PROPOSAL STUDI KASUS

INFORMATIKA

ANALISIS ALGORITMA DAN LOGIKA PADA PERMAINAN CAPSA ATAU BIG TWO

Dosen Pengampu:

Merryana Lestari, S.Kom., M.Kom

Disusun oleh :

Varel Gabriel Wungow – 32230062

Nathan Raffael Simatupang - 32230078

UNIVERSITAS BUNDA MULIA

SERPONG

2024

ABSTRAKSI

Capsa atau "*Big Two*" adalah permainan kartu yang cukup dikenal di Indonesia, bahkan di seluruh dunia. Capsa dimainkan oleh 4 orang pemain. Capsa memiliki aturan permainan yang cukup beragam dan tergantung di daerah mana capsa dimainkan dan kebiasaan main pemain yang berbeda-beda. Indonesia memiliki jenis aturan permainan capsa sendiri yang berbeda dengan aturan di negara lain. Permainan capsa cukup menarik dan mudah dimainkan. Untuk memenangkan permainan capsa, dibutuhkan sedikit pemikiran taktis dan strategi yang jitu, proses berpikir yang tidak tidak terlalu berat inilah yang membuat permainan kartu ini cocok untuk menghilangkan kebosanan. Dalam makalah ini, penulis berusaha menerapkan algoritma greedy ke dalam permainan capsa dalam bentuk strategi permainan yang dapat digunakan untuk memenangkan permainan capsa. Walaupun begitu, pemikiran dan strategi bukanlah satu-satunya faktor yang menentukan kemenangan dalam capsa. Seperti kebanyakan permainan kartu lainnya, keberuntungan juga memiliki bagian yang cukup berdampak dalam permainan ini. Algoritma greedy sendiri bukan algoritma yang mampu menghasilkan hasil yang mangkus dan sangkil dalam setiap kondisi. Oleh karena itu, strategi permainan capsa dengan menggunakan prinsip algoritma greedy merupakan alternatif strategi dalam permainan capsa.

Keywords—capsa/big two; greedy; capsa/big two; permainan

BAB I STUDI KASUS

Permainan kartu adalah permainan yang sangat populer di berbagai kalangan. Permainan kartu dapat dimainkan bersama teman, keluarga, bahkan ada yang bisa dimainkan sendirian. Terdapat berbagai jenis permainan kartu, Salah satu diantaranya adalah permainan capsa. Permainan capsa merupakan permainan yang dimainkan oleh 2 hingga 4 orang dengan menggunakan kartu remi. Kartu remi merupakan sekumpulan kartu yang berisi simbol sekop, hati, keriting, dan wajik. Masing-masing symbol memiliki 13 nilai, yaitu 2 hingga 10, jack, queen, king dan as. Total kartu pada satu deck kartu capsa adalah 52 kartu. Untuk memenangkan permainan capsa, pemain harus menghabiskan kartu yang ada di tangannya paling pertama. Untuk mencapai itu, diperlukan sebuah strategi. Bermain secara asalasalan dapat menyebabkan pemain untuk tidak membuang kartu atau bahkan menambahkan kartu di tangan, tergantung variasi peraturan apa yang digunakan. Terdapat berbagai macam strategi yang dapat diterapkan pada permainan capsa. Salah satunya adalah strategi yang mengimplementasikan algoritma greedy. Algoritma greedy adalah sebuah algoritma yang tujuan mulanya adalah untuk mengoptimasi program. Seperti namanya; "Greedy" yang berarti rakus atau serakah, algoritma ini mengutamakan pilihan terbaik hanya pada sat itu saja dan tidak mengkalkulasikan *outcome* untuk pilihan nantinya

BAB II
ALGORITMA YANG DIAJUKKAN

Algoritma Greedy adalah algoritma yang ditujukan untuk menyelesaikan permasalahan dengan tujuan mengoptimalkan suatu nilai. Greedy dalam Bahasa Indonesia berarti rakus. Sesuai Namanya, algoritma ini akan mencari nilai optimum lokal di setiap langkahnya dengan harapan mendapatkan nilai optimum global di akhirnya. Ini sejalan dengan semboyan algoritma greedy, yaitu "Take what you can get now". Algoritma ini lebih cepat dibandingkan dengan algoritma brute force atau exhaustive search. Namun, algoritma ini tidak menjamin nilai akhir adalah nilai optimum global. Selain itu algoritma ini juga memberikan kita banyak pilihan sehingga dapat memilih kartu pilihan kita. Hasil yang dikeluarkan apabila kita menggunakan algoritma greedy dapat lebih optimal.

Dalam permainan capsa atau "big two", algoritma greedy dapat diterapkan dengan cara sebagai berikut:

1. Definisikan Kriteria Pilihan Terbaik: Tentukan kriteria yang digunakan untuk menentukan kartu terbaik yang harus dimainkan pada setiap giliran. Kriteria ini bisa berupa jumlah poin yang bisa diperoleh, kemungkinan memenangkan "*trick*" (pada permainan seperti Bridge), atau kriteria lain yang sesuai dengan aturan permainan.

2. Evaluasi Kartu yang Tersedia: Pada setiap giliran, evaluasi kartu yang tersedia dalam tangan berdasarkan kriteria yang telah ditetapkan. Kartu yang paling sesuai dengan kriteria tersebut akan menjadi pilihan terbaik pada tahap tersebut.

