Aplikasi Kombinatorika dalam Menentukan Strategi Paling Optimal pada Permainan Rummikub

Rafif Farras - 13523095[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]rafiffarras023@gmail.com, 13523095@std.stei.itb.ac.id

***Abstrak*—Makalah ini membahas mengenai aplikasi kombinatorika pada strategi optimasi dalam permainan Rummikub, di mana pemain berusaha membentuk kombinasi ubin yang valid dan optimal untuk memenangkan permainan. Melalui analisis terhadap 150 percobaan, ditemukan bahwa pemain 1 yang menggunakan algoritma berbasis kombinatorika hanya berhasil menang sebanyak 77 kali (51.33%), mencerminkan efektivitas algoritma yang masih perlu ditingkatkan. Faktor-faktor yang mempengaruhi hasil, termasuk pembobotan nilai dalam evaluasi langkah, kompleksitas permainan, dan unsur keberuntungan, diidentifikasi sebagai elemen kunci yang mempengaruhi performa pemain. Penyesuaian bobot dalam evaluasi langkah dan pengembangan algoritma yang lebih adaptif disarankan untuk meningkatkan efektivitas strategi optimasi. Makalah ini memberikan wawasan mengenai aspek strategis dalam Rummikub dan membuka peluang untuk penelitian lebih lanjut dalam pengembangan strategi permainan yang lebih baik.**

***Kata Kunci*—kombinatorika, rummikub, strategi optimasi, efektivitas algoritma**

I. PENDAHULUAN

Di dunia serba komputasi saat ini, penerapan matematika diskrit, khususnya kombinatorika, banyak digunakan untuk mengoptimalkan strategi dalam permainan. Salah satu penerapan yang menarik dari kombinatorika ini dapat ditemukan dalam permainan Rummikub. Rummikub adalah sebuah permainan papan yang menggabungkan elemen dari permainan kartu remi dan mahjong. Rummikub pertama kali diciptakan oleh Ephraim Hertzano di tahun 1930-an. Permainan ini menggabungkan kemampuan strategis dan analisis berbagai kombinasi yang mungkin dan memilih *move* yang paling optimal untuk memperbesar kemungkinan menang.

Gambar 1: Bentuk Fisik Permainan Rummikub
(Sumber: https://reviewapasaja.net/2020/03/30/rummikub/)

Dalam permainan Rummikub, diperlukan pengambilan keputusan strategis yang seringkali melibatkan analisis terhadap berbagai kemungkinan kombinasi ubin yang dapat dimainkan. Setiap giliran, pemain dihadapkan dengan banyak kemungkinan *move* yang membutuhkan pertimbangan matematis untuk menentukan langkah optimal. Semakin banyak ubin di tangan pemain, maka semakin banyak kemungkinan *move* yang bisa dilakukan. Hal ini membuka peluang untuk menerapkan konsep kombinatorika dalam menentukan *move* paling optimal yang pastinya akan berdampak pada peluang pada giliran selanjutnya.

Penelitian ini bertujuan untuk mengaplikasikan prinsip-prinsip kombinatorika dalam mengembangkan strategi optimal untuk permainan Rummikub. Dengan membandingkan dua pendekatan berbeda yaitu strategi berbasis kombinatorika (mempertimbangkan *next move*) dan strategi dasar (melakukan *move* yang pertama kali di temukan tanpa mempertimbangkan adanya kemungkinan *move* lain), penelitian ini ingin melihat adanya pengaruh dalam penerapan konsep kombinatorika yang dapat meningkatkan peluang pemain dalam memenangkan permainan. Melalui simulasi program yang menjalankan ratusan permainan sekaligus, penelitian ini juga bertujuan untuk memvalidasi strategi berbasis kombinatorika yang penulis ambil dalam optimasi permainan Rummikub ini.

II. LANDASAN TEORI

A. Kombinatorika

Kombinatorika adalah cabang matematika untuk menghitung (*counting*) jumlah penyusunan objek-objek tanpa harus mengenumerasi semua kemungkinan susunannya. Bidang ini mencakup studi tentang cara menghitung banyaknya kemungkinan susunan objek dalam suatu himpunan, baik dengan memperhatikan urutan maupun tidak. Konsep-konsep dalam kombinatorika memiliki aplikasi luas dalam berbagai bidang, mulai dari teori probabilitas hingga optimasi dan ilmu komputer.

Beberapa konsep fundamental dalam kombinatorika meliputi:

1. Kaidah Perkalian

Prinsip ini menyatakan bahwa jika suatu kejadian A dapat terjadi dalam m cara yang berbeda, dan kejadian B dapat terjadi dalam n cara yang berbeda, maka kedua kejadian tersebut dapat terjadi bersama dalam $m \times n$ cara.

2. Kaidah Penjumlahan

Prinsip ini menyatakan bahwa jika suatu kejadian *A* dapat terjadi dalam *m* cara yang berbeda, dan kejadian *B* dapat terjadi dalam *n* cara yang berbeda, maka banyak kemungkinan kejadian A atau kejadian B dapat terjadi dalam *m* + *n* cara.

