;
            MessageBox.Show($"You can only select up to {MaxTeamSize} characters.");
        }
    }
}
```

Observação:

Este Form contém a propriedade `private const int MaxTeamSize = 3`; utilizada na função anterior.

Após o preenchimento correto do Forms os objectos das classes são criados e adicionados à equipa onde depois é verificado se a quantidade de objetos dentro da lista que constitui a equipa é igual ao `MaxTeamSize`.

Se esta verificação for verdadeira a equipa é carregada para o menu, se não os objetos são eliminados e voltamos para o Form de seleção de equipa.



## Sistema de Batalha MVC

O sistema de batalha é representado por 3 projetos diferentes:

1. BattleModel
2. BattleView
3. BattleController

Estes projetos representam a o formato MVC.

O **MVC** (Model-View-Controller) é um padrão de design arquitetural utilizado no desenvolvimento de software, especialmente em aplicações com interface gráfica. Ele separa a aplicação em três componentes principais, cada um com uma responsabilidade distinta:

1. **Model (Modelo):**  
Representa os dados e a lógica de negócio da aplicação. É responsável por gerenciar e manipular os dados, interagir com a base de dados e notificar as outras camadas (como a View) sobre alterações.
2. **View (Vista):**  
É a interface com o utilizador. Apresenta os dados ao utilizador e recebe as interações dele. A View é responsável apenas pela exibição, sem incluir lógica de negócio.
3. **Controller (Controlador):**  
Atua como intermediário entre o Model e a View. Recebe as entradas do utilizador através da View, processa essas entradas, chama os métodos apropriados no Model e determina qual View deve ser exibida.

A principal vantagem do MVC é a **separação de responsabilidades**, o que facilita a manutenção, teste e escalabilidade da aplicação. Além disso, permite que diferentes equipas trabalhem simultaneamente em cada componente.

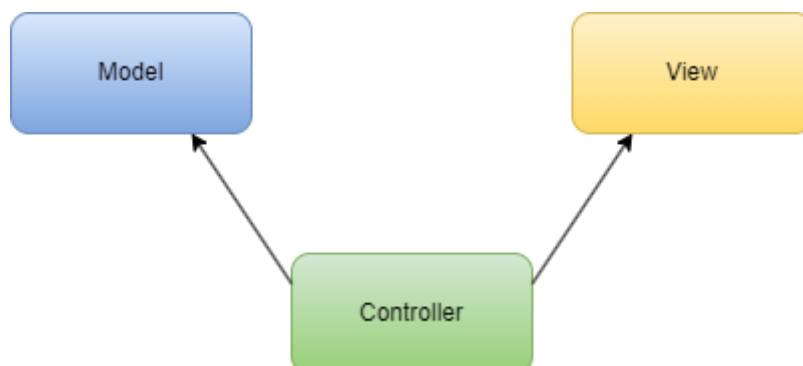


Figura 8 MVC Model

## Model (BattleModel/Model.cs)

Durante a batalha todos os calculas de dano, e construção de logs de batalha são feitos dentro do Model, este atualiza diretamente os objetos que após serem alterados são enviados pelo Controller para o View para que a interface seja atualizada.

É no Model também que é realizada a verificação de estado de jogo, como a verificação se todos os membros de uma equipa estão vivos, se uma das equipas não tiver nenhum personagem vivo o jogo termina.

```
public bool IsGameOver()
{
    if (team1.All(c => !c.alive) || team2.All(c => !c.alive))
    {
        GetWinner();
        return true;
    }
    return false;
}

public void GetWinner()
{
    if (team1.All(c => !c.alive)) this.victor = user2;
    else this.victor = user1;
}
```

Observação: alive é uma função boolean da superclasse Class, que retorna o valor desta condição

```
public bool alive
{
    get { return this.HP > 0; }
}
```

Também é no modelo que a lógica de jogo é feita, passado a vez de jogar de um jogador para o outro no fim de cada turno e verifica se o jogo terminou.