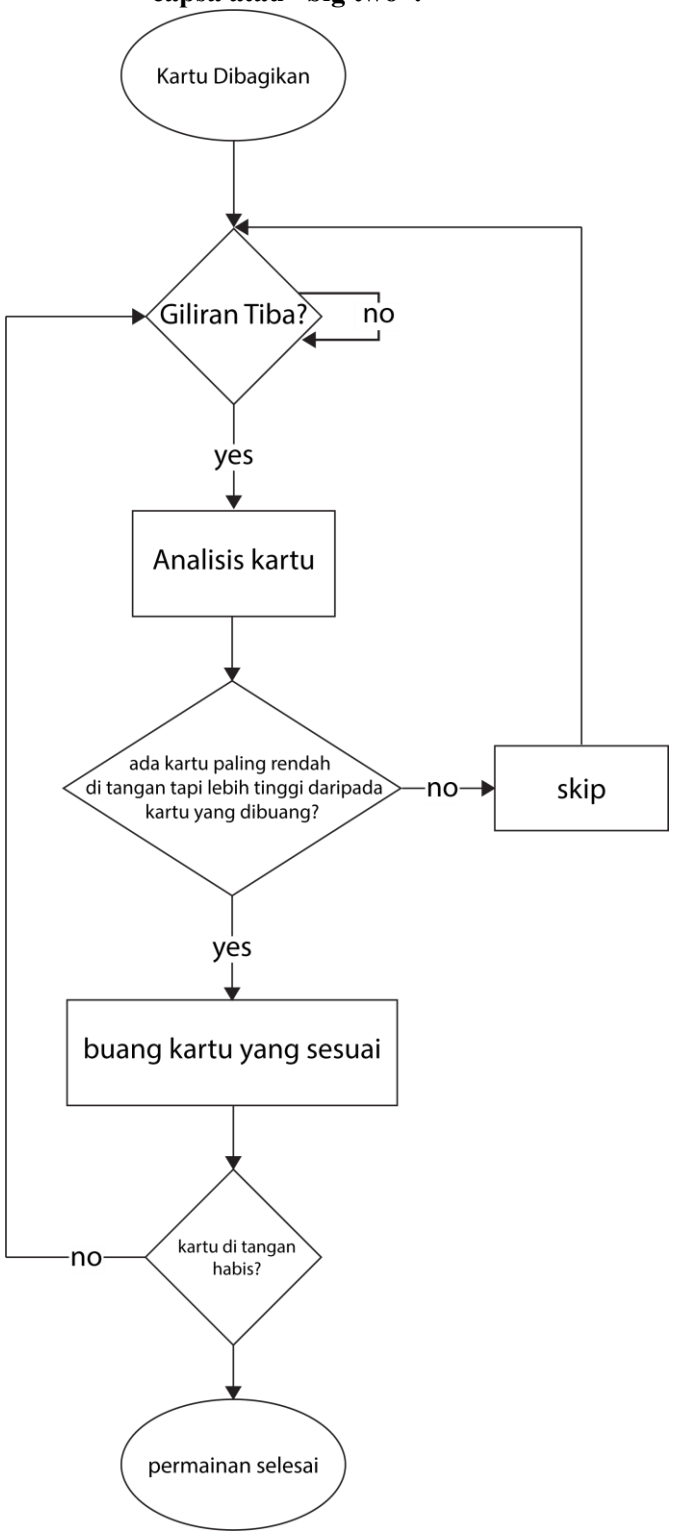
3. Pilih Kartu Terbaik: Setelah evaluasi, pilih kartu yang memenuhi kriteria tersebut. Algoritma greedy akan memilih kartu terbaik pada setiap giliran berdasarkan kriteria yang telah ditetapkan.

4. Lakukan Iterasi: Ulangi proses tersebut untuk setiap giliran selama permainan berlangsung, dengan mempertimbangkan kartu yang tersedia pada setiap tahap.

5. Perbarui Status Permainan: Setelah setiap giliran, perbarui status permainan dan evaluasi ulang kriteria jika diperlukan. Misalnya, jika ada kartu tertentu yang sudah dimainkan oleh lawan, itu dapat mempengaruhi kriteria yang digunakan untuk memilih kartu pada giliran berikutnya.

Meskipun algoritma greedy dapat memberikan solusi yang cepat, namun tidak selalu menghasilkan solusi terbaik secara keseluruhan, terutama dalam permainan yang kompleks seperti capsa. Oleh karena itu, seringkali perlu dikombinasikan dengan strategi lain untuk meningkatkan kualitas keputusan, tapi dalam implementasi kodingan kelompok kami, kita tidak menggunakan greedy dengan kombinasi strategi lain, melainkan hanya greedy seutuhnya

Flowchart algoritma greedy pada permainan capsa atau "big two":



Flowchart Algoritma Greedy

Ilustrasi Input/Output dan implementasi kodingan Python:

Untuk mengimplementasikan permainan kartu capsa kedalam bentuk kode program (python) maka pertama kita harus mengetahui peraturan paling dasar permainan kartu capsa, dan apa saja yang harus diperlukan jika ibaratnya kita akan memainkannya secara konvensional di dunia nyata.

Hal-hal yang diperlukan:

- Kartu main : kartu main yang berjumblah 52 (tanpa joker) kartu yaitu *Diamond :* 2-Ace, *Clove :* 2-Ace*, Heartens :* 2-Ace*, Spaids :* 2-Ace.
- Pemain yang berjumblah 4 orang maksimal : untuk memainkan kartu capsa, maka diperlukan pemain yang berjumblah maksimal sampai 4 orang.

