7. Metoda Monte Carlo

Zadání:

Metoda Monte Carlo představuje rodinu metod a filosofický přístup k modelování jevů, který využívá vzorkování prostoru (například prostor čísel na herní kostce, které mohou padnout) pomocí pseudonáhodného generátoru čísel. Jelikož se jedná spíše o filosofii řešení problému, tak využití je téměř neomezené. Na hodinách jste viděli několik aplikací (optimalizace portfolia aktiv, řešení Monty Hall problému, integrace funkce, aj.). Nalezněte nějaký zajímavý problém, který nebyl na hodině řešen, a získejte o jeho řešení informace pomocí metody Monte Carlo. Můžete využít kódy ze sešitu z hodin, ale kontext úlohy se musí lišit.

Řešení:

Rozhodla jsem se metodu Monte Carlo využít při výpočtu pravděpodobnosti líznutí různých výherních kombinací v Pokeru. Kód spočítá na základě daného počtu iterací, jaká je pravděpodobnost líznutí všech jednotlivých výherních kombinací při prvním líznutí prvních pěti karet ze zamíchaného balíčku.

Kód:

První jsem si vytvořila funkci monte_carlo, která přijme jako hodnotu počet iterací. Ve funkci si první určím proměnné všech kombinací s hodnotou 0. Poté vytvořím klasický pokerový balíček s 52 kartami jako list dvojic. První z dvojice je hodnost. Každá karta má hodnost od 2 do 14 s tím, že J je 11, Q je 12, K je 13 a A je 14. Druhá z dvojice je barva. Poté za každou iteraci balíček zamíchám a uložím prvních 5 karet z balíčku do proměnné liznute_karty. Dále kontroluji, jakou mám kombinaci v liznute_karty.

```python
def monte_carlo(pocet_iteraci):
    cista_postupka = 0
    ctverice = 0
    trojice_a_dvojice = 0
    barva5 = 0
    postupka = 0
    trojice = 0
    dvojice = 0
    par = 0
    zadna_kombinace = 0

    balicek = [(hodnost, barva) for hodnost in range(2, 15) for barva in ['Piky', 'Srdce', 'Káry', 'Kříže']]
    for i in range(pocet_iteraci):
        random.shuffle(balicek)
        liznute_karty = balicek[:5] #Vezme prvních 5 karet z balíčku
        if liznuta_kombinace(liznute_karty) == "Čistá postupka":
            cista_postupka += 1
        if liznuta_kombinace(liznute_karty) == "Čtveřice":
            ctverice += 1
        if liznuta_kombinace(liznute_karty) == "Trojice a Dvojice":
            trojice_a_dvojice += 1
        if liznuta_kombinace(liznute_karty) == "Barva":
            barva5 += 1
        if liznuta_kombinace(liznute_karty) == "Postupka":
            postupka += 1
        if liznuta_kombinace(liznute_karty) == "Trojice":
            trojice += 1
        if liznuta_kombinace(liznute_karty) == "Dvojice":
            dvojice += 1
        if liznuta_kombinace(liznute_karty) == "Pár":
            par += 1
        if liznuta_kombinace(liznute_karty) == "Žádná kombinace":
            zadna_kombinace += 1
```

Jakou držím kombinaci zjistím pomocí funkce liznuta_kombinace, která přijme jako hodnotu liznute_karty. V této funkci si první zadám několik prázdných proměnných, do kterých budu ukládat potřebné informace. Poté si rozdělím dvojice z líznutých karet, a to na hodnost a barvu, které přidám do listů hodnosti a barvy. Abych mohla určit, kolik je přesně jakých hodností v mém listu, použiji Counter, který vytvoří slovník s danou hodností jako key a jejím počtem jako value. Dále si hodnosti seřadím od nejmenšího k největšímu. Obě tyto proměnné se budou později hodit k určení líznuté kombinace.

```python
def liznuta_kombinace(liznute_karty):
    vysledek = ""
    mezery = 0
    barvy = []
    hodnosti = []
    pocet_hodnoty = []

    for karta in liznute_karty: #Projde všechny
        hodnost, barva = karta
        hodnosti.append(hodnost)
        barvy.append(barva)

    pocet_stejnych_hodnosti = Counter(hodnosti)
    serazene_hodnosti = sorted(hodnosti)
```

Dále pomocí len(set(barvy)) určím, jestli jsou všechny líznuté karty stejné barvy či nikoliv. Set nám totiž vymaže z listu všechny duplicitní výrazy a ukáže je pouze jednou, tudíž pokud se len bude rovnat 1, tak jsou všechny karty stejné barvy, z čehož by vyplývalo, že výsledek je pouze barva či čistá postupka. Pokud by byl výsledek jiný než 1, máme vícero možností.