3. Permutasi

Permutasi adalah susunan terurut dari beberapa atau semua anggota suatu himpunan. Dalam permutasi, urutan berpengaruh. Konsep permutasi sangat relevan dalam menganalisis kemungkinan susunan yang valid, terutama ketika urutan mempengaruhi kevalidan suatu kombinasi. Rumus umum permutasi adalah sebagai berikut:

$$P(n,r) = n(n-1)(n-2)...(n-(r-1)) = \frac{n!}{(n-r)!}$$

4. Kombinasi

Kombinasi merupakan cara memilih sejumlah objek dari sekumpulan objek tanpa memperhatikan urutan. Konsep ini menjadi dasar dalam analisis berbagai kemungkinan pemilihan objek ketika urutan tidak relevan. Rumus umum kombinasi adalah sebagai berikut:

$$\frac{n(n-1)(n-2)...(n-(r-1))}{r!} = \frac{n!}{r!(n-r)!} = C(n,r)$$

5. Peluang Diskrit

Peluang suatu kejadian *A* dinotasikan sebagai *P(A)*. Peluang merupakan perbandingan antara titik sampel dengan kemungkinan seluruh kejadian. Suatu peluang memiliki *range* $0 \leq P(A) \leq 1$. Jika dalam suatu percobaan dapat menghasilkan *N* macam hasil dan terdapat sebanyak *n* hasil dari *N* yang berkaitan dengan kejadian *A*, maka peluang kejadian *A* adalah:

$$P(A) = \frac{n}{N}$$

B. Rummikub

Rummikub adalah permainan papan yang menggabungkan strategi dan keberuntungan, dimainkan dengan ubin berwarna yang memiliki nilai dan jenis. Dikenalkan pada tahun 1930-an, permainan ini biasanya terdiri dari 2 hingga 6 pemain yang diawal memiliki 14 ubin. Tujuan permainan adalah membentuk kombinasi ubin yang valid, baik dalam bentuk *group* maupun *run*. Rummikub menuntut keterampilan strategis dalam mengatur dan memanfaatkan ubin, menjadikannya permainan yang menarik dan menantang.

1. Komponen Permainan

Rummikub dimainkan dengan beberapa komponen fisik. Permainan dasar menggunakan 106 ubin yang terdiri dari 104 ubin bernomor (8 set lengkap dari angka 1-13 dalam empat warna berbeda: merah, biru, kuning, dan hitam) dan 2 joker. Setiap pemain menggunakan rak dengan penyangga untuk menyusun ubinnya. Untuk akomodasi permainan dengan 5-6 pemain, jumlah ubin ditingkatkan menjadi 160 buah (12 set) untuk memastikan ketersediaan ubin yang cukup bagi semua pemain.

2. Tujuan Permainan

Esensi dari Rummikub terletak pada objektifnya yang sederhana namun menantang, pemain berlomba menjadi pemain pertama yang berhasil menghabiskan seluruh ubin di rak mereka. Hal ini dicapai dengan membentuk kombinasi-kombinasi valid dari ubin-ubin tersebut. Kesederhanaan tujuan ini sangat berbanding tebalik dengan kompleksnya strategi yang diperlukan untuk mencapainya.

3. Jenis Set yang Valid

Pada permainan Rummikub terdapat dua jenis kombinasi ubin yang valid, yaitu:

- *Group*: Merupakan kumpulan tiga atau empat ubin dengan angka identik namun warna berbeda, mencerminkan konsep kombinasi dalam matematika.
- *Run*: Merupakan rangkaian tiga atau lebih ubin berurutan dengan warna sama.

Pembatasan angka 1 sebagai nilai terendah dan tidak dapat mengikuti angka 13 menciptakan batasan logis dalam pembentukan run.

4. Persiapan Permainan

Tahap persiapan merupakan tahapan penting untuk permainan ini. Setiap pemain memulai dengan 14 ubin yang diambil secara acak. Sisa ubin membentuk "*pool*" yang berfungsi sebagai sumber ubin tambahan selama permainan. Penentuan pemain pertama berdasarkan nilai ubin tertinggi yang semakin memperlihatkan kalau Rummikub memadukan elemen keberuntungan yang seimbang dengan strategi.

Gambar 2: Permainan Rummikub yang Sedang Berlangsung (Sumber: https://reviewapasaja.net/2020/03/30/rummikub/)

5. Aturan Dasar

Pada awal permainan, terdapat konsep "*initial meld*". Konsep ini mengharuskan gerakan pertama yang dilakukan setiap pemain menghasilkan minimal 30 poin. Poin ini merupakan jumlah dari ubin yang dikeluarkan setiap gilirannya. Pada gerakan pertama diperbolehkan memasukkan lebih dari 1 set yang valid sehingga bisa mencapai minimal 30 poin. Sedangkan, jika bukan kondisi "*initial meld",* pemain hanya diperbolehkan memainkan 1 set. Disinilah strategi setiap pemain diuji. Pemain harus memikirkan kombinasi dengan nilai paling tinggi dan juga peluang susunan ubin di giliran selanjutnya. Pemain yang tidak memiliki set yang valid harus mengambil 1 ubin dari *pool.* Ubin yang baru diambil tidak bisa langsung dimainkan. Pada setiap gilirannya, pemain diberi waktu 1 menit yang semakin membuat permainan ini menarik.

6. Manipulasi Set

Bagian ini yang membedakan Rummikub dari permainan serupa. Pemain dapat:

- Menata ulang set yang sudah ada di meja.
- Menambahkan ubin ke set yang sudah ada.
- Memecah set menjadi kombinasi baru.
- semua set tetap harus validdi akhir giliran.

