Implementasi Algoritma Brute Force dalam menentukan langkah terbaik dalam permainan *Cascadia*

Matthew Vladimir Hutabarat - 13522093
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): vladimirmatthew791@gmail.com

***Abstract*—Cascadia menjadi permainan papan telah menjadi permainan yang banyak disukai oleh banyak orang. Dalam permainan papan melibatkan proses membuat keputusan untuk mendapatkan hasil yang maksimal disetiap giliran. Algoritma Brute Force dapat membantu pemain untuk menentukan langkah paling optimal. Makalah ini akan membahas implementasi pencarian solusi Cascadia menggunakan Brute Force**

Keywords—Cascadia, Brute Force, Solusi terbaik

I. PENDAHULUAN

Permainan papan atau biasa disebut dengan *boardgame* merupakan permainan yang melibatkan lebih dari satu pemain dan dimainkankan diatas papan sebagai media utama. Permainan papan sering kali menggunakan berbagai komponen seperti kartu, dadu, token, atau bidak yang dijadikan alat berinteraksi datas papan. Permainan papan mempunyai banyak jenis permainan dan cara bermain yang berbeda-beda. Salah satu contoh permainan papan ialah *Cascadia.*

Cascadia merupakan permainan papan yang dapat dimainkan oleh 1-4 orang. Tujuan dari permainan ini ialah mencetak skor sebanyak mungkin sehingga skor pemain lebih tinggi dibandingkan pemain lainnya. Permainan ini membutuhkan kecermatan dalam membuat keputusan. Setiap langkah yang diputuskan pemain akan menentukan seberapa banyak skor akhir yang akan diraih oleh pemain.

Salah satu algoritma yang dapat dipakai dalam membuat keputusan pada permainan ini ialah algoritma *brute force*. Dengan algoritma *brute force*. Pemain dapat membuat pilihan yang paling menguntungkan dengan mencoba setiap kemungkinan dan meminimalisir kerugian.

Gambar 1. Contoh permainan *Cascadia (sumber : https://i.redd.it/chj3pky2hfga1.jpg)*

II. LANDASAN TEORI

A. Permainan Cascadia

Permainan Cascadia adalah sebuah permainan *tile-placement* berbasis giliran. Permainan ini mempunyai banyak komponen seperti berikut.

1. Habitat Tiles, yang terdiri dari 5 habitat yaitu *Mountains, Forests, Prairies, Wetlands,* dan *Rivers.* Setiap tiles bisa teridiri satu atau dua habitat dan setiap habitat dapat ditempati oleh 1-3 jenis hewan namun habitat hanya dapat menempati satu hewan saja.
2. Wildlife Token, yang terdiri dari 5 jenis hewan seperti *Bear, Elk, Salmon, Hawk,* dan *Fox*. Token ini menjadi simbol bahwa Habitat tiles menampung hewan yang disimbolkan oleh token tersebut.
3. Nature Tokens, merupakan token yang didapat ketika meletakkan wildlife token ke habitat tiles yang mengandung simbol nature token. Nature Token dapat digunakan untuk mengambil habitat tiles dan wildlife token yang tidak berpasangan.

Perhitungan skor yang dipakai pada Makalah ini menggunakan *Wildlife Scoring Card* bertipe A. Contoh Penilaian dapat melihat Gambar 1 dengan penjelasan sebagai berikut.

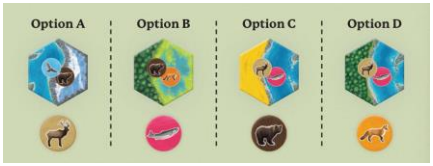
1. Mating Pairs (Bear), Setiap pasangan beruang yang tidak memiliki beruang lain di sekitarnya akan mendapat skor. Semakin banyak pasangan beruang maka semakin tinggi skor yang didapat. Detail perhitungan skor dapat dilihat pada Tabel 1.
2. Lines (Elk), Setiap kelompok rusa yang membentuk garis lurus akan mendapat skor. Setiap rusa hanya dapat membentuk satu kelompok saja sehingga tidak boleh menghitung rusa yang sama untuk dua kelompok yang berbeda. Semakin panjang kelompok maka semakin tinggi skor yang didapat. Detail perhitungan skor dapat dilihat pada Tabel 1.
3. Long Run (Salmon), setiap kelompok salmon yang membentuk barisan akan mendapatkan skor. Setiap salmon yang terdapat pada kelompok tidak boleh bersisian dengan salmon lain selain salmon yang terdapat pada kelompok itu sendiri Semakin panjang barisan maka semakin tinggi skor yang didapat. Detail perhitungan skor dapat dilihat pada Tabel 1.
4. Solitary (Hawk), setiap burung yang tidak memiliki burung lain di sekitarnya akan mendapatkan skor. Semakin tinggi banyak burung yang memenuhi kondisi tersebut maka semakin tinggi skor yang didapat. Detail penilaian skor dapat dilihat pada Tabel 1.
5. Nearby Animals (Fox), setiap rubah yang disekitarnya terdapat hewan lain akan mendapatkan skor. Semakin banyak hewan yang berbeda di sekitarnya termasuk fox itu sendiri maka semakin tinggi skor yang didapat.

