

”

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: 22/11/24

TRABALHO / RESOLUÇÃO:

O meu Projeto para a unidade curricular de Programação por objetos é um jogo de cartas Blackjack (21) feito em linguagem python com a utilização do IDE PyCharm. Escolhi o jogo de Blackjack por ser um desafio prático que envolve lógica, interatividade e uma aplicação direta de conceitos de PPO, como encapsulamento e modularidade. O projeto será dividido em três fases conforme especificado pelo docente da unidade. Nesta primeira fase (Efolio A), apenas será feito um esqueleto do jogo em si, na segunda fase (Efolio B) já o mesmo estará completo e funcional e numa terceira fase (Efolio Global) será implementado o ambiente gráfico com utilização de Tkinter.

Inicialmente o jogo iniciará num menu (ainda não implementado) com as opções a serem tomadas pelo utilizador (para já esta pensado jogar, leaderboard, sair) mas como o projeto é sequencial poderá ter mais funcionalidades adicionadas em Efolio B e Efolio G.

Nesta primeira fase (Efolio A), apenas criei as classes, atributos e métodos sem olhar ao pormenor as regras do jogo em si, apenas um esqueleto do que será no futuro. Utilizei como referência o jogo de cartas sueca da AF no fórum da UC, tendo aproveitado trechos de código que me facilitaram a implementação das classes. O ficheiro anexado a este relatório (efolioA.py) é um ficheiro em linguagem python com a implementação do que aqui foi falado, o mesmo não apresenta erros embora a única coisa que faz é a criação de um deck (52 cartas), distribui duas pelo dealer e duas pelo Player demonstrando o resultado de duas cartas em ambas as mãos (conforme prints). No EfolioB começarei por colocar cada classe no seu próprio ficheiro .py para melhor organização, clareza, reutilizações futuras e modularização, implementarei a solicitação do nome do Player, as regras de visualização de cartas (sendo que do dealer não se vê uma, até ao momento final), implementarei o menu e garantirei o funcionamento esperado de um jogo de Blackjack. Já no EfolioGlobal irei implementar o ambiente gráfico em Tkinter.

1. Quais são as classes que criou para o projeto? Explique a função de cada classe e o motivo de ter optado por essa estrutura.

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.

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

- Atributos:
 - cards – Lista todas as card 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
- Métodos
 - add_card – Acrescenta uma carta a mão.
 - calculate_score – Calcula a pontuação atual do jogador .
 - show_hand – Mostra a mão do jogador.

Classe Game - Controla a lógica do jogo.

- Atributos:
 - deck – Baralho para jogo.
 - player – O jogador.
 - dealer – O dealer (adversário).
- Métodos
 - start_game – Inicia o jogo.
 - player_turn – Controla turno jogador.
 - dealer_turn – Controla turno dealer.
 - check_winner – Verifica vencedor.

Classe Leaderboard - Controla a classificação de todos os jogos.

A vermelho é o que está pensado mas ainda não implementado.

A estrutura escolhida para as classes foi baseada em boas práticas de programação orientada para objetos, que facilitam a organização, reutilização e manutenção do código. Cada classe foi criada para cumprir um papel específico dentro do jogo, refletindo conceitos fundamentais de PPO como responsabilidade única, encapsulamento e modularidade.

2. Proponha uma resposta alternativa à anterior, explicando porque preferiu uma em relação a outra.

Uma abordagem alternativa seria as classes card e deck serem apenas uma classe, nessa abordagem o deck teria os atributos de naipe e valor associado. Optei por ser em duas classes pois assim será mais fácil de implementar alterações caso queira adicionar mais tarde outras funcionalidades ao deck ou cartas.

Uma alternativa seria usar herança entre Player e Dealer, contudo, preferi manter ambas como instâncias da mesma classe para simplificar a implementação inicial. Isso torna o código mais direto e elimina a necessidade de lidar com comportamentos específicos de subclasses nesta fase do projeto, sendo possível revisitar esta opção nos próximos efolios.

Poderia ter usado também o conceito de sobrecarga mas não encontrei aplicabilidade direta com o meu projeto, sendo assim também optei pela não inclusão.

3. Como decidiu a distribuição de responsabilidades entre as classes?

Utilizei o princípio da responsabilidade única, em que cada classe apenas lida com as responsabilidades da sua classe:

Card – lida com a representação de carta individual

Deck – lida com o baralho de cartas e as operações do baralho.

