

”

UNIDADE CURRICULAR: Programação por Objetos

CÓDIGO: 21093

DOCENTES: Jorge Morais e Leonel Morgado (professores) e José Félix Póvoa e Rúdi Gualter (tutores)

A preencher pelo estudante

NOME: Luís Carlos Crispim Pereira

N.º DE ESTUDANTE: 2300163

CURSO: LEI – Licenciatura em Engenharia Informática

DATA DE ENTREGA: 17/12/24

TRABALHO / RESOLUÇÃO:

O meu projeto para a unidade curricular de Programação por Objetos consiste num jogo de cartas Blackjack (21) desenvolvido em Python, utilizando o IDE PyCharm. O projeto foi dividido em três fases, conforme especificado pelo docente da unidade. Na primeira fase (Efolio A), foi criado apenas o esqueleto do jogo, com a definição das classes e da estrutura básica.

Na fase Efolio B, o desenvolvimento do jogo foi avançado, tornando-o mais completo e funcional. A organização do código foi significativamente aprimorada, com a separação das classes em ficheiros distintos (.py), o que favoreceu a modularidade e a clareza. As classes principais, como Card, Deck, Player, entre outras, foram isoladas, o que facilitou a reutilização e a manutenção do código a longo prazo. Foi implementado um menu interativo, permitindo ao utilizador escolher entre as opções de jogar, visualizar as classificações, configurar o número de jogadores e o nível de dificuldade, além de sair do jogo. A solicitação do nome do jogador foi incluída logo no início, proporcionando uma interação mais personalizada com o utilizador. No que diz respeito às regras do jogo, as cartas foram ajustadas para serem exibidas corretamente, com a carta do dealer sendo mantida virada para baixo até o momento em que o dealer revela sua mão. As restantes regras do jogo foram implementadas de forma a garantir que o fluxo de jogo ocorra conforme o esperado onde cada jogador tem a opção de pedir mais cartas, ficar com a mão atual ou desistir. O dealer segue regras automáticas para jogar, com base na dificuldade selecionada. A lógica para determinar o vencedor foi também concluída, levando em consideração as diferentes condições possíveis: vitória, derrota, empate e a verificação de Blackjack. Além disso, o sistema de classificações foi implementado e integrado ao jogo, permitindo o acompanhamento da performance dos jogadores.

A fase Efolio B assegurou que o jogo estivesse funcional e pronto para a próxima etapa, que consistirá na integração de uma interface gráfica na fase Efolio Global, utilizando o Tkinter. O projeto, neste ponto, está estável, com todas as funcionalidades principais em funcionamento, e pronto para ser aprimorado com a interface gráfica.

1. Implementação Técnica:

Classe Card - Representa uma carta individual com um valor e um naipe.

- Atributos:
 - value – Valor da carta (ex. 2,3,4,5,...).
 - suit – Naipe da carta (ex. Ouros, espadas, ...).
- Métodos:
 - __str__ – Retorna a carta como string.
 - get_value – Retorna o valor da carta, considerando o valor das figuras (Rei, Dama, Valete) e do Ás, que pode ser 1 ou 11.

Classe Deck - Representa o baralho completo, incluindo lógica para embaralhar e distribuir cartas.

- Atributos:
 - cards – Lista todas as cartas representadas no baralho.
- Métodos:
 - build_deck – Cria o baralho com as 52 cartas.
 - shuffle_deck – Baralha o baralho.
 - deal_card – Dá as cartas aos Player conforme as regras.

Classe Player - Representa um jogador, que pode ser o utilizador ou o dealer.