7. Aturan Joker

Joker dalam Rummikub memiliki peran strategis yang unik. Joker berfungsi sebagai wild card yang dapat menggantikan ubin apapun. Tetapi, joker juga memiliki penalti yang signifikan (-30 poin) jika ubin joker masih tersisa pada board pemain di akhir permainan.

8. Perhitungan Skor

Pemenang akan mendapat poin sebanyak jumlah total poin ubin yang ada pada rak lawan. Sedangkan pemain yang kalah, skor yang mereka dapatkan dikali -1 sehingga mereka punya skor negatif.

9. Kondisi Menang

Kondisi kemenangan pada permainan dapat terjadi pada dua kondisi berikut:

- Kondisi utama: menghabiskan semua ubin dan pemain menyebutkan "Rummikub!"
- Kondisi alternatif saat pool habis: pemain dengan nilai ubin terendah langsung menjadi pemenang.

Rummikub biasanya dimainkan dalam beberapa ronde. Kemenangan keseluruhan ditentukan oleh akumulasi game dalam ronde.

III. PEMBAHASAN

A. Perancangan Kode Simulasi Rummikub

Perlu menjadi catatan bahwa penulis pada penerapan kode ini hanya menggunakan 2 pemain untuk melihat apakah ada perbedaan yang signifikan pada 2 strategi yang diterapkan pada kedua pemain. Strategi pertama adalah menggunakan kombinatorika untuk mencari *optimal move*, sedangkan strategi kedua akan melakukan *move* yang pertama kali ditemukan yang sesuai dengan aturan Rummikub. Implementasi kartu joker juga dihilangkan agar algoritma optimasi yang digunakan dapat lebih terlihat dampaknya.

Penulis mengimplementasikannya pada 5 Class sebagai berikut:

1. Struktur Data Dasar (Class Tile)

Implementasi dimulai dengan perancangan struktur data dasar berupa class Tile yang merepresentasikan ubin dalam permainan Rummikub. Class ini diimplementasikan menggunakan Python dataclass untuk efisiensi penulisan kode. Setiap ubin memiliki dua atribut utama yaitu number (angka 1-13) dan color (warna: Red, Blue, Yellow, Black). Class ini juga dilengkapi dengan method **str** untuk representasi string yang memudahkan visualisasi, serta method **eq** dan **hash** untuk mendukung operasi perbandingan dan penggunaan ubin dalam struktur data set. Format string yang digunakan menampilkan huruf pertama warna diikuti angka, misalnya "R12" untuk ubin merah bernomor 12.

2. Manajemen Set Ubin (Class RummikubSet)

RummikubSet merupakan class yang mengelola kumpulan ubin yang membentuk kombinasi valid dalam permainan. Konstruktor class ini menerima list of Tile dan mengurutkannya berdasarkan nomor dan warna untuk konsistensi. Method is_valid() mengimplementasikan logika untuk memvalidasi apakah sekumpulan ubin dapat membentuk kombinasi yang valid menurut aturan Rummikub. Validasi dilakukan untuk dua jenis kombinasi: *run* (ubin berurutan dengan warna sama) dan *group* (ubin dengan nomor sama tapi warna berbeda).

3. Permainan Utama (Class RummikubGame)

RummikubGame berfungsi sebagai program utama yang mengatur jalannya permainan. Class ini menginisialisasi dan mengelola seluruh state permainan, termasuk tile_pool (kumpulan ubin yang belum diambil), board_sets (kombinasi yang sudah dimainkan di meja), dan tile tangan masing-masing pemain. Method create_tile_pool() membuat set lengkap ubin Rummikub dengan duplikasi sesuai aturan. Method draw_tiles() mengelola pengambilan ubin dari pool, sementara calculate_hand_value() menghitung nilai total ubin di tangan pemain. Class ini juga menyimpan status first_move untuk setiap pemain dan mengelola skor permainan. Implementasi *scoring* mengikuti aturan resmi Rummikub di mana pemenang mendapat poin positif dari total nilai negatif ubin yang tersisa di tangan pemain lain.

4. Analisis Strategi (Class RummikubStrategy)

RummikubStrategy merupakan komponen paling kompleks yang mengimplementasikan logika pengambilan keputusan untuk strategi optimal. Class ini memiliki beberapa method utama. Method group_tiles() mengatur ubin berdasarkan nomor dan warna untuk memudahkan analisis kombinasi.

```
def group_tiles(self, tiles: List[Tile]) -> Tuple[Dict[int, List[Tile]],
Dict[str, List[Tile]]]:
    by_number = {}
    by_color = {}
    for tile in tiles:
        by_number.setdefault(tile.number, []).append(tile)
        by_color.setdefault(tile.color, []).append(tile)
    return by_number, by_color
```

calculate_needed_tiles() mengidentifikasi ubin yang diperlukan untuk melengkapi kombinasi potensial, baik untuk *run* maupun *group* dengan memperhitungkan ubin yang ada pada pemain dan yang ada pada pool. Ini digunakan untuk menentukan kemungkinan di masa depan. Jika needed_tiles banyak, maka kemungkinan di masa depan untuk mendapatkannya juga semakin besar dan bisa menjadi pertimbangan untuk melakukan *move* pada giliran tertentu.

```
def calculate_needed_tiles(self, tile: Tile, remaining_tiles: List[Tile],
                           played_tiles: Set[Tile], pool_tiles: List[Tile])
-> List[Tile]:
    needed_tiles = []

    # run
    if tile.number > 1:
        needed_tiles.append(Tile(tile.number - 1, tile.color))
    if tile.number < 13:
        needed_tiles.append(Tile(tile.number + 1, tile.color))

    # group
    for color in RummikubGame.COLORS:
        if color != tile.color:
            needed_tiles.append(Tile(tile.number, color))

    # Return tiles yang di pool
    return [t for t in needed_tiles if t not in played_tiles and
            t not in remaining_tiles and any(t == pt for pt in pool_tiles)]
```