```
public bool EndTurn()
{
    currentTurn = (currentTurn == 1) ? 2 : 1; // Alternate between 1 and 2
    ResetSelected();
    ResetDefense();
    return IsGameOver();
}
```

Para todas as ações, excepto Block(), temos um jogo de chances a imitar um lançamento de dados na vida real que calcula quanto do dano calculado é realmente infligido no adversário.

```
public static int diceRoll(int min, int max)
{
    Random rand = new Random();
    int roll = rand.Next(min, max);
    return roll;
}
public static int CritResistance(int damage, int roll)
{
    double mult = roll / 10.0;
    return (int)(damage * mult);
}
```

O calculo do dano é realizado com base nas estatísticas dos personagens, este calculo é feito nas funções abstratas de cada personagem.

Algumas ações requerem pontos de poder que é recarregável através de ações básicas como Attack() e Block(), as ações que requerem este poder passam por uma verificação que é se a quantidade de poder que eles têm é suficiente.

```
public bool CanUse(int team1Index, int team2Index, int cost)
{
    GetSelected(team1Index, team2Index);
    if (this.attacker.GetGauge() >= cost)
    {
        return true;
    }
    return false;
}
```

Para além disto ainda temos que no inicio de cada ronda a estatística de defesa de cada personagem volta ao seu estado original, isto porque ao usar a função Block() existe um aumento dessa mesma estatística.

```
private void ResetDefense()
{
    List<Class> Team = (currentTurn == 1) ? team1 : team2;
    foreach (Class c in Team)
    {
        c.defense = c.originalDefense;
    }
}
```

O Controller cada vez que diz ao Model para realizar uma ação ele envia 2 indexes, o index do personagem da equipa 1 e o da equipa 2, que com base no calculo de turnos coloca os personagens indicados nos seus respetivos papeis, attacker e defender.

```
private void GetSelected(int team1Index, int team2Index)
{
    List<Class> currentTeam = (currentTurn == 1) ? team1 : team2; //if true team 1 else team 2
    List<Class> enemyTeam = (currentTurn == 1) ? team2 : team1;
    if (currentTurn == 1)
    {
        this.attacker = currentTeam[team1Index];
        this.attackerTeam = user1;
        //block action can be perfomed with only the attacker selected
        if (team2Index != -1)
        {
            this.defender = enemyTeam[team2Index];
            this.defenderTeam = user2;
        }
    }
    else
    {
        this.attacker = currentTeam[team2Index];
        this.attackerTeam = user2;
        //block action can be perfomed with only the attacker selected
        if (team1Index != -1)
        {
            this.defender = enemyTeam[team1Index];
            this.defenderTeam = user1;
        }
    }
}
```

E no fim de cada turno são removidos para que não fiquem restos do ultimo turno.

```
private void ResetSelected()
{
    this.attacker = null;
    this.defender = null;
}
```

No fim de todas as ações temos também a contrução dos logs de batalha que posteriormente no Controller serem adicionados à lista de strings no View, e têm todos um formato parecido com o seguinte:

```
//log making
this.dice = this.attackerTeam + " rolled: " + roll;
this.log = this.attackerTeam + "s " + this.attacker.name + " performed Basic Attack on " + this.defenderTeam +
"s " + this.defender.name + " and did " + damage + " damage!";
```

## View (BattleView/View.cs)

Devido a problema de referencia, para invocar o Controller no View foram utilizados neste projeto eventos:

```
public event Action UpdateUI;
public event Action AttackButton;
public event Action SpecialButton;
public event Action UltimateButton;
public event Action BlockButton;
public event Action CloseGame;
```

Estes eventos são invocados quando um botão é selecionado, notificando o Controller que consequentemente chama o Model para realizar operações de dados nos objetos.

Este projeto tem apenas lógica de UI como updates aos valores de poder dos personagens, o seu estado (vivo ou morto), os seus pontos de vida e os logs de batalha.

Estes updates são ativados pelo Controller no fim de todas as rondas, enviado todos os dados necessários para o View.

A única lógica presente no View seria a seleção de personagens, index do personagem da equipa 1 e 2, esta lógica é feita ao clicar na imagem do personagem, e para confirmar a seleção aparece uma seta branca no lado da imagem, é possível alternar os personagens uma vez que selecionados bastando clicar em um outro.



Figura 9 Battle View

## Controller (BattleController/Controller.cs)

O Controller é criado no Menu ao pressionar o botão “Start”, isto envia todos os dados necessário para começar a batalha: o nome de utilizador e as equipas selecionadas.