- Atributos:
 - name – Nome do jogador.
 - hand – Cartas na mão do jogador.
 - score – Pontuação atual do jogador
 - stayed – Indica se o jogador decidiu "ficar" (não pedir mais cartas). É um valor booleano, inicialmente definido como False.
 - folded – Indica se o jogador desistiu da ronda. Também é um valor booleano, inicialmente definido como False.
 - victory_percentage – Percentagem de vitórias do jogador, inicialmente definida como 0.0. Este atributo é utilizado para avaliar o desempenho do jogador nas partidas.
- Métodos:
 - add_card – Acrescenta uma carta a mão.
 - calculate_score – Calcula a pontuação atual do jogador.
 - show_hand(reveal_dealer=False): Mostra as cartas na mão do jogador e Dealer (ocultando a primeira).
 - clear_hand(): Limpa a mão do jogador, ou seja, remove todas as cartas da lista hand.
 - calculate_hand_value(reveal_dealer=False): Calcula o valor da mão do jogador e dealer com base nas cartas visíveis.
 - take_action(): Solicita ao jogador uma ação durante o seu turno.
 - has_busted(): Verifica se o jogador estourou, ou seja, se o valor da sua mão ultrapassou 21.
 - has_stayed(): Verifica se o jogador escolheu a ação de "ficar".
 - has_folded(): Verifica se o jogador desistiu da ronda.
 - has_blackjack(): Verifica se o jogador tem uma mão de Blackjack.
- Herança:
 - JogadorHumano: Subclasse da classe Player que representa o jogador humano. Substitui o método take_action para permitir que o jogador humano escolha uma ação de forma interativa.
 - Dealer: Subclasse da classe Player que representa o dealer. Substitui o método take_action para implementar a lógica automatizada do dealer.

Classe Game - Controla a lógica do jogo.

- Atributos:
 - deck – Baralho para jogo. Objeto da classe deck.
 - **player** – O jogador.
 - **players** – Lista dos jogadores humanos.
 - dealer – O dealer (adversário). Objeto da classe dealer.
 - **actions** – Armazena as ações possíveis durante os turnos dos jogadores.
 - **turno** – Inteiro que indica o número atual do turno.
 - **game_over** – Booleano que indica se o jogo acabou.
 - **classificações** – Permite atualizar o histórico de classificações. Objeto da classe **classificações**.
- Métodos:
 - **__init__()** – Inicializa todos os atributos necessários para a classe.
 - **restart_game()** – Reinicia o jogo, limpando as mãos e criando novo baralho.
 - **start_game** – Inicia o jogo.
 - **player_turn** – Controla turno jogador.
 - **dealer_turn** – Controla turno dealer.
 - **check_winner** – Verifica vencedor.
 - **exibir_resultados()** - Exibe os resultados após jogo terminado e atualiza classificações.

Classe Leaderboard - Controla a classificação de todos os jogos.

Classe Classificacoes - Gere a classificação de todos os jogadores.

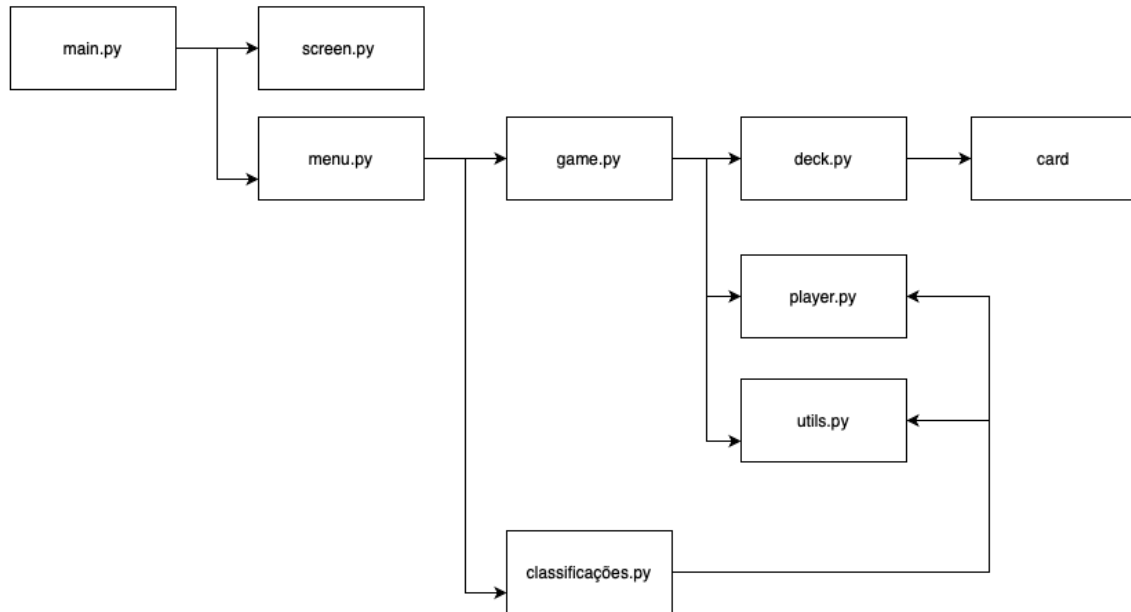
- Atributos:
 - **file** – Caminho do ficheiro onde as classificações são armazenadas.
 - **classificacoes** – Dicionário que guarda os dados das classificações dos jogadores.
- Métodos:
 - **__init__()** – Inicializa todos os atributos necessários para a classe.
 - **carregar_classificacoes()** – Carrega as classificações a partir do ficheiro JSON.
 - **guardar_classificacoes()** – Guarda as classificações no ficheiro JSON.
 - **atualizar_classificacoes(nome_jogador, venceu)** – Atualiza as estatísticas de um jogador com base no resultado do jogo.
 - **calcular_percentagem_media()** – Calcula a percentagem média de vitórias de todos os jogadores registados.
 - **mostrar_classificacoes()** – Exibe o Top 10 de jogadores ordenados pela percentagem de vitórias e compara com a média geral.