Player – lida com os jogadores (dealer e jogador) e com as operações envolvendo jogadores.

Game – lida com toda a mecânica do jogo, combinando funcionalidades com as outras classes.

Por exemplo, a classe Game coordena as interações entre Deck e Player, enquanto Deck é responsável apenas pela criação e manipulação das cartas. Esta abordagem modular reflete boas práticas de PPO, permitindo que cada classe tenha uma responsabilidade bem definida.

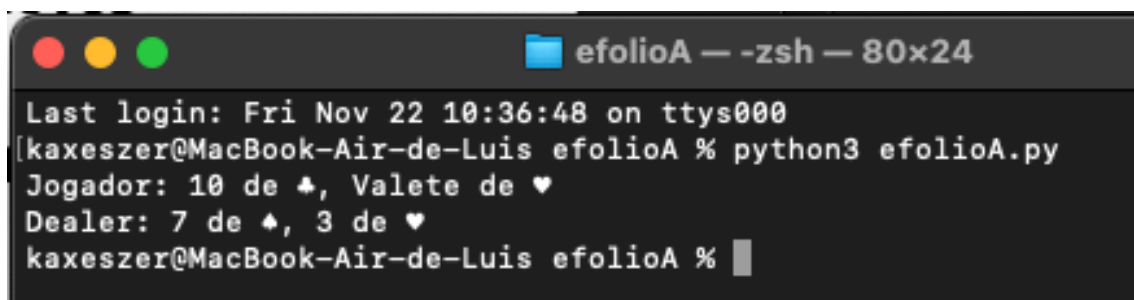
4. O seu projeto pode evoluir para versões futuras com novas funcionalidades ou uma interface melhorada. Quais modificações seriam necessárias no design inicial para incluir essas variantes no futuro?

Sim o projeto pode evoluir para versões futuras com interface melhorado aplicando o Tkinter. Visto o projeto ter construção modular (pelo menos teórica), será de fácil implementação funcionalidades futuras como multijogador, dificuldades extra, etc..

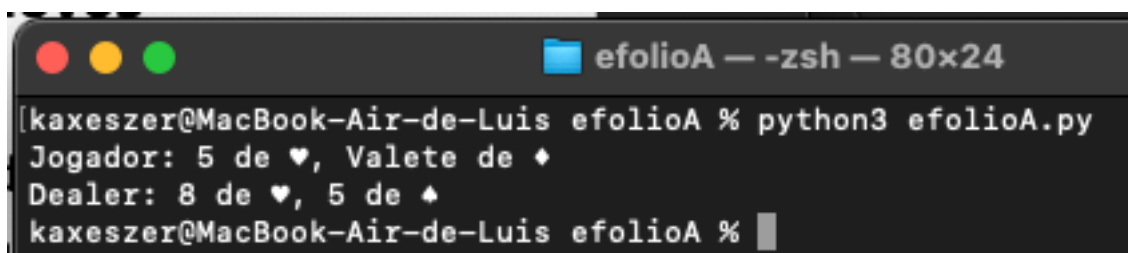
5. Como poderia estruturar o código de forma que seja fácil adicionar novos elementos sem causar grandes modificações no código existente?

Para ser fácil de adicionar elementos será fundamental a utilização de modularidade (já discutida anteriormente e a ser implementada em EfolioB), pois facilita a leitura, a possibilidade de testes de unidade e a implementação de funcionalidades mais direcionada (apenas modificando o módulo com alterações).

Prints EfolioA



```
efolioA — -zsh — 80x24
Last login: Fri Nov 22 10:36:48 on ttys000
[kaxeszer@MacBook-Air-de-Luis efolioA % python3 efolioA.py
Jogador: 10 de ♠, Valete de ♥
Dealer: 7 de ♠, 3 de ♥
kaxeszer@MacBook-Air-de-Luis efolioA %
```



```
efolioA — -zsh — 80x24
[kaxeszer@MacBook-Air-de-Luis efolioA % python3 efolioA.py
Jogador: 5 de ♥, Valete de ♠
Dealer: 8 de ♥, 5 de ♠
kaxeszer@MacBook-Air-de-Luis efolioA %
```