calculate_combinations() menghitung jumlah kombinasi yang mungkin dibentuk dari sekumpulan ubin, mempertimbangkan baik *run* ataupun *group*. Setelah itu, semua kombinasi yang mungkin ini akan diperhitungkan mana langkah yang paling optimal berdasarkan pendekatan kombinatorika yang penulis pilih.

```
def calculate_combinations(self, tiles: List[Tile]) -> int:
    by_number, by_color = self.group_tiles(tiles)
    combinations_count = 0

    # Count potential sets
    for number, number_tiles in by_number.items():
        n = len(number_tiles)
        if n >= 2:
            combinations_count += (n * (n-1)) // 2  # nC2

    # Count potential runs
    for color, color_tiles in by_color.items():
        numbers = sorted([t.number for t in color_tiles])
        consecutive = 1
        for i in range(1, len(numbers)):
            if numbers[i] == numbers[i-1] + 1:
                consecutive += 1
            else:
                if consecutive >= 2:
                    combinations_count += consecutive
                consecutive = 1
        if consecutive >= 2:
            combinations_count += consecutive

    return combinations_count
```

calculate_flexibility() menganalisis fleksibilitas dari sekumpulan ubin yang tersisa di board, memberikan skor berdasarkan potensi pembentukan kombinasi di masa depan. Khusus untuk *run*, method ini akan memberikan bobot lebih tinggi daripada *group* karena dalam konsep kombinatorika *run* dapat menciptakan permainan yang lebih fleksibel disbanding *group*. Run memungkinkan perpanjangan dari urutan yang sudah ada, yang berpotensi menghasilkan skor lebih tinggi dan lebih banyak opsi untuk langkah berikutnya, sementara *group* terbatas pada set ubin yang tetap.

calculate_flexibility() menggunakan dua komponen utama dalam perhitungannya, penilaian untuk *group* menggunakan formula nC2 yang dimodifikasi, dan penilaian untuk *run* dengan bobot 1.5 kali dari panjang rangkaian terpanjang. Untuk *group*, formula (n * (n-1)) / 2 memberikan nilai yang meningkat secara non-linear sesuai jumlah ubin dengan nomor sama, mencerminkan potensi pembentukan kombinasi yang semakin besar. Sementara untuk *run* penggunaan faktor pengali 1.5 memberikan prioritas lebih tinggi, mengingat fleksibilitas dan potensi pengembangannya yang lebih besar. Kombinasi kedua komponen ini menghasilkan sistem penilaian yang seimbang sesuai dengan peluang masing-masing dan mendukung pengambilan keputusan strategis dalam permainan.

```
def calculate_flexibility(self, tiles: List[Tile]) -> float:
    by_number, by_color = self.group_tiles(tiles)
    flexibility_score = 0

    # Score untuk group
    for number, number_tiles in by_number.items():
        n = len(number_tiles)
        if n >= 2:
            flexibility_score += (n * (n-1)) / 2

    # Score untuk run
    for color, color_tiles in by_color.items():
        numbers = sorted([t.number for t in color_tiles])
        consecutive = 1
        max_consecutive = 1
        for i in range(1, len(numbers)):
            if numbers[i] == numbers[i-1] + 1:
                consecutive += 1
                max_consecutive = max(max_consecutive, consecutive)
            else:
                consecutive = 1
        flexibility_score += max_consecutive * 1.5  #

    return flexibility_score
```

evaluate_move() adalah method kunci yang mengevaluasi kualitas suatu langkah berdasarkan tiga factor: direct_value dari kombinasi yang dibentuk, potential_value dari ubin yang tersisa, dan flexibility_bonus untuk mempertahankan opsi di masa depan.

```
def evaluate_move(self, hand: List[Tile], possible_move: List[RummikubSet],
                  pool_size: int, played_tiles: Set[Tile], pool_tiles:
List[Tile]) -> float:
    # Direct Value
    direct_value = sum(tile.number for set_obj in possible_move
                       for tile in set_obj.tiles)

    used_tiles = {tile for set_obj in possible_move for tile in
set_obj.tiles}
    remaining_tiles = [t for t in hand if t not in used_tiles]

    # Potential Value
    potential_value = 0
    for tile in remaining_tiles:
        needed_tiles = self.calculate_needed_tiles(tile, remaining_tiles,
                                                   played_tiles,
pool_tiles)

        if needed_tiles:
            available = len(needed_tiles)
            p_completion = available / pool_size if pool_size > 0 else 0
            combinations_possible= self.calculate_combinations(needed_tiles)
            potential_value += p_completion * tile.number *
(combinations_possible + 1)

    # Flexibility
    flexibility_bonus = self.calculate_flexibility(remaining_tiles)

    # Combine all factors with weights
    final_score = (direct_value * 1.0 +
                   potential_value * 0.8 +
                   flexibility_bonus * 0.5)

    return final_score
```

Dalam method evaluate_move(), implementasi pembobotan dirancang untuk menyeimbangkan tiga aspek penting dalam pengambilan keputusan yaitu nilai ubin (direct_value) dengan bobot 1.0, nilai potensial (potential_value) dengan bobot 0.8, dan bonus fleksibilitas (flexibility_bonus) dengan bobot 0.5. Bobot tertinggi diberikan pada nilai langsung karena merepresentasikan keuntungan yang pasti, sedangkan nilai potensial dan fleksibilitas diberi bobot lebih rendah karena sifatnya yang spekulatif dan bergantung pada perkembangan permainan. Pembobotan ini mencerminkan prinsip bahwa nilai pasti lebih diutamakan dibanding potensi masa depan, namun tetap memberikan pertimbangan untuk peluang pengembangan kombinasi selanjutnya.