	Bear	Elk	Salmon	Hawk	Fox
1	4	2	2	2	1
2	11	5	5	5	2
3	19	9	8	8	3
4	27	13	12	11	4
5	27	13	16	14	5
6	27	13	20	18	5
7	27	13	25	22	6
8+	27	13	25	26	7

Tabel 1. Tabel penilaian setiap hewan untuk pasangan / panjang kelompok / unit / hewan yang berbeda disekitarnya.

Para pemain yang memainkan *Cascadia* akan bermain bergiliran searah jarum jam. Pada setiap giliran pemain akan memilih kombinasi Habitat Tile dan Wildlife Token dan meletakkannya di lingkungan mereka. Setiap Habitat Tile dan Wildlife Token yang diambil pemain akan diletakkan tile dan token yang baru dari penyimpanan secara berurutan. Penjelasan setiap giliran dapat dilihat sebagai berikut.

1. Pada awal giliran, pemain akan menerima empat opsi Habitat Tiles dan Wildlife Token. Pemain hanya boleh memilih 1 opsi saja. Namun, pemain dapat memilih kombinasi dari opsi berbeda jika memakai Nature Token. Sebagai contoh, Pemain A memakai nature token sehingga Pemain A Memilih Habitat Tiles dari opsi A dan memilih Wildlife Token dari opsi C.



Gambar 2. Contoh empat opsi yang dapat dipilih pemain

2. Pemain diberikan kesempatan untuk mengacak ulang opsi jika ada Wildlife Token yang sama di tiga opsi yang berbeda seperti Gambar 3. Namun, Jika ada 4 Wildlife Token yang sama di empat opsi yang berbeda seperti Gambar 4. Pemain harus mengacak ulang semua opsi tersebut.

Gambar 3. Keadaaan 3 Wildlife Token yang sama

Gambar 4. Keadaaan 4 Wildlife Token yang sama

3. Pemain dapat memakai Nature Token berapapun selama persediaan pemain masih cukup.
4. Habitat Tile dan Wildlife Token yang diambil pemain harus dapat diletakkan di lingkungan pemain. Jika Habitat Tile atau Wildlife Token tidak bisa ditempatkan maka pemain harus memilih opsi lain yang memungkinkan. Jika tidak ada kemungkinan yang bisa termasuk menggunakan nature token, maka giliran tersebut akan hangus dan Habitat Tile dan Wildlife Token dikembalikan ke persediaan.
5. Habitat Tile dan Wildlife Token yang sudah ditempatkan tidak boleh dipindahkan ke tempat lain.
6. Habitat Tile harus ditempatkan bersisian dengan minimal satu habitat yang sama dengan habitat tile yang sudah ada seperti pada Gambar 5. Sebagai contoh, Habitat Tile B mempunyai habitat *River* dan *Wetlands*. Maka Tile B harus diletakkan bersisian dengan habitat *River* atau *Wetlands* minimal satu buah. Daftar kemungkinan yang tersedia dapat dilihat pada Gambar 5.

Gambar 5. Kemungkinan tile dapat diletakkan

Permainan akan berlanjut hingga tidak ada lagi Habitat Tiles yang tersisa dan perhitungan skor akan dimulai. Penjelasan perhitungan skor sebagai berikut.

1. Setiap pemain akan menghitung skor untuk hewan sesuai pada Tabel 1.
2. Setiap jenis habitat tile dengan jumlah tertinggi akan mendapat skor satu per tile
3. Pemain yang mempunyai habitat tile dengan jumlah tertinggi tiap jenis habitatnya akan mendapat bonus skor sebanyak 3. Jika 2 atau 3 pemain mendapat jumlah tertinggi maka tiap pemain tersebut akan mendapatkan bonus skor 2 dan 1 secara berurutan.
4. Nature Token yang tersisa pada akhir permainan akan dihargai 1 skor per token.