Ilustrasi permainan:

1. kartu akan diacak *(shuffle)* agar tidak ada pemain yang dapat mengetahui kartu apa yang akan mereka dapatkan, dan kartu apa yang akan lawan mereka dapatkan.
2. Setelah kartu telah teracak, maka kartu akan dibagikan kepada masing-masing pemain secara merata. Jika total pemain ada 4 orang maka kartu yang akan masing-masing pemain dapatkan adalah 13 kartu
3. Setelah kartu dibagikan secara acak dan merata, maka permainan dimulai. Permainan dimulai dan pada putaran pertama ini, semua pemain akan membuang kartu paling rendah yaitu kartu 3 dengan logo apapun seperti : Diamond 3, Clove 3, Heartens 3, dan Spaids 3. Tujuan dari membuang kartu-kartu yang berangka 3 tersebut adalah untuk menentukan siapa yang akan menjadi pemain pertama untuk mengeluarkan kartu selanjutnya, pemain yang akan mengeluarkan kartu selanjutnya adalah pemain yang memiliki kartu tertinggi yaitu Spaids 3.
4. Setelah menentukan pemain pertama yang mengeluarkan kartu, pemain selanjutnya yang akan mengeluarkan kartu adalah pemain berikutnya. Contoh : jika pemain nomor dua yang mengeluarkan Spaids 3 maka dia menjadi orang yang mengeluarkan kartu paling pertama, maka setelah pemain nomor dua, pemain nomor tiga lah yang menjadi giliran mengeluarkan kartu, setelah pemain nomor tiga, maka pemain nomor empat, dan karena pemain nomor empat adalah pemain terakhir, maka balik ke pemain nomor satu, dan pemain nomor dua setelahnya. Perputaran itu dilakukan berulang-ulang kali sampai mendapatkan pemain yang kartunya sudah habis duluan, dan dialah yang menjadi pemenang.

IMPLEMENTASI KODINGAN KE DALAM PYTOHN

```python
import random
import time
try:
    from colorama import Fore, Back, Style
    print(f"\n{Fore.GREEN}COLORMA INCLUDED{Style.RESET_ALL}\n")
    Bred = Back.RED
    red = Fore.RED
    fgreen = Fore.GREEN
    Black = Fore.BLACK
    fblue = Fore.BLUE
    fmagenta = Fore.MAGENTA
    fyelow = Fore.YELLOW
    res = Style.RESET_ALL
    bwhite = Back.WHITE
    Bblue = Back.BLUE
except:
    print("colorma not included")
    print("output will not be colored")
    time.sleep(3)
    red = str("")
    fgreen = str("")
    Black = str("")
    fblue = str("")
    fmagenta = str("")
    fyelow = str("")
    res = str("")
    bwhite = str("")
    Bblue = str("")
```

```python
def CardGenerate(cl): #fungsi menghasilkan semua 52 kartu main secara otomatis
    style = str(input("gunakan Unicode(gambar) atau Huruf sebagai simbol kartu? 1/2 : "))
    if style == "1": # memilih penggunaan output yang akan digunakan untuk simbol kartu
        default_symbols = {'♦' : 0.1, '♣' : 0.2, '♥' : 0.3, '♠' : 0.4} #list simbol limbol kartu (logo/Unicode)
    elif style == "2":
        default_symbols = {'D' : 0.1, 'C' : 0.2, 'H' : 0.3, 'S' : 0.4} #list simbol limbol kartu (alfabet)
        print("D = Diamond\nC = Clove\nH = Hearten\nS = Spaids")
    else:
        default_symbols = {'♦' : 0.1, '♣' : 0.2, '♥' : 0.3, '♠' : 0.4} #list simbol limbol kartu (logo/Unicode)

    Hrank = {'J':11, 'Q':12, 'K':13, 'A':14, '2':15}
    for symbol in default_symbols:
        for lowR in range(3,11):
            cl[symbol + str(lowR)] = lowR + default_symbols[symbol]
        for highR in Hrank:
            cl[symbol + str(highR)] = Hrank[highR] + default_symbols[symbol]
    return cl, style
```

```python
def ShowDealerCards(): #method untuk menunjukkan semua kartu yang ada
    for cards in cards_list:
        print(f"{cards} : {cards_list[cards]}")
    print("cards total : ",len(cards_list))

```

```python
def GiveCard(): #method tuntuk membagikan kartu pada masing-masing pemain seacra acak
    choose_player = random.choice(list(players))
    if len(list(choose_player)) != 13:
        choose_card = random.randint(0, len(cards_list))
        try:
            choose_player[list(cards_list)[choose_card]] = cards_list[list(cards_list)[choose_card]]
            cards_list.pop(list(cards_list)[choose_card])
        except:
            print("", end=" ")
    else:
        GiveCard()
```

```
def ShowAllPlayerCards(): #method untuk menunjukkan semua kartu yang ada pada masing masing pemain
    print(f"\n{fgreen}PLAYER 1 : {list(p1)} \n{fblue}PLAYER 2 : {list(p2)} \n{fmagenta}PLAYER 3 : {list(p3)} \n{fyelow}PLAYER 4 : {list(p4)}{res}")

    print(len(p1),": player one cards total")
    print(len(p2)," : player two cards total")
    print(len(p3)," : player three cards total")
    print(len(p4)," : player four cards total")
    print(len(cards_list)," : cards left on the dealer")
    print(len(cards_on_the_table)," : cards thrown on the table")
```

```
def InitiatePlay(sty): #fungsi untuk memainkan game
    print()
    time.sleep(0.3)
    print(bwhite + Black + "Game initiating..." + res)
    time.sleep(0.3)
    print("\nYou are player number one.")
    print("Here is your hand:")
    for i in p1:
        print(fgreen + f"{i}" + res, end= " | ")
    #putaran pertama | buang 3
    print()
    for turn in players:
        print()
        print(f"Player {players.index(turn)+1}")
        for check in turn:
            if (turn[check] == 3.1) or (turn[check] == 3.2) or (turn[check] == 3.3):
                print(check)
                cards_on_the_table[check] = turn[check]
            elif turn[check] == 3.4:
                first = players[players.index(turn)]
                firstnm = players.index(turn)
                cards_on_the_table[check] = turn[check]
                print(red + f"{check}" ,res)
    def recur_del_threes(iteration,style):
        if iteration != 4:
            for y in range(1,5):
                if style == "1":
                    for i in ['♠3','♣3','♥3','♦3'] :
                        if i in players[iteration]:
                            #print(players[iteration])
                            players[iteration].pop(i)
                elif style == "2":
                    for it in ['D3','C3','H3','S3'] :
                        if it in players[iteration]:
                            #print(players[iteration])
                            players[iteration].pop(it)
                else:
                    for ith in ['♥3','♣3','♥3','♠3'] :
                        if ith in players[iteration]:
                            #print(players[iteration])
                            players[iteration].pop(ith)
            recur_del_threes(iteration + 1, style)
    recur_del_threes(0,sty)
    temp = {}
    if style == "1":
        temp['♦3'] = max(cards_on_the_table.values())
        cards_on_the_table.pop('♦3')
        cards_on_the_table['♦3'] = max(temp.values())
    elif style == "2":
        temp['S3'] = max(cards_on_the_table.values())
        cards_on_the_table.pop('S3')
        cards_on_the_table['S3'] = max(temp.values())
    else:
        temp['♠3'] = max(cards_on_the_table.values())
        cards_on_the_table.pop('♠3')
        cards_on_the_table['♠3'] = max(temp.values())
    print(f"player {firstnm + 1} will be playing first")
    return firstnm
```

```python
def CheckThrownCards(): #method untuk menunjukkan semua kartu yang telah dibuang oleh pemain-pemain
    print("\nthrown cards are : ")
    for i in cards_on_the_table:
        print(f"{i}", end= " | ")