Terakhir, method find_all_valid_sets dan find_optimal_move bekerja secara berpasangan untuk mengoptimalkan strategi permainan. Find_all_valid_sets menggunakan modul itertools.combinations untuk menghasilkan semua kemungkinan kombinasi ubin dengan panjang 3 hingga maksimal, kemudian memvalidasi setiap kombinasi menggunakan method is_valid() untuk memastikan kombinasi tersebut memenuhi aturan Rummikub baik sebagai *group* maupun *run*. Selanjutnya, find_optimal_move mengambil hasil dari find_all_valid_sets dan melakukan evaluate_move() dari ubin yang memenuhi syarat untuk menentukan *optimal move* jika lebih dari 1 kemungkinan.

5. Program Utama (Play Game)

Method play_game() mengimplementasikan *loop* utama permainan Rummikub yang mengatur jalannya permainan antara dua pemain dengan strategi berbeda. Setiap iterasi, method ini mengelola giliran pemain, menampilkan status permainan (jumlah ubin di pool dan set di meja), dan mengeksekusi strategi yang berbeda untuk masing-masing pemain. Pemain pertama menggunakan strategi optimal berbasis kombinatorika melalui find_optimal_move yang mengevaluasi semua kemungkinan langkah terbaik, sedangkan pemain kedua menggunakan strategi dasar yang hanya mencari dan memainkan set valid pertama yang ditemukan.

```
if current_player == 1:
    # Optimal combinatorics-based strategy
    optimal_sets, remaining_tiles = strategy.find_optimal_move(
        hand, game.board_sets, len(game.tile_pool),
        game.played_tiles, game.tile_pool, is_first_move
    )
            ...
            ...
            ...
else:
    # Basic strategy
    possible_sets = strategy.find_all_valid_sets(hand)
            ...
            ...
            ...
```

Method ini juga mengatur kondisi akhir permainan (saat satu pemain menghabiskan ubin atau pool kosong), melakukan perhitungan skor akhir sesuai aturan resmi Rummikub, dan menentukan pemenang berdasarkan skor tersebut.

B. Pengujian

Pada pengujian saya menjalankan program dan menganalisis output dari setiap giliran pada pemain. Berikut adalah output/simulasi programnya:

1. Program Awal

Memberikan 14 random ubin ke setiap pemain. Ketika tidak ada *move* yang bisa dilakukan, player mengambil ubin dari *pool.*

```
=== Starting Rummikub Game ===
Player 1 uses optimal combinatorics-based strategy
Player 2 uses basic strategy

=== Player 1's turn ===
Current scores - Player 1: 0 | Player 2: 0
Tiles in pool: 76
Current hand: R3 B5 Y13 B3 B11 B8 R11 Y10 R1 B1 R7 B2 B12 B9
```

```
Board sets: 0

Analyzing possible moves...
Found 0 possible valid sets

Player 1 draws a tile

=== Player 2's turn ===
Current scores - Player 1: 0 | Player 2: 0
Tiles in pool: 75
Current hand: B8 B4 Y13 R2 R4 R8 B4 B8 Y6 R8 B12 B1 R12 B5
Board sets: 0
Player 2 draws a tile
```

2. Initial Meld

Pada gerakan pertama, poin harus minimal 30. Berikut contoh ketika ada yang memenuhi kondisi Initial Meld.

```
=== Player 2's turn ===
Current scores - Player 1: 0 | Player 2: 0
Tiles in pool: 53
Current hand: B8 B4 Y13 R2 R4 R8 B4 B8 Y6 R8 B12 B1 R12 B5 B11 Y4 B2 B11 B12 B10 Y5 B7 R13 Y7
B13
Board sets: 0

Player 2 plays (Score: +39): B13 R13 Y13
```

Berikut contoh Ketika ada kombinasi yang membentuk sets atau groups, tetapi tidak bisa memenuhi initial meld yang ada syarat minimal 30 poin.

```
=== Player 1's turn ===
Current scores - Player 1: 0 | Player 2: 0
Tiles in pool: 58
Current hand: R3 B5 Y13 B3 B11 B8 R11 Y10 R1 B1 R7 B2 B12 B9 Y10 Y2 R3 R4 Y4 Y7 Y2 B9 B4
Board sets: 0

Analyzing possible moves...
Found 1 possible valid sets

Evaluating all possible moves:
--------------------------------------------------

==================================================
No valid moves found
==================================================
Player 1 draws a tile
```

Pada contoh diatas terdapat 1 valid move yaitu R4 Y4 B4 (Poin=4+4+4=12), tetapi tidak melebihi 30 poin sehingga tidak bisa dijalankan.