V případě, kdy je len(set) jiný než 1, tak začnu tím, že projdu values slovníku, který jsem vytvořila pomocí Counter, a všechny počty hodnot si přidám do nového listu. V tomto novém listu určím maximum. Dále budu kontrolovat, jaká je délka slovníku. Values ve slovníku se vždycky musí rovnat 5. To znamená, že když mám délku slovníku 2, tak values musí být buď 4 a 1 nebo 3 a 2, to jsou dvě možné kombinace, proto kromě délky slovníku musím ještě určit maximální hodnotu, která se tam objevila. Když to bude 4, bude výsledná kombinace čtveřice. Ve stejném duchu určím kombinace trojice a dvojice, samotné trojice, samotné dvojice a páru. Jako poslední kombinaci v tomto případě musím určit postupku. První určím postupku s esem na začátku, která je speciálním případem, protože máme eso dané s hodností 14 nikoliv 1. Všechny ostatní postupky zjistím způsobem vypočítání mezer. V listu seřazených hodnot vždy odečtu druhé číslo od prvního, pak třetí od druhého atd. Pokud jdou čísla za sebou, výsledek musí vyjít 4, tím pádem máme postupku. Pokud číslo nevyšlo 4, nezbývá jiná možnost než žádná kombinace.

```
if len(set(barvy)) != 1:  #Spustí se, když nemají všechny karty stejn
    for pocet in pocet_stejnych_hodnosti.values(): #Projde values slo
        pocet_hodnoty.append(pocet)
    maximum = max(pocet_hodnoty)
    if len(pocet_stejnych_hodnosti) == 2 and maximum == 4:
        vysledek = "Čtveřice"
    elif len(pocet_stejnych_hodnosti) == 2 and maximum == 3:
        vysledek = "Trojice a Dvojice"
    elif len(pocet_stejnych_hodnosti) == 3 and maximum == 3:
        vysledek = "Trojice"
    elif len(pocet_stejnych_hodnosti) == 3 and maximum == 2:
        vysledek = "Dvojice"
    elif len(pocet_stejnych_hodnosti) == 4:
        vysledek = "Pár"
    elif serazene_hodnosti == [2, 3, 4, 5, 14]:  #Speciální případ po
            vysledek = "Postupka"
    else:
        for i in range(len(serazene_hodnosti) - 1): #Kontroluje, jest
            mezera = serazene_hodnosti[i + 1] - serazene_hodnosti[i]
            mezery += mezera
            if mezery == 4:
                vysledek = "Postupka"
            else:
                vysledek = "Žádná kombinace"
```

V případě, kdy se len(set) rovná 1, budu postupovat úplně stejně jako u obyčejné postupky s jedním rozdílem. Když se mezery nebudou rovnat 4, tak výsledná kombinace bude barva.

```
elif len(set(barvy)) == 1:  #Spustí se, když mají všechny karty stejn
    if serazene_hodnosti == [2, 3, 4, 5, 14]:  #Speciální případ post
        vysledek = "Čistá postupka"
    else:
        for i in range(len(serazene_hodnosti) - 1): #Kontroluje, jest
            mezera = serazene_hodnosti[i + 1] - serazene_hodnosti[i]
            mezery += mezera
            if mezery == 4:
                vysledek = "Čistá postupka"
            else:
                vysledek = "Barva"
```

Dále si už jen do monte_carlo funkce přidám výpočty pro pravděpodobnosti všech kombinací, které vypočítám jako danou kombinaci / počet iterací. Tyto hodnoty vrátím.

Abych tyto hodnoty mohla jednotlivě vypsat, tak si je ještě musím rozdělit, protože se mi z funkce vrátily ve formě tuple a pak už jenom vypíšu.

```
Pravděpodobnost líznutí čisté postupky při počtu iterací 1000000 je 0.0019%.
Pravděpodobnost líznutí čtveřice při počtu iterací 1000000 je 0.0253%.
Pravděpodobnost líznutí trojice a dvojice při počtu iterací 1000000 je 0.1517%.
Pravděpodobnost líznutí barvy při počtu iterací 1000000 je 0.1948%.
Pravděpodobnost líznutí postupky při počtu iterací 1000000 je 0.4013%.
Pravděpodobnost líznutí trojice při počtu iterací 1000000 je 2.1073%.
Pravděpodobnost líznutí dvojice při počtu iterací 1000000 je 4.7419%.
Pravděpodobnost líznutí páru při počtu iterací 1000000 je 42.2628%.
Pravděpodobnost líznutí žádné kombinace při počtu iterací 1000000 je 50.1130%.
```