```

```python
def RecursivePlay(turns, win, skip): #fungsi rekursif untuk memainkan game secara berulang kali sampai ada pemenang
    CheckThrownCards()
    #ShowAllPlayerCards()
    print(f"\nskiped {skip} times")
    if skip == 3:
        cards_on_the_table[list(cards_on_the_table.keys())[-1]] = 0
        print(f"{Bblue}{list(cards_on_the_table.keys())[-1]} became free{res}")
        skip = 0
    current_highest_card = max(list(p1.values()) + list(p2.values()) + list(p3.values()) + list(p4.values()))
    is_thrown_highcard = False
    for plyr in players:
        for keys in plyr:
            if plyr[keys] == current_highest_card:
                ky = keys
    #print("highest card",current_highest_card, "or", ky)
    if turns == 0:
        print(f"\n{bwhite}your turn{res}")
        if max(list(p1.values())) < list(cards_on_the_table.values())[-1]:
            print(f"\n{bwhite}{Black} you skiped{res}")
            RecursivePlay(turns + 1,False, skip + 1)
            return
        else:
            print("\nyour turn to play")
            print("here is your hand : ")
            for i in p1:
                print(fgreen + f"{i}" + res, end= " | ")
            #throw = str(input(f"what card do you want to throw : {fgreen}"))
            throw = random.choice(list(p1.keys()))
            print(res)
            while throw not in list(p1.keys()):
                print("card unavailable")
                #throw = str(input("please input again : "))
                throw = random.choice(list(p1.keys()))
            while (p1[throw] < list(cards_on_the_table.values())[-1]):
                print(f"\n{throw} card is too low")
                #throw = str(input("please input again: "))
                throw = random.choice(list(p1.keys()))

            if p1[throw] == current_highest_card:
                is_thrown_highcard = True
                print(f"{bwhite}{Black}player {turns + 1} throws {throw}{res}")
                cards_on_the_table[throw] = 0
                p1.pop(throw)
            else:
                print(f"{bwhite}{Black}player {turns + 1} throws {throw}{res}")
                cards_on_the_table[throw] = p1[throw]
                p1.pop(throw)
        #time.sleep(0.1)
```

