Uge 2: Blackjack

Afleveringsbeskrivelse

I denne opgave skal du udvikle et tekstbaseret Blackjack-spil, Afleveringen skal bestå af:

- En kravspecifikation skal afleveres inden kodningen af spillet påbegyndes.
- En fungerende implementering af Blackjack, der overholder de beskrevne betingelser.
- Kildekode i et GitHub repository, gerne dokumenteret med kommentarer.
- En kort README-fil med en beskrivelse af projektet, herunder:
    - En oversigt over funktionaliteten.
    - Instruktioner til at køre programmet.
- En 10 minutters præsentation, der f.eks. kunne indeholde følgende punkter:
    - En kort introduktion til spillet og dets funktionalitet.
    - Demonstration af implementeringen (evt. screenshots eller live-demo).
    - Forklaring af centrale kodeelementer og valg af teknologi.
    - Refleksion over kravspecifikationen og om den var realistisk
    - Refleksion over tilgangen til opgaven og udførelsen.
    - Overvejelser om mulige forbedringer og udvidelser.

Derudover kan en bonusopgave inkluderes, hvor der tilføjes ekstra funktioner såsom statistik over spil, simuleringer af spillere eller håndtering af flere end to spillere.

Case beskrivelse

Udvikl et Blackjack-spil, der kan spilles enten i en kommandolinje eller en grafisk brugergrænseflade. Spillet følger de klassiske regler for Blackjack, hvor spilleren konkurrerer mod dealeren. Spilleren starter spillet og får to kort, mens dealeren viser ét synligt kort. Spilleren kan herefter vælge at trække et ekstra kort (*hit*) eller beholde sin nuværende hånd (*stand*). I bestemmer selv om I vil håndtere kun en spiller, eller op til 4 spillere som er standarden.

Regler

- Målet er at få en håndværdi tættere på 21 end dealeren uden at overskride 21.
- Kort 2-10 har deres pålydende værdi, billedkort (J, Q, K) giver 10 point, og et es (A) kan være 1 eller 11, afhængigt af hvad der er bedst for hånden.
- Hvis spilleren overstiger 21 (*bust*), taber de øjeblikkeligt.
- Når spilleren står, trækker dealeren kort, indtil hånden har en værdi på mindst 17.
- Hvis dealeren overstiger 21, vinder spilleren.
- Hvis ingen går bust, vinder den med den højeste værdi. Uafgjort resulterer i en *push* (ingen vinder).

For det fulde regelsæt, se under ressourcer.

Eksempel på opgave:

1. **Delopgave 1:** Implementér klasser, lister eller if/else-statements til at lave kort og kortbunke. Kortbunken skal kunne blandes og dele kort ud (Evt. Gennem import af random-modulet).
2. **Delopgave 2:** Lav en funktion der kan udregne værdien af spilleren og dealerens hånd (her kan også bruges klasser eller if/else-statements
3. **Delopgave 3:** Implementér logik der initialiserer spillet og skifter mellem henholdsvis spilleren og dealerens tur
4. **Delopgave 4:** Tilføj muligheden for brugerinput, så spilleren kan vælge “hit” eller “stand” (her kan eksempelvis bruges et while-loop)
5. **Delopgave 5:** Lav en funktion afgør om spilleren eller dealeren er vinder
6. **Delopgave 6 (valgfri):** Tilføj eventuelt en grafisk brugergrænseflade, eller lav andre tilføjelser til spillet som du finder relevant eller interessante
7. **Delopgave 7 (valgfri):** Tilføj eventuelt mulighed for at kunne spille med flere mennesker, eller simple bot’s man kan spile med

Ressourcer:

- https://www.pokerstars.dk/casino/how-to-play/blackjack/rules/
- https://www.wikihow.com/Play-Blackjack

Kickstart-code:

```
import random


class Card:
    def __init__(self, suit, value):
        self.suit = suit
        self.value = value

    def __repr__(self):
        return f"{self.value} of {self.suit}"


class Deck:
    def __init__(self):
        suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
        values = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q',
'K', 'A']
        self.cards = [Card(suit, value) for suit in suits for value in
values]
        random.shuffle(self.cards)

    def deal_card(self):
        return self.cards.pop()


class Hand:
    def __init__(self):
        self.cards = []
        self.value = 0

    def add_card(self, card):
        self.cards.append(card)
        self.calculate_value()

    def calculate_value(self):
        self.value = 0
        has_ace = False
        for card in self.cards:
            if card.value.isnumeric():
                self.value += int(card.value)
            elif card.value in ['J', 'Q', 'K']:
                self.value += 10
            elif card.value == 'A':
                has_ace = True
                self.value += 11

        if has_ace and self.value > 21:
            self.value -= 10
```