A preto é o que estava aplicado em efolioA.

A vermelho é o que estava pensado aplicar no efolioB mas não foi aplicado ou foi substituído.

A azul é o que foi implementado em efolioB.

Diagrama de dependências:



2. Reflexão e Alternativas:

Abordagem à Persistência de Dados no Projeto de Blackjack

No desenvolvimento do jogo de Blackjack, a persistência de dados é um aspeto crucial para garantir que as informações dos jogadores, como as classificações e estatísticas, sejam armazenadas e recuperadas. No código atual, a persistência é implementada através do uso de um ficheiro JSON para armazenar as classificações. No entanto, existem diversas abordagens alternativas para melhorar ou até substituir esta solução. Duas delas são a utilização de uma base de dados relacional (SQLite) ou uso de um formato de ficheiro CSV. Abaixo, são apresentadas as vantagens e desvantagens de cada uma dessas abordagens.

Persistência com Base de Dados Relacional (SQLite) - A utilização de uma base de dados relacional como o SQLite pode ser uma solução interessante, principalmente se o jogo crescer em termos de número de jogadores e classificações. O SQLite é uma base de dados leve e local, que não requer um servidor separado para funcionar, o que a torna ideal para projetos como o jogo de Blackjack.

- Vantagens:
 - Escalabilidade: A base de dados é mais eficiente para armazenar e manipular grandes volumes de dados, tornando-a uma boa escolha à medida que o número de jogadores e jogos aumenta.
 - Consultas complexas: Utilizar SQL permite realizar consultas mais avançadas, como ordenações, filtros e agregações, facilitando a análise das classificações.
 - Estruturação eficiente: A base de dados permite armazenar os dados de forma estruturada e organizada, minimizando o risco de erros e tornando o acesso aos dados mais rápido e seguro.

- Desvantagens:
 - Complexidade de implementação: A configuração e manipulação de uma base de dados requer um pouco mais de código e conhecimento, o que pode aumentar a complexidade do projeto.
 - Sobrecarga de código: Além de criar e gerir tabelas, é necessário tratar a comunicação com a base de dados, como INSERT, UPDATE e SELECT, o que pode tornar o código mais verboso e difícil de manter.

Persistência com Ficheiro CSV - Uma outra alternativa mais simples é utilizar o formato CSV, que armazena os dados em formato de texto simples, facilmente legível e manipulável. Esta abordagem é adequada para jogos menores, onde a quantidade de dados não será muito grande e a simplicidade é mais importante.

- Vantagens:
 - Simplicidade: O formato CSV é fácil de ler e escrever, tanto programaticamente quanto por ferramentas externas como o Excel, o que facilita a visualização e análise dos dados.
 - Compatibilidade: Qualquer aplicação ou ferramenta que lide com dados tabulares (como folhas de cálculo) pode abrir ficheiros CSV, tornando os dados facilmente acessíveis.
 - Baixa sobrecarga: A implementação é simples, sem a necessidade de bibliotecas ou configurações complexas, o que facilita a manutenção do código.
- Desvantagens:
 - Limitações para grandes volumes de dados: Embora o CSV seja uma boa solução para volumes reduzidos de dados, não é tão eficiente quanto uma base de dados relacional quando o número de jogadores e a quantidade de dados aumentam.
 - Falta de consultas complexas: O CSV não permite a execução de consultas avançadas, como filtros e ordenações complexas, tornando a análise dos dados mais limitada.