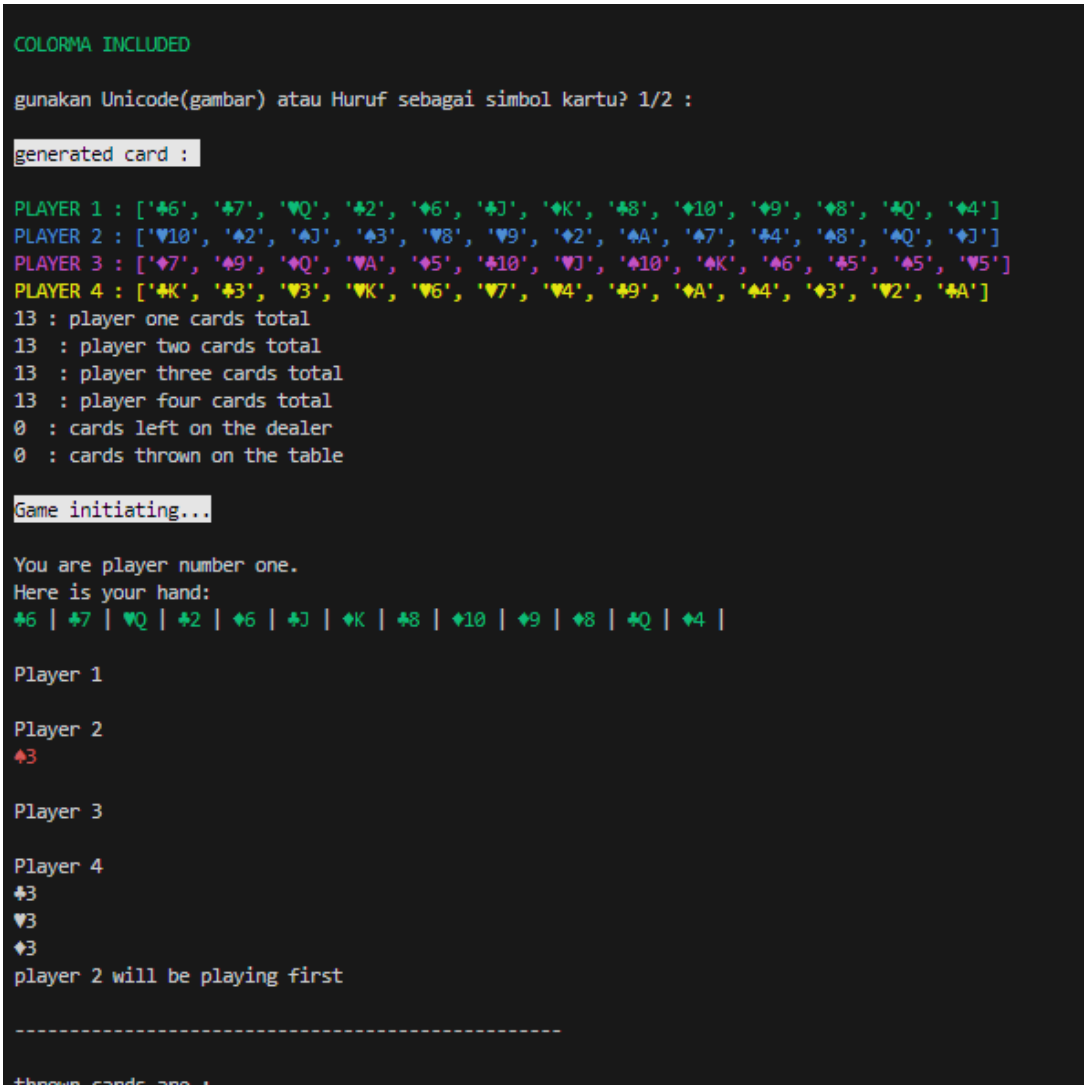
```python
elif turns == 1:
    print(f"{bwhite}player 2's turn{res}\n")

    if max(list(p2.values())) < list(cards_on_the_table.values())[-1]:
        print(f"\n{bwhite}{Black} player {turns + 1} skiped{res}")
        if turns != 3:
            RecursivePlay(turns + 1, False, skip + 1)
            return
        elif turns == 3:
            RecursivePlay(turns - 3, False, skip + 1)
            return
        pass

    print(f"{Bred}GREEDY INITIATED{res}")

    temp_dict_unsorted = p2.copy()
    temp_dict_sorted = {}

    def sort_the_temporary_dict():
        if len(temp_dict_sorted) != len(p2):
            temp_dict_sorted[list(temp_dict_unsorted.keys())[list(temp_dict_unsorted.values()).index(min(list(temp_dict_unsorted.values())))]] = min(list(temp_dict_unsorted.values()))
            temp_dict_unsorted.pop(list(temp_dict_unsorted.keys())[list(temp_dict_unsorted.values()).index(min(list(temp_dict_unsorted.values())))])
        else:
            sort_the_temporary_dict()

    while len(temp_dict_sorted) < len(p2):
        sort_the_temporary_dict()

    for element in temp_dict_sorted:
        if p2[element] < list(cards_on_the_table.values())[-1]:
            pass
        elif p2[element] > list(cards_on_the_table.values())[-1]:
            put = element
            break

    cards_on_the_table[put] = p2[put]
    print(f"{bwhite}{Black}player 2 throws {put}{res}")
    p2.pop(put)
    time.sleep(0.1)
```

```
    elif (turns == 2) or (turns == 3):
        print(f"\n{bwhite}player {turns + 1}'s turn{res}")

        if max(list(players[turns].values())) < list(cards_on_the_table.values())[-1]:
            print(f"\n{bwhite}{Black} player {turns + 1} skiped{res}")
            if turns != 3:
                RecursivePlay(turns + 1, False, skip + 1)
                return
            elif turns == 3:
                RecursivePlay(turns - 3, False, skip + 1)
                return
            pass

        else:
            enemy_throw = random.choice(list(players[turns].keys()))
            while (players[turns][enemy_throw] < list(cards_on_the_table.values())[-1]):
                enemy_throw = random.choice(list(players[turns].keys()))

            if players[turns][enemy_throw] == current_highest_card:
                is_thrown_highcard = True
                print(f"{bwhite}{Black}player {turns + 1} throws {enemy_throw}{res}")
                cards_on_the_table[enemy_throw] = 0
                players[turns].pop(enemy_throw)
            else:
                print(f"\n{bwhite}{Black}player {turns + 1} throws {enemy_throw}{res}")
                cards_on_the_table[enemy_throw] = players[turns][enemy_throw]
                players[turns].pop(enemy_throw)
        #time.sleep(0.1)

    if len(players[turns]) == 0:
        print(f"\nGAME ENDED WITH PLAYER {turns + 1} AS THE WINER")
        return

    elif len(players[turns]) != 0:
        if is_thrown_highcard:
            RecursivePlay(turns, False, 0)
            return

        else:
            if turns != 3:
                RecursivePlay(turns + 1, False, 0)
                return
            elif turns == 3:
                RecursivePlay(turns - 3, False, 0)
                return
```