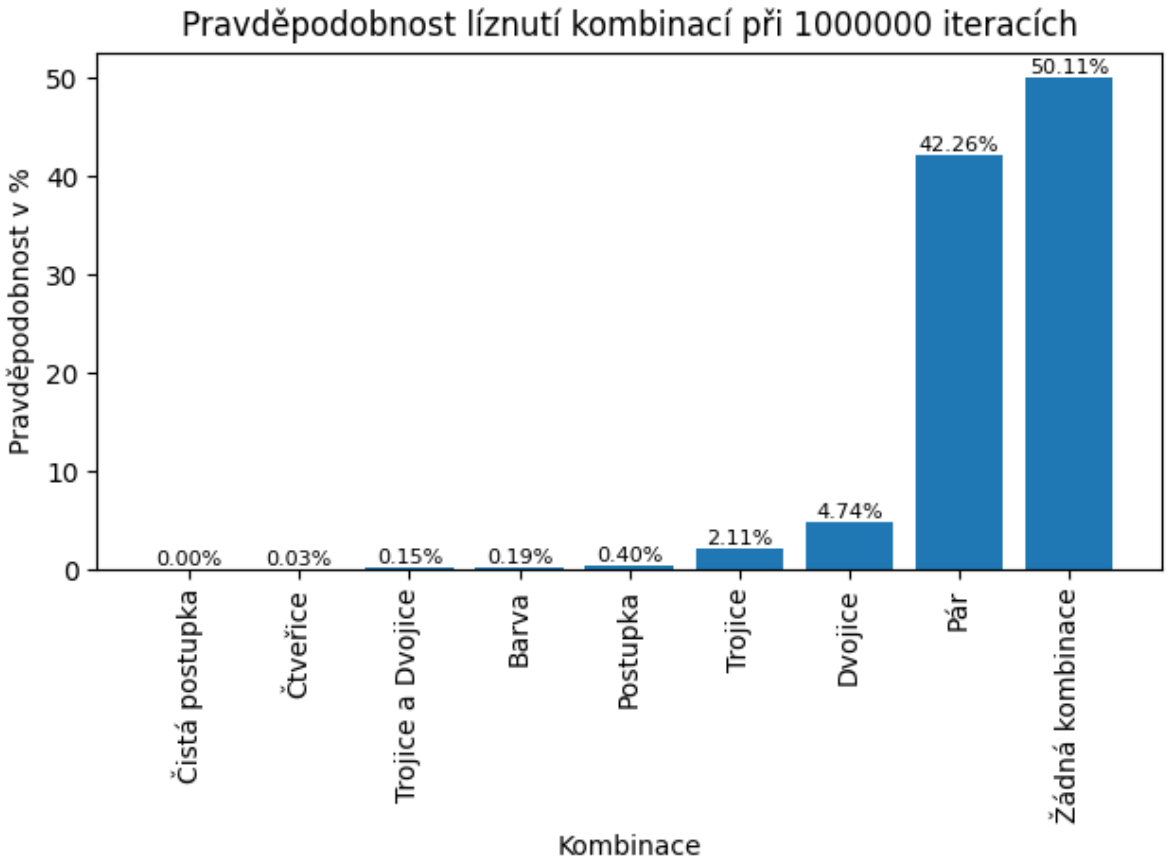
Pro lepší znázornění jsem použila knihovnu matplotlib.pyplot, díky které jsem vytvořila sloupcový graf, na kterém je výsledek krásně vidět. Stačilo uložit výsledky pravděpodobnosti a názvy kombinací do dvou listů a vložit je do plt.bar. Na závěr jsem ještě upravila názvy kombinací otočením o 90 stupňů a k jednotlivým sloupcům přidala jejich hodnotu pomocí enumerate a plt.annotate.

```
pravdepodobnosti = [pr_cista_postupka * 100, pr_ctverice * 100, pr_trojice_a_dvojice * 100, pr_barva * 100, pr
pr_zadna_kombinace * 100]
kombinace = ['Čistá postupka', 'Čtveřice', 'Trojice a Dvojice', 'Barva', 'Postupka', 'Trojice', 'Dvojice', 'Pá

#Vytvoří graf
plt.bar(kombinace, pravdepodobnosti)
plt.title(f"Pravděpodobnost líznutí kombinací při {pocet_iteraci} iteracích")
plt.xlabel("Kombinace")
plt.ylabel("Pravděpodobnost v %")

#Upraví názvy kombinací a přidá nejvyšší hodnotu na jednotlivých sloupcích
plt.xticks(rotation = 90)
for i, pravdepodobnost in enumerate(pravdepodobnosti):
    plt.annotate(f"{pravdepodobnost:.2f}%", (i, pravdepodobnost), ha = "center", va = "bottom", fontsize = 8)

plt.tight_layout()
plt.show()
```



Jak můžeme vidět, graf postupně stoupá od čisté postupky až po žádnou kombinaci. Výsledek tedy v závislosti na hodnosti výherních kombinací dává smysl.