Persistência via Ficheiro JSON (Atualmente Implementado) - O JSON é uma escolha prática para o projeto, pois permite armazenar dados em um formato estruturado e legível. A vantagem é a facilidade de implementação, o que permite guardar as classificações dos jogadores com a percentagem de vitórias, número de mãos jogadas e ganhas. A estrutura hierárquica do JSON é adequada para este tipo de projeto, pois facilita a leitura e a escrita dos dados sem exigir uma complexidade adicional no código.

- Vantagens:
 - Simplicidade na implementação e manutenção.
 - Fácil leitura e escrita dos dados.
 - Compatível com várias linguagens e ferramentas de desenvolvimento.
- Desvantagens:
 - Não é tão eficiente para grandes volumes de dados ou consultas complexas.
 - Pode tornar-se menos flexível conforme o projeto cresce.

Após analisar as alternativas, a solução adotada foi o uso de um ficheiro JSON devido à sua simplicidade, estruturação eficiente e flexibilidade para o âmbito atual do projeto. Embora o CSV fosse uma alternativa mais simples, não satisfaz bem as necessidades de estruturação de dados mais complexos. A base de dados SQLite foi descartada devido à sua complexidade de implementação, que não se justificava para o armazenamento relativamente simples das classificações dos jogadores. O uso de JSON oferece um equilíbrio entre simplicidade, flexibilidade e facilidade de manutenção, sendo a escolha que achei mais adequada para o projeto de Blackjack.

3. Preparação para a Fase Final:

A fase final do projeto (efolioGlobal) envolve a implementação de uma interface gráfica com Tkinter, que vai transformar a interação do jogo de textual para visual. O código atual está bem estruturado e modularizado, o que facilita a transição para essa nova fase. As classes e métodos que já estão definidos (como Game, Player, Deck e Classificações) fornecem uma boa base para integrar a interface gráfica. No entanto, como ainda não tenho experiência com Tkinter, não sei se conseguirei implementar a interface gráfica por completo. Mesmo com a estrutura atual, algumas modificações serão necessárias para adaptar o jogo ao novo formato.

A maior mudança será a forma como o utilizador interage com o jogo. Atualmente, as interações acontecem via terminal, com escolhas por texto. No Tkinter, isso exigirá a criação de botões e menus, o que pode ser um desafio, especialmente para quem ainda não está familiarizado com a biblioteca. A forma de exibir informações, como as cartas e as classificações, também precisará ser adaptada para uma janela gráfica. Embora a lógica principal do jogo, como a distribuição das cartas e o cálculo das classificações, deva permanecer a mesma, será necessário ajustá-la para garantir que a interface gráfica reflita corretamente o andamento do jogo. Este ajuste envolve enviar as informações do jogo para a interface gráfica de forma eficiente. A interface gráfica pode aumentar a complexidade do código, especialmente em jogos com múltiplos jogadores. Vou precisar garantir que o desempenho não seja comprometido, com animações e atualizações visuais eficientes, pelo que se o desempenho for comprometido, o modo multijogador pode ser removido para garantir essa mesma fluidez. As classificações atuais, exibidas no terminal, precisarão ser apresentadas graficamente, o que exigirá ajustar a forma como essas informações são processadas e mostradas.

Embora o código atual esteja bem estruturado, a maior limitação para a fase final é a falta de experiência com Tkinter, o que pode dificultar a implementação da interface gráfica de forma fluida e completa. As alterações mais significativas envolvem transformar a interação textual para uma interface gráfica e garantir que os eventos sejam tratados adequadamente. Isso exigirá ajustes na forma como o código está organizado e em como as funções interagem com o utilizador.

Nota: Criado 2 Easter Eggs no jogo um com a dificuldade impossível onde o Dealer tira sempre 21 e outro com o nome de jogador Kaxeszer que tira sempre 21.