Berikut Initial meld pemain 1 yang berhasil dan memperlihatkan hasil perhitungan evaluate_move.

```
=== Player 1's turn ===
Current scores - Player 1: 0 | Player 2: 48
Tiles in pool: 68
Current hand: R11 B2 R13 Y10 R2 B10 B7 R9 Y6 B1 Y9 B1 B11 Y10 B9 R7 R6 Y5 Y1
Board sets: 2
Set 1: B12 R12 Y12
Set 2: B3 B4 B5

Analyzing possible moves...
Found 2 possible valid sets

Evaluating all possible moves:
--------------------------------------------------
Move 1:
Sets: B9 R9 Y9 | B1 B1 Y1
Score: 46.89
- Direct Value: 30
- Potential Value: 11
- Flexibility Score: 16.00
*** This is the new best move! ***
==================================================
Selected best move:
Move: B9 R9 Y9 | B1 B1 Y1
Final Score: 46.89
==================================================
Player 1 plays (Score: +30):
Played set: B9 R9 Y9
Played set: B1 B1 Y1
```

3. Strategi Pemain 2

Menggunakan strategi biasa yang memainkan *move* yang pertama kali ditemukan. Pada kombinasi dibawah, seharusnya player 2 bisa mendapat poin lebih jika memainkan B5 B6 B7. Tetapi, karena *move* yang pertama kali ditemukan adalah B4 B5 B6, maka itu yang dijalankan. Strategi ini cenderung kurang optimal.

```
=== Player 2's turn ===
Current scores - Player 1: 177 | Player 2: 195
Tiles in pool: 7
Current hand: B4 B11 B4 R5 B6 B1 Y12 R13 B7 B10 R11 R5 B8 B1 B9 B5
Board sets: 19
Set 1: B9 B10 B11
Set 2: Y2 Y3 Y4
Set 3: B12 R12 Y12
Set 4: R2 R3 R4
Set 5: B1 B1 Y1
Set 6: R9 R10 R11
Set 7: B8 B8 R8
Set 8: B3 B3 R3
Set 9: B7 B7 R7
Set 10: Y8 Y9 Y10
Set 11: B12 B12 R12
Set 12: B13 R13 Y13
Set 13: B6 B6 R6
Set 14: B9 B10 B11
Set 15: B2 B2 Y2
Set 16: B2 B2 R2
Set 17: Y4 Y5 Y6
Set 18: B3 B3 Y3
Set 19: B5 B5 Y5

Player 2 plays (Score: +15): B4 B5 B6
```

4. Strategi Pemain 1

Mengevaluasi semua kemungkinan move, lalu menjalankan move yang memiliki skor paling tinggi.

```
=== Player 1's turn ===
Current scores - Player 1: 150 | Player 2: 84
Tiles in pool: 24
Current hand: B2 R13 Y10 B10 B7 Y10 R7 Y5 Y1 R9 B5 B7 Y5 B3 B13 B4 B10 B5
Board sets: 13
Set 1: B12 R12 Y12
Set 2: B3 B4 B5
Set 3: B9 R9 Y9
Set 4: B1 B1 Y1
Set 5: B6 R6 Y6
Set 6: B6 B6 R6
Set 7: B8 B8 Y8
Set 8: B11 R11 Y11
Set 9: B2 R2 Y2
Set 10: R2 R3 R4
Set 11: B1 B1 R1
Set 12: B12 B12 Y12
Set 13: B3 B3 Y3

Analyzing possible moves...
Found 3 possible valid sets

Evaluating all possible moves:
--------------------------------------------------
Move 1:
Sets: B5 B5 Y5
Score: 30.85
- Direct Value: 15
- Potential Value: 12
- Flexibility Score: 17.50
Move 2:
Sets: B3 B4 B5
Score: 29.00
- Direct Value: 12
- Potential Value: 13
- Flexibility Score: 19.00
==================================================
Selected best move:
Move: B5 B5 Y5
Final Score: 30.85
==================================================
Player 1 plays (Score: +15):
Played set: B5 B5 Y5
```

5. End Game

Pada beberapa percobaan yang saya lakukan, permainan sering berakhir karena *pool* sudah habis. Jarang permainan yang berakhir dengan habisnya ubin pada *board* salah satu pemain. Pada output dibawah juga terlihat bagaimana perhitungan skor dilakukan.

```
=== Player 2's turn ===
Current scores - Player 1: 195 | Player 2: 150
Tiles in pool: 1
Current hand: Y13 B9 B10 B7 R5 B12 Y13 B13 B6 B5 R4 B10 R10 Y9 R12 R10 R5 B9 B13 B2 B11 Y7
Board sets: 19
…
…
Player 2 draws a tile

=== Game over - Tile pool empty ===

Remaining tiles value:
Player 1: 141
Player 2: 203

Final scores:
Player 1: 203
Player 2: -203
Player 1 wins!
```