B. Algoritma Brute Force

Algoritma *brute force* merupakan algoritma yang dapat memecahkan hamper semua permasalahan dengan pendekatan *straightforward*. Algoritma *brute force* memiliki kelebihan seperti penggunaannya yang sederhana, langsung, serta memiliki cara yang jelas.

Cara kerja dari algoritma *brute force* adalah dengan mencoba semua kemungkinan yang terjadi secara berurutan kemudian mengevaluasi kemungkinan tersebut apakah solusi terbaik atau tidak. Dengan cara algoritma *brute force* yang mencoba semua kemungkinan sehingga algoritma ini pasti akan menghasilkan solusi optimal meskipun mempunyai waktu eksekusi yang lambat. Algoritma ini sering dipakai untuk persoalan yang tidak mempunyai langkah yang begitu besar.

```
procedure SelectionSort(input/output s1, s2, ..., sn : integer)
{ Mengurutkan s1, s2, ..., sn sehingga tersusun menaik dengan metode pengurutan seleksi.
  Masukan: s1, s2, ..., sn
  Luaran: s1, s2, ..., sn (terurut menaik)  }
Deklarasi
   i, j, imin, temp : integer
Algoritma:
   for i ← 1 to n – 1  do   { jumlah pass sebanyak n – 1 }
      { cari elemen terkecil di dalam s[i], s[i+1, ..., s[n] }
      imin ← i  { elemen ke-i diasumsikan sebagai elemen  terkecil sementara }
      for  j ← i+1 to n do
         if  s[j] < s[imin] then
            imin ← j
         endif
      endfor
      {pertukarkan s[imin] dengan s[i] }
      temp ← s[i]
      s[i] ← s[imin]
      s[imin] ← temp
   endfor
```

Gambar 6. Contoh Implementasi Algoritma *brute force* pada persoalan *Selection Sort*

III. IMPLEMENTASI DAN PEMBAHASAN

Pada Makalah ini, penulis menggunakan bahasa pemrograman Java. Program yang dibuat menerima input file txt yang berisikan keadaan kondisi lingkungan yang sudah ada beserta opsi opsi yang tersedia. Berikut langkah – langkah penyelesaian.

1. Pada setiap opsi yang tersedia, cari kemungkinan habitat tile dan hewan token dapat diletakkan.
2. Kemudian, untuk setiap kemungkinan tersebut evaluasi semua skor yang tercipta jika meletakkan habitat tile dan animal token di salah satu kemungkinan peletakan tersebut.
3. Simpan solusi dengan skor terbesar

```
public void Solution() {
    int maxPoint = 0;
    Pair<Tile,String> optionSolution = new Pair<Tile,String>(key:null, value:null);
    Tile tilePlacedSolution = new Tile(t:null);
    Tile animalPlacedSolution = new Tile(t:null);

    for (Pair<Tile, String> opsi : getOpsi()) {
        List<Tile> possiblePlacementTile = findPossiblePlacementTile(opsi.getTile());
        List<Tile> possiblePlacementAnimal = findPossiblePlacementAnimal(opsi.getAnimal());
        for(Tile possibleTile : possiblePlacementTile) {
            placeTile(opsi.getTile(), possibleTile.getX(), possibleTile.getY());
            for (Tile possibleAnimal : possiblePlacementAnimal) {
                placeAnimal(opsi.getAnimal(), possibleAnimal.getX(), possibleAnimal.getY());
                int pointTerritory = gradingTerritory();
                int pointBear = gradingBear();
                int pointElk = gradingElk();
                int pointSalmon = gradingSalmon();
                int pointHawk = gradingHawk();
                int pointFox = gradingFox();
                int maxLocal = pointBear + pointElk + pointSalmon + pointHawk + pointFox + pointTerritory;
                if (maxPoint < maxLocal) {
                    maxPoint = maxLocal;
                    optionSolution = opsi;
                    tilePlacedSolution = possibleTile;
                    animalPlacedSolution = possibleAnimal;
                }
                revertAnimal(opsi.getAnimal());
            }
            revertTile(opsi.getTile());
        }
    }

    System.out.println("Pilih option Habitat "+ optionSolution.getTile() +" dan Animal "+ optionSolution.getAnimal());
    System.out.println("Letakkan Habitat di ("+ tilePlacedSolution.getX() + "," +tilePlacedSolution.getY() + ")");
    System.out.println("Letakkan Animal di ("+ animalPlacedSolution.getX() + "," +animalPlacedSolution.getY() + ")");
}
```

Berikut langkah-langkah untuk penilaian Hawk :

