Use Case - Finding the winning strategy in a Card game in Python

15/10/25

Problem description:- Imagine a card game where each player receives on hand of cards with values. The objective is to find the best way to maximize the score for a player, assuming the players take turns drawing cards - Each player can either pick the first or last card from the remaining pile.

Assumptions:-

* Each player tries to maximize their score.
* Cards are represented by integers, which indicate their values
* Two players alternate turns and each player picks a card from either the beginning or the end of the list.

You need to design an algorithm that helps a player find the optimal strategy to guarantee the highest possible score given that the opponent is also playing optimally.

Plan:-

We can solve this problem using Dynamic programming by calculating the optimal score for every possible score given that the opponent scenario, taking into account the best choice for both players.

Steps:-

1. Define the Game:- Represent the pile of cards as a list of integers.
2. Recursive strategy: A function will recursively determine the best score a player can achieve.
3. Dynamic Programming :- Store intermediate results to avoid recalculating them.
4. Base cases:- when only one card is left, the current player takes it.

Program:

```
def find_optimal_strategy(cards):
    n = len(cards)
#Create a memoization table to
    dP = [[0]* n for _ in range(n)]
# Fill the table for subproblems of increasing size
    for length in range(1, n+1):
        for i in range(n-length+1):
            j = i + length - 1
#If only one card is left, the player takes it
            if i == j:
                dP[i][j] = cards[i]
            else:
#Choose the best of two choices
#1. Take the left card, and the opponent plays
#optimally on the remaining (i+1, j)
# 2. Take the right card, and the opponent plays
    optimally on the remaining (i, j-1)
    take_left = cards[i] - dP[i+1][j]
    take_right = cards[j] - dP[i][j-1]
    dP[i][j] = max(take_left, take_right)
# dP[0][n-1] will have the optimal score difference
    for the first player return (dP[0][n-1] + sum(cards)
    2 # First player's maximum possible score.
# Examples case
cards = [3, 9, 1, 2]
print("First Player's optimal score:", find_optimal_strategy
    (cards))
```

Example walkthrough:-

Consider the array of cards: [3, 9, 1, 2]

1. First player (you) can choose between:
   - Taking the leftmost card (3), leaving the cards [9, 1, 2]
   - Taking the rightmost card (2), leaving the card [3, 9, 1]

2. The opponent will then take their turn, playing optimally to minimize the first player's score.

This program computes the best possible for the first player

(=irst player> optimal score: 5

irst player, if plays optmaly, can guarantee a score of 5 regardless of how the opponent plays

ptimizing Scatergy:-

y using dynamic programming we ensure that the solution is computed efficiently, avoiding redundant calculations. This approach ensures both players play optimally, and the first player sets the highest score possible given the opponent's best move.

VEL TECH	
EX No.	13
PERFORMANCE (5)	5
RESULT AND ANAYSIS (5)	5
VIVA VOCE (5)	5
RD (5)	5
	20
TE	25/10