6. Percobaan 50 kali Permainan secara Parallel

Percobaan ini dilakukan dengan membuat kode yang dapat menjalankan permainan rummikub secara parallel 50 game. Hal ini bertujuan untuk menunjukkan apakah strategi benar-benar

mampu memperbesar peluang kemenangan pemain 1 yang menggunakan kombinatorika dibanding dengan player 2 yang menggunakan strategi biasa. Kekurangan dari kode ini adalah waktu eksekusi yang sangat lama.

```python
def play_multiple_games_parallel(num_games: int = 50) -> Dict:
    """Play multiple games in parallel using ProcessPoolExecutor"""
    stats = {
        'player1_wins': 0,
        'player2_wins': 0,
    }
    num_processes = os.cpu_count()

    with ProcessPoolExecutor(max_workers=num_processes) as executor:
        future_to_game = {
            executor.submit(play_game_wrapper, i): i
            for i in range(num_games)
        }

        for future in concurrent.futures.as_completed(future_to_game):
            result, game_num = future.result()

            # Update statistics

            if result['winner'] == 1:
                stats['player1_wins'] += 1
            elif result['winner'] == 2:
                stats['player2_wins'] += 1
            else:
                stats['ties'] += 1
    return stats

if __name__ == "__main__":
    random.seed(42)

    print("\n=== Running 50 games simulation in parallel ===")

    # Run games in parallel
    stats = play_multiple_games_parallel(50)
    print("\n=== Final Statistics ===")
    print(f"Total games played: 50")
    print(f"Player 1 (Optimal Strategy) wins: {stats['player1_wins']}
({stats['player1_wins']/50*100:.1f}%)")
    print(f"Player 2 (Basic Strategy) wins: {stats['player2_wins']}
({stats['player2_wins']/50*100:.1f}%)")
```

Hasil dari 3 kali run program adalah sebagai berikut:

Pertama:

```
=== Running 50 games simulation in parallel ===

=== Final Statistics ===
Total games played: 50
Player 1 (Optimal Strategy) wins: 29 (58%)
Player 2 (Basic Strategy) wins: 21 (42%)
```

Kedua:

```
=== Running 50 games simulation in parallel ===

=== Final Statistics ===
Total games played: 50
Player 1 (Optimal Strategy) wins: 23 (46%)
Player 2 (Basic Strategy) wins: 27 (54%)
```

Ketiga:

```
=== Running 50 games simulation in parallel ===

=== Final Statistics ===
Total games played: 50
Player 1 (Optimal Strategy) wins: 25 (50%)
Player 2 (Basic Strategy) wins: 25 (40%)
```

Penulis memilih hanya menjalankan program sebanyak 3 kali karena waktu eksekusi yang sangat lama.

C. Analisis Hasil

Dalam penerapan strategi optimasi pada permainan Rummikub, hasil yang diperoleh dari 150 percobaan menunjukkan bahwa pemain 1, yang menggunakan strategi berbasis kombinatorika, hanya berhasil menang sebanyak 77 kali. Persentase kemenangan ini, yang hanya sekitar 51,33%, menunjukkan bahwa algoritma yang diterapkan belum sepenuhnya efektif dalam mengoptimalkan keputusan permainan. Meskipun strategi ini dirancang untuk mengevaluasi semua kemungkinan langkah terbaik, hasil yang kurang memuaskan ini mengindikasikan adanya faktor-faktor lain yang mempengaruhi performa pemain, termasuk pembobotan poin yang digunakan dalam evaluasi langkah. Hal ini menimbulkan pertanyaan mengenai apakah pendekatan yang diambil sudah cukup komprehensif untuk menangani kompleksitas permainan Rummikub.

Salah satu aspek penting dalam analisis ini adalah pembobotan yang diterapkan dalam metode evaluate_move(). Dalam implementasi ini, nilai langsung (direct_value) dari kombinasi yang dibentuk diberikan bobot tertinggi sebesar 1.0, diikuti oleh nilai potensial (potential_value) dengan bobot 0.8, dan bonus fleksibilitas (flexibility_bonus) dengan bobot 0.5. Meskipun pendekatan ini mencerminkan prioritas pada nilai yang memegang pengaruh lebih tinggi, bobot yang lebih rendah pada nilai potensial dan fleksibilitas dapat mengakibatkan kurangnya perhatian terhadap peluang pengembangan kombinasi di masa depan. Misalnya, jika pemain 1 hanya fokus pada kombinasi yang sudah ada tanpa mempertimbangkan kemungkinan kombinasi baru yang dapat terbentuk dari ubin yang tersisa, maka mereka mungkin kehilangan kesempatan untuk mengubah arah permainan. Oleh karena itu, penyesuaian bobot yang lebih seimbang antara nilai langsung dan nilai potensial mungkin diperlukan untuk meningkatkan efektivitas strategi.

Selain pembobotan, faktor lain yang perlu diperhatikan adalah kompleksitas permainan Rummikub itu sendiri. Permainan ini melibatkan banyak variabel, termasuk kombinasi ubin yang tersedia, strategi lawan, dan kondisi permainan yang terus berubah. Meskipun algoritma berusaha untuk mengevaluasi semua kemungkinan kombinasi, ada kemungkinan bahwa beberapa situasi permainan yang lebih kompleks tidak dapat diantisipasi dengan baik oleh algoritma. Diantaranya adalah perhitungan potential_value yang hanya memperhitungkan ubin di *pool*, seharusnya memperhitungkan ubin di *pool* dan yang di lawan, karena pemain tidak tau ubi napa yang ada di lawan. Selain itu, terdapat sedikit *miss* dimana tidak memperhitungkan salah satu aturan permainan yaitu bisa memanipulasi set yang ada di board. Unsur keberuntungan juga berperan besar dalam permainan ini, di mana hasil pengambilan ubin dari pool dapat mempengaruhi langkah-langkah yang tersedia untuk pemain. Jika pemain 2 mendapatkan ubin yang lebih menguntungkan, hal ini dapat memberikan mereka keunggulan yang signifikan, sehingga mengurangi peluang kemenangan bagi pemain 1 meskipun strategi yang diterapkan sudah optimal.