1. Cari semua lokasi Hawk dan catat lokasinya.
2. Untuk setiap hawk, evaluasi apakah kondisi tersebut memungkinkan atau tidak untuk mendapatkan skor.
3. Hitung hawk yang memenuhi kondisi mendapatkan nilai.
4. Hitung poin yang didapatkan.

```
public int gradingHawk() {
    List<Tile> listHawk = new ArrayList<>();
    for (Tile tile : this.listTile) {
        if (tile.getAnimalPlaced().contains(s:"HAWK")) {
            listHawk.add(new Tile(tile));
        }
    }
    int countValid = 0;
    for (Tile Hawk : listHawk) {
        int currentX = Hawk.getX();
        int currentY = Hawk.getY();

        // KIRI
        if (!isEmpty(currentX - 2, currentY) && this.getTile(currentX - 2, currentY).getAnimalPlaced().contains(s:"HAWK")) {
            continue;
        }
        // KANAN
        if (!isEmpty(currentX + 2, currentY) && this.getTile(currentX + 2, currentY).getAnimalPlaced().contains(s:"HAWK")) {
            continue;
        }
        // ATAS KIRI
        if (!isEmpty(currentX - 1, currentY + 3) && this.getTile(currentX - 1, currentY + 3).getAnimalPlaced().contains(s:"HAWK")) {
            continue;
        }
        // ATAS KANAN
        if (!isEmpty(currentX + 1, currentY + 3) && this.getTile(currentX + 1, currentY + 3).getAnimalPlaced().contains(s:"HAWK")) {
            continue;
        }
        // BAWAH KIRI
        if (!isEmpty(currentX - 1, currentY - 3) && this.getTile(currentX - 1, currentY - 3).getAnimalPlaced().contains(s:"HAWK")) {
            continue;
        }
        // BAWAH KANAN
        if (!isEmpty(currentX + 1, currentY - 3) && this.getTile(currentX + 1, currentY - 3).getAnimalPlaced().contains(s:"HAWK")) {
            continue;
        }

        countValid++;
    }
```

```
        switch (countValid) {
            case 1:
                return 2;
            case 2:
                return 5;
            case 3:
                return 8;
            case 4:
                return 11;
            case 5:
                return 14;
            case 6:
                return 18;
            case 7:
                return 22;
            default:
                if (countValid >= 8) {
                    return 26;
                } else {
                    return 0;
                }

        }
    }
```

Berikut langkah-langkah untuk penilaian Fox :

1. Cari semua lokasi Fox dan catat lokasinya.
2. Untuk setiap fox, hitung banyak hewan yang berbeda disekitarnya.
3. Untuk setiap hewan yang berbeda tambahkan 1 poin

```
public int gradingFox() {
    List<Tile> listFox = new ArrayList<>();
    for (Tile tile : this.listTile) {
        if (tile.getAnimalPlaced().contains(s:"FOX")) {
            listFox.add(new Tile(tile));
        }
    }

    int point = 0;
    for (Tile Hawk : listFox) {
        List<String> listDiffAnimal = new ArrayList<>();
        int currentX = Hawk.getX();
        int currentY = Hawk.getY();
        Tile tile = new Tile(t:null);
        // KIRI
        if (!isEmpty(currentX - 2, currentY)) {
            tile = new Tile(getTile(currentX - 2, currentY));
            if (!listDiffAnimal.contains(tile.getAnimalPlaced())){
                listDiffAnimal.add(tile.getAnimalPlaced());
            }
        }
        // KANAN
        if (!isEmpty(currentX + 2, currentY)) {
            tile = new Tile(getTile(currentX + 2, currentY));
            if (!listDiffAnimal.contains(tile.getAnimalPlaced())){
                listDiffAnimal.add(tile.getAnimalPlaced());
            }
        }
        // ATAS KIRI
        if (!isEmpty(currentX - 1, currentY + 3)) {
            tile = new Tile(getTile(currentX - 1, currentY + 3));
            if (!listDiffAnimal.contains(tile.getAnimalPlaced())){
                listDiffAnimal.add(tile.getAnimalPlaced());
            }
        }
```

```
        // ATAS KANAN
        if (!isEmpty(currentX + 1, currentY + 3)) {
            tile = new Tile(getTile(currentX + 1, currentY + 3));
            if (!listDiffAnimal.contains(tile.getAnimalPlaced())){
                listDiffAnimal.add(tile.getAnimalPlaced());
            }
        }
        // BAWAH KIRI
        if (!isEmpty(currentX - 1, currentY - 3)) {
            tile = new Tile(getTile(currentX - 1, currentY - 3));
            if (!listDiffAnimal.contains(tile.getAnimalPlaced())){
                listDiffAnimal.add(tile.getAnimalPlaced());
            }
        }
        // BAWAH KANAN
        if (!isEmpty(currentX + 1, currentY - 3)) {
            tile = new Tile(getTile(currentX + 1, currentY - 3));
            if (!listDiffAnimal.contains(tile.getAnimalPlaced())){
                listDiffAnimal.add(tile.getAnimalPlaced());
            }
        }
        point += listDiffAnimal.size();

    }
    return point;
}
```