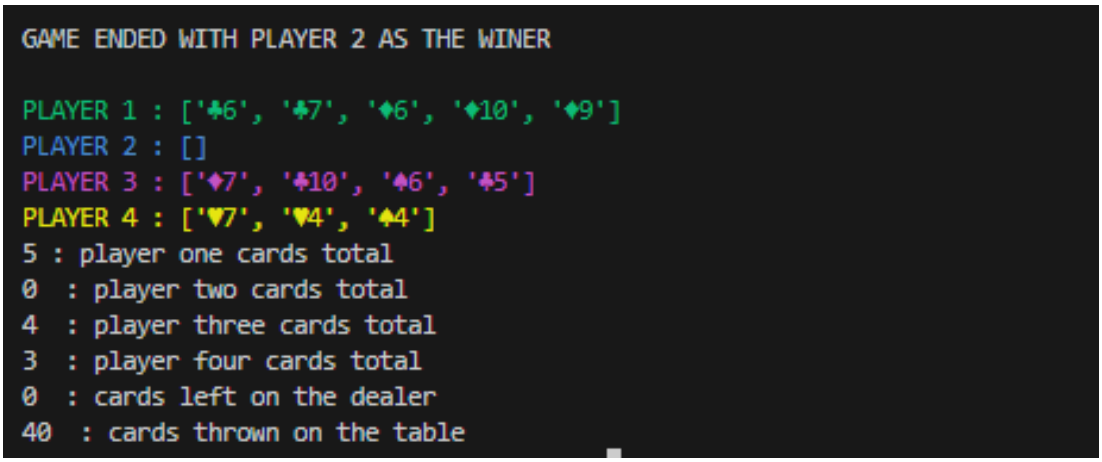
```
### =MAIN PROGRAM= ###
p1 = {}
p2 = {}
p3 = {}
p4 = {}
cl = {} #temporary card-list place holder
cards_on_the_table = {} #kartu yang telah dibuang dan sudah tidak dipegang oleh para pemain maupun dealer
players = [p1, p2, p3, p4] #list yang menyimpan masing-masing dictionary p1, p2, p3, p4

cards_list, style = CardGenerate(cl)

#ShowDealerCards()
while len(cards_list) != 0:
    #ShowDealerCards()
    #print(f"{'-' * 30}")
    GiveCard()

print(f"\n{bwhite}{Black}generated card : {res}")
ShowAllPlayerCards()
who_plays_first = InitiatePlay(style)
print()
print(f"{'-' * 50}") ################################
RecursivePlay(who_plays_first, False, 0)
ShowAllPlayerCards()
```

OUTPUT 1 (UNICODE/SIMBOL)

Awal game:

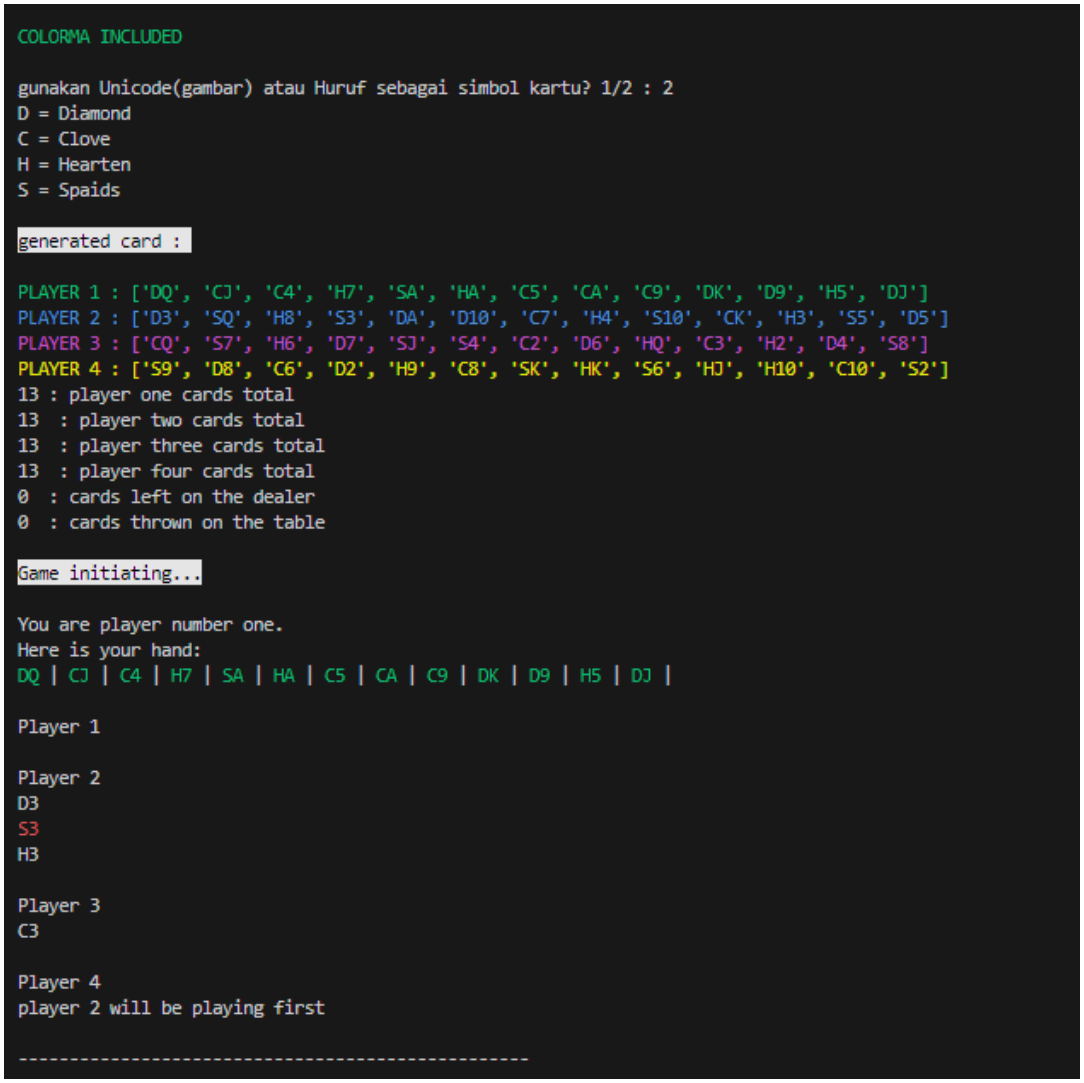


Akhir game:

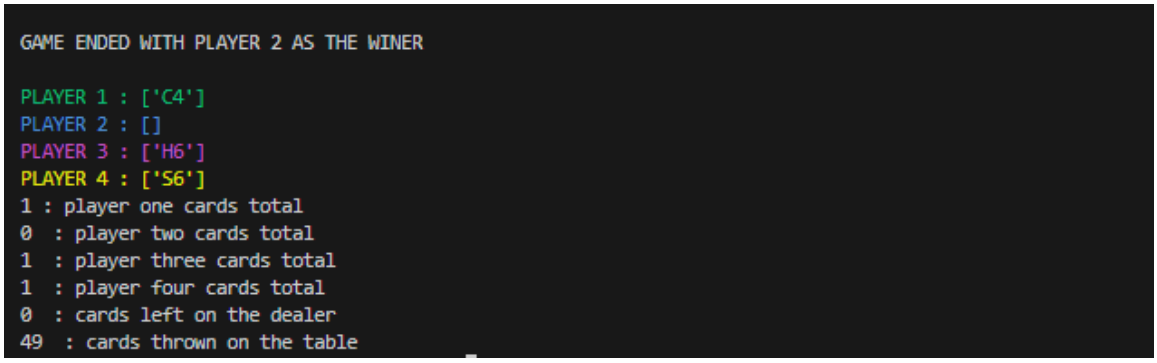


OUTPUT 2 (HURUF)

Awal game:



Akhir game:



BAB 3 HASIL

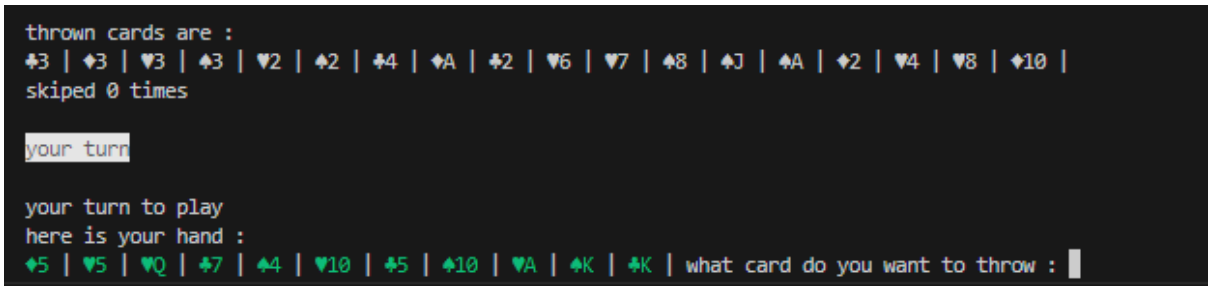
Dalam laporan ini, kelompok kami akan mempersembahkan hasil dari studi kasus tentang penerapan algoritma *Greedy* sebagai strategi untuk memainkan game Capsa atau *Big two.* Dalam laporan ini, kelompok kita memberikan penjelasan singkat tentang cara kerja kodingan yang kami buat, bagaimana cara untuk menjalankan program, dan bagaimana cara user dapat ikut serta bermain permainan capsa menggunakan kodingan yang kelompok kita berikan.

BAGAIMANA KODINGAN DIBUAT

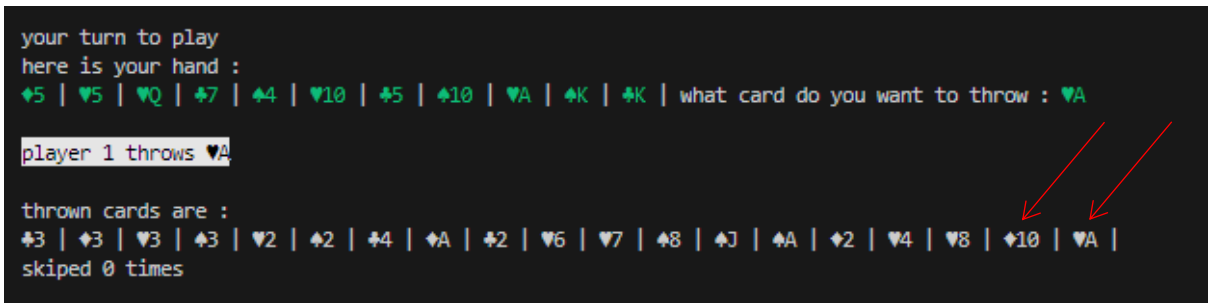
Kodingan yang kelompok kita buat pada dasarnya adalah prototipe dari game capsa, alasan mengapa kodingan tersebut hanyalah prototipe dari game capsa karena ada satu aturan main yang tidak kami implementasikan kedalam kode yaitu penggunaan kombinasi kartu atau *Hand Rankings* seperti : *pairs, threes, straight, flush, full house, four of a kind, straight flush* dan *royal flush.* Alasan mengapa penggunaan kombinasi kartu tidak diimplementasikan kedalam kodingan ada dua faktor yaitu, pertama, kodingan akan menjadi makin rumit dan waktu pengerjaan project tidaklah cukup untuk membuat permainan capsa yang utuh. Faktor kedua adalah, untuk mengimplementasikan dan menganalisis algoritma *Greedy* sebagai strategi bermain permainan capsa tidaklah perlu permainan capsa yang utuh, dan prototipe yang kolompok kami buat sudahlah cukup untuk menganalisis jalannya algoritma yang ingin dianalisa.

PERAN PLAYER 1-4 DAN PENERAPAN ALGORITMA *GREEDY* DALAM KODINGAN.

Lalu bagaimana algoritma *greedy* diimplementasikan kedalam program dan apa peran beserta dampaknya seiring berjalannya program? Untuk mengetahui itu, maka harus diketahui terlebih dahulu tentang cara kerja para pemain dalam program. Dalam program prototipe permainan capsa yang kami buat, ada empat pemain yang bergilir untuk membuang kartu. Peran algoritma *greedy* di situ adalah untuk menentukan kartu yang akan dibuang oleh “player 2”. Sedangkan untuk “player 3” dan “player 4”, cara mereka menentukan kartu apa yang dibuang adalah dengan hanya mengambil sembarang kartu dari tangan mereka (selama kartu yang ditentukan dapat mengalahkan kartu yang saat itu ada di meja). Untuk “player 1” cara dia menentukan kartu apa yang akan dibuang itu sedikit unik, karena yang menentukan kartu apa yang akan dikeluarkan “player 1” adalah user, dengan kata lain, user menggunakan inputan untuk menentukan kartu apa yang ingin dikeluarkan.