Terakhir, penting untuk mempertimbangkan bahwa pengujian yang dilakukan hanya melibatkan dua pemain dengan strategi yang berbeda. Hal ini mungkin tidak mencerminkan dinamika permainan Rummikub yang lebih luas, di mana interaksi antara lebih dari dua pemain dapat menghasilkan hasil yang berbeda. Dalam konteks ini, pengembangan algoritma yang lebih canggih yang dapat mempertimbangkan strategi lawan dan memprediksi langkah-langkah mereka mungkin diperlukan untuk meningkatkan efektivitas strategi optimasi. Misalnya, algoritma dapat diadaptasi untuk melakukan simulasi terhadap berbagai strategi lawan, sehingga dapat mengantisipasi langkah-langkah yang mungkin diambil oleh pemain lain. Dengan memperbaiki pembobotan, mempertimbangkan kompleksitas permainan, dan mengadaptasi algoritma untuk situasi yang lebih dinamis, diharapkan hasil yang lebih baik dapat dicapai dalam percobaan selanjutnya.

IV. KESIMPULAN

Dalam makalah ini, telah dibahas penerapan strategi optimasi dalam permainan Rummikub dengan menggunakan dua pendekatan yang berbeda. Hasil dari 150 percobaan menunjukkan bahwa meskipun pemain 1 yang menggunakan strategi berbasis kombinatorika berhasil menang sebanyak 77 kali, persentase kemenangan yang hanya sekitar 51,33%

mengindikasikan bahwa algoritma yang diterapkan belum sepenuhnya efektif. Beberapa faktor, seperti pembobotan nilai dalam evaluasi langkah, kompleksitas permainan, dan unsur keberuntungan, telah diidentifikasi sebagai elemen yang mempengaruhi hasil akhir.

Dari analisis yang dilakukan, dapat disimpulkan bahwa pembobotan yang diterapkan dalam metode evaluate_move() perlu ditinjau kembali untuk menciptakan keseimbangan yang lebih baik antara nilai langsung, nilai potensial, dan fleksibilitas. Selain itu, penting untuk mempertimbangkan kompleksitas permainan Rummikub yang melibatkan banyak variabel dan interaksi antar pemain. Unsur keberuntungan juga harus diakui sebagai faktor yang dapat mempengaruhi hasil, sehingga strategi yang lebih adaptif dan responsif terhadap kondisi permainan dapat dikembangkan.

Sebagai saran, pengembangan algoritma yang lebih canggih dan adaptif sangat dianjurkan untuk meningkatkan efektivitas strategi optimasi. Hal ini dapat mencakup penyesuaian bobot dalam evaluasi langkah, serta penerapan simulasi untuk memprediksi langkah-langkah lawan. Selain itu, melakukan pengujian dengan lebih dari dua pemain dapat memberikan wawasan yang lebih baik mengenai dinamika permainan dan interaksi antar strategi. Dengan langkah-langkah ini, diharapkan hasil yang lebih baik dapat dicapai dalam percobaan selanjutnya, serta memberikan kontribusi yang lebih signifikan terhadap pengembangan strategi dalam permainan Rummikub.

V. UCAPAN TERIMAKASIH

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa yang telah memberikan karunia, sehingga penulis dapat menyelesaikan makalah yang berjudul "Aplikasi Kombinatorika dalam Menentukan Strategi Paling Optimal pada Permainan Rummikub" yang selesai tepat pada waktunya. Penulis juga mengucapkan terima kasih kepada diri sendiri yang telah berhasil menyelesaikan makalah ini dengan baik. Penulis mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, MT. sebagai dosen pengampu yang sekaligus telah memberikan referensi dan sumber pembelajaran melalui situs beliau. Terakhir, penulis mengucapkan terima kasih kepada orang tua, keluarga, teman-teman terdekat, dan seluruh pihak yang membantu penulis dalam menyelesaikan makalah ini.

REFERENSI

[1] Rinaldi Munir, “Kombinatorika (Bagian 1) ”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/18-Kombinatorika-Bagian1-2024.pdf. [Diakses 5 Januari 2025 pukul 13.38.]

[2] Rinaldi Munir, “Kombinatorika (Bagian 2) ”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/19-Kombinatorika-Bagian2-2024.pdf. [Diakses 5 Januari 2025 pukul 14.20.]

[3] "How To Play - Rummikub," Rummikub. [Online]. Available: https://rummikub.com/rules/. [Diakses 5 Januari 17.21].

[4] "Rummikub Algorithm," Stack Exchange, 2021. [Online]. Available: https://cs.stackexchange.com/questions/85954/rummikub-algorithm. [Diakses 5 Januari 19.30].

[5] "Rummikub Algorithm," Stack Overflow, 2013. [Online]. Available: https://stackoverflow.com/questions/19002692/rummikub-algorithm. [Diakses 5 Januari 19.43].

[6] J. N. van Rijn, F. W. Takes, and J. K. Vis, "The Complexity of Rummikub Problems," Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands, 2023. [Online]. Available: https://github.com/erogluegemen/101/files/10107449/rummikub.pdf

[7] D. den Hertog and P. B. Hulshof, "Solving Rummikub Problems by Integer Linear Programming," *The Computer Journal*, vol. 49, no. 6, pp. 665-669, 2006. [Online]. Available: https://github.com/erogluegemen/101/files/10107439/solving.rummikub.problems.by.integer.linear.programming.1.pdf

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 5 Januari 2025

Rafif Farras
13523095