```
public Controller(List<Class> player1Team, List<Class> player2Team, string user1, string user2)
{
    this.model = new Model(player1Team, player2Team, user1, user2);
    this.view = new BattleView.View();

    //view events
    this.view.AttackButton += HandleAttack;
    this.view.SpecialButton += HandleSpecial;
    this.view.UltimateButton += HandleUltimate;
    this.view.BlockButton += HandleBlock;
    this.view.UpdateUI += Update;
    this.view.CloseGame += Close;
}
```

Na criação do Controller é também criado o Model e View já com os dados suficientes para começar o jogo.

No menu ainda antes de mostrar o View é realizada a função StartBattle() do Controller que atualiza todos os dados no UI de forma a estarmos prontos para começar o jogo:

```
public void StartBattle()
{
    this.view.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
    this.view.UpdatePic(this.model.Team1, this.model.Team2);
    this.view.UpdateText(this.model.username1, this.model.username2, this.model.Team1, this.model.Team2);
    this.view.roundCount = 0;
    this.model.DefenderTeam = this.model.username1;
    Update();
    this.view.ShowDialog();
}
```

No fim de todas as rondas é atualizado o UI com a função Update() também utilizada no início do jogo.

```
private void Update()
{
    this.view.UpdateValues(this.model.Team1, this.model.Team2, this.model.DefenderTeam);
    this.view.team1 = this.model.Team1;
    this.view.team2 = this.model.Team2;
}
```

Durante o jogo o Controller perfoma todas as ações de forma semelhante:

1. Perfoma ação no Model;
2. Após realizar ações adiciona à lista de logs do View os novos logs do Model;
3. Reinicia os indexs das equipas do View;
4. E de seguida verificamos se o jogo terminou:
  - 4.1. Se sim obtemos um vencedor e atualizamos a base de dados, incrementando o campo “wins” e “matches” de cada jogador;
  - 4.2. Se não atualizamos o UI com a função Update e continua o jogo.

```
private void HandleUltimate()
{
    if (this.model.CanUse(this.view.player1, this.view.player2, 50))
    {
        int damage = model.PerformUltimate(this.view.player1, this.view.player2);
        this.view.battleLog.Add(this.model.dice);
        this.view.battleLog.Add(this.model.log);
        //reset selected
        this.view.player1 = 0;
        this.view.player2 = 0;
        if (this.model.EndTurn())
        {
            Winner();
        }
        else
        {
            Update();
        }
    }
    else
    {
        throw new Exception("To use the Ultimate ability you need a full gauge!");
    }
}
```

No fim de jogo a função Winner() é chamada que chama as queries para incrementar as estatísticas dos utilizadores.

E no fim fechamos a janela limpando o MVC por completo com a função close, que também é chamada no botão de fechar do View.

```
private void Close()
{
    this.view.Close();
    this.view = null;
    this.model = null;
    GC.Collect();
}
```



## Conclusão

Este projeto demonstrou a aplicação prática de conceitos fundamentais de desenvolvimento de software, integrando programação orientada a objetos e o padrão arquitetural MVC. A implementação de um jogo RPG de turnos com personagens representados como objetos.

O uso do padrão MVC no sistema de batalha foi particularmente eficaz na separação de responsabilidades, permitindo que a lógica do combate, a interface com o utilizador e o fluxo de interação fossem desenvolvidos e ajustados de forma independente. Esta abordagem contribuiu para a clareza e organização do código, facilitando futuras manutenções e expansões do sistema.

Em suma, o projeto alcançou os objetivos propostos, demonstrando o valor de boas práticas de desenvolvimento e destacando a importância de padrões como MVC.