gambar contoh: terminal meminta inputan user untuk membuang kartu apa agar mengalahkan *diamond 10*



gambar contoh: user menginput *heartens ace* maka kartu yang dimeja(thrown cards) ditambahkan *heartens ace*

HASIL EKSPERIMEN DAN ANALISIS EFEKTIFITAS ALGORITMA *GREEDY*

Dibagian ini akan membahas hasil running program dan melihat seberapa efektif jika algoritma *greedy* dijadikan sebagai strategi untuk memainkan permainan capsa. Perlu diingatkan lagi bahwa algoritma *greedy* memang bisa dipakai untuk strategi bermain kartu capsa, tetapi karena sifat permainan yang agak bertolak belakang dengan cara *greedy* menyelesaikan masalah, maka algoritma *greedy* bukanlah strategi paling optimal untuk memainkan permainan kartu capsa.

TABEL PERCOBAAN 1

Run	Pemenang
1	Player 2
2	Player 4
3	Player 2
4	Player 1
5	Player 2
6	Player 3
7	Player 2
8	Player 2
9	Player 4
10	Player 4

TABEL PERCOBAAN 2

Run	Pemenang
1	Player 2
2	Player 2
3	Player 3
4	Player 2
5	Player 2
6	Player 1
7	Player 4
8	Player 2
9	Player 2
10	Player 2

Catatan: di test running pada tabel percobaan satu dan dua, player 1 bukanlah inputan user, melainkan disengaja menggunakan cara pengambilan kartu seperti player 3 dan 4 yaitu mengambil secara acak. Ini dilakukan untuk melihat perbandingan yang jelas se-efektif apa algoritma *greedy* yang diterapkan untuk menghadapi lawan yang seharusnya lebih inferior darinya.

Bisa dilihat dari tabel percobaan satu dan dua bahwa pemain yang menggunakan *greedy* (Player 2) memiliki kesempatan menang lebih tinggi bahkan bisa mencapai 70% tingkat kesuksesan melawan pemain-pemain yang hanya membuang kartu secara asal dan sembarang. Hasil ini cukup sesuai dengan prediksi kelompok kita yang awalnya beranggapan bahwa strategi algoritma *greedy* memang tidak akan menang setiap saat, tapi tentunya memiliki probabilitas menang yang lebih tinggi. Tapi kelompok kita hanya memperkirakan *outcome* tersebut akan terjadi jika player 2 hanya melawan player-player yang mengeluarkan kartu secara sembarang, jika player 2 yang menggunakan algoritma *greedy* dihadapi dengan algoritma yang dapat mengkalkulasikan strategi sampai beberapa langkah kedepan, maka kemungkinan besar player 2 akan kalah lebih banyak dibandingkan player dengan strategi atau algoritma yang lebih matang tersebut.

Cara kami mengimplementasikan algoritma *greedy* dalam kodingan sebagai player 2 dan bagaimana ia mencari kartu yang terbaik untuk dibuang pada saat itu adalah sebagai berikut.

1. Saat giliran player 2 tiba untuk kesempatan membuang kartu, cek terlebih dahulu apakah ada kemungkinan setidaknya satu kartu yang dipegang player 2 yang dapat mengalahkan kartu di meja. Berikut line code nya:
   ```
   if max(list(p2.values())) < list(cards_on_the_table.values())[-1]:
   ```
   sample koding di atas akan melihat kartu dengan nilai paling tinggi di tangan player 2 dan mengecek apakah nilai tersebut lebih kurang dari kartu terakhir di meja(makanya pakai index [-1]). Jika *if statement* tersebut bernilai *True* berarti player 2 otomatis tidak bisa membuang kartu apapun pada perputaran saat itu karena bahkan kartu tertinggi di tangan player 2 tidak bisa mengalahkan kartu yang ada di meja, maka giliran player 2 saat itu akan di skip dan harus menunggu perputaran selanjutnya. Sebaliknya jika *if statement* nya bernilai *False,* itu berarti ada setidaknya satu kartu yang dapat dibuang oleh player 2 dan giliran player 2 saat itu tidak di skip.
2. Jika *if statement* di langkah pertama bernilai *False* dan giliran player 2 tidak di skip, maka selanjutnya akan ada dictionary baru yang bersifat sementara yang isinya hanyalah sebuah *copy* dari dictionary "p2" yang berisi kartu-kartu di yangan player 2. Perbedaan dari dictionary sementara dan dictionary asli(p2) adalah dictionary sementara kartu-kartunya akan disortir mulai dari kartu terrendah hingga kartu tertinggi (*ascending*).
3. Setelah dictionary sementara di langkah kedua selesai disortir, maka program akan melakukan iterasi pada dictionary sementara untuk mengecek satu-satu jika ada kartu yang dapat mengalahkan kartu yang ada di meja. Berikut sample codingannya:
   ```
   for element in temp_dict_sorted:
       if p2[element] < list(cards_on_the_table.values())[-1]:
           pass
       elif p2[element] > list(cards_on_the_table.values())[-1]:
           put = element
           break

   cards_on_the_table[put] = p2[put]
   print(f"{bwhite}{Black}player 2 throws {put}{res}")
   p2.pop(put)
   time.sleep(0.1)
   ```
   karena dictionary sementara sudah di sortir secara *ascending* dari awal, dan iterasi *for loop* pasti mulai dari index awal, maka pada saat iterasi mendapatkan kartu yang dapat mengalahkan kartu yang ada di meja, kartu yang akan dibuang sudah otomatis merupakan kartu paling kecil yang ada di tangan player 2 tapi cukup untuk mengalahkan kartu yang ada di meja, dengan kata lain : kartu yang akan dibuang yang didapatkan dari hasil iterasi sudahlah merupakan pilihan kartu paling optimal setidaknya untuk perputaran saat itu.