Berikut langkah-langkah untuk penilaian Bear :

1. Cari semua lokasi Bear dan catat lokasinya.
2. Untuk setiap beruang, catat juga pasangan Bear
3. Evaluasi sekitar pasangan beruang tersebut apakah terdapat beruang atau tidak
4. Catat jumlah pasangan beruang yang memenuhi syarat
5. Hitung poin yang didapatkan sesuai jumlah pasangan beruang

```
public int gradingBear() {
    List<Tile> listBear = new ArrayList<>();
    for (Tile tile : this.listTile) {
        if (tile.getAnimalPlaced().contains(s:"BEAR")) {
            listBear.add(new Tile(tile));
        }
    }
    int countValid = 0;
    for (Tile Bear : listBear) {
        Tile bearPair = findBearPair(Bear);
        if (bearPair.equals(obj:null)) {
            continue;
        }
        listBear.remove(bearPair);

        int currentX = Bear.getX();
        int currentXPair = bearPair.getX();
        int currentY = Bear.getY();
        int currentYPair = bearPair.getY();
        int countBear = 0;
        int countPair = 0;

        // KIRI
        if (!isEmpty(currentX - 2, currentY) && this.getTile(currentX - 2, currentY).getAnimalPlaced().contains(s:"BEAR")) {
            countBear++;
        }
        if (!isEmpty(currentXPair - 2, currentYPair) && this.getTile(currentXPair - 2, currentYPair).getAnimalPlaced().contains(s:"BEAR")) {
            countPair++;
        }
        // KANAN
        if (!isEmpty(currentX + 2, currentY) && this.getTile(currentX + 2, currentY).getAnimalPlaced().contains(s:"BEAR")) {
            countBear++;
        }
        if (!isEmpty(currentXPair + 2, currentYPair) && this.getTile(currentXPair + 2, currentYPair).getAnimalPlaced().contains(s:"BEAR")) {
            countPair++;
        }

        // ATAS KIRI
        if (!isEmpty(currentX - 1, currentY + 3) && this.getTile(currentX - 1, currentY + 3).getAnimalPlaced().contains(s:"BEAR")) {
            countBear++;
        }
        if (!isEmpty(currentXPair - 1, currentYPair + 3) && this.getTile(currentXPair - 1, currentYPair + 3).getAnimalPlaced().contains(s:"BEAR")) {
            countPair++;
        }
```

```
        // ATAS KANAN
        if (!isEmpty(currentX + 1, currentY + 3) && this.getTile(currentX + 1, currentY + 3).getAnimalPlaced().contains(s:"BEAR")) {
            countBear++;
        }
        if (!isEmpty(currentXPair + 1, currentYPair + 3) && this.getTile(currentXPair + 1, currentYPair + 3).getAnimalPlaced().contains(s:"BEAR")) {
            countPair++;
        }
        // BAWAH KIRI
        if (!isEmpty(currentX - 1, currentY - 3) && this.getTile(currentX - 1, currentY - 3).getAnimalPlaced().contains(s:"BEAR")) {
            countBear++;
        }
        if (!isEmpty(currentXPair - 1, currentYPair - 3) && this.getTile(currentXPair - 1, currentYPair - 3).getAnimalPlaced().contains(s:"BEAR")) {
            countPair++;
        }
        // BAWAH KANAN
        if (!isEmpty(currentX + 1, currentY - 3) && this.getTile(currentX + 1, currentY - 3).getAnimalPlaced().contains(s:"BEAR")) {
            countBear++;
        }
        if (!isEmpty(currentXPair + 1, currentYPair - 3) && this.getTile(currentXPair + 1, currentYPair - 3).getAnimalPlaced().contains(s:"BEAR")) {
            countPair++;
        }

        if (countBear == 1 && countPair == 1) {
            countValid++;
        }
    }

    switch (countValid) {
        case 1:
            return 4;
        case 2:
            return 11;
        case 3:
            return 19;
        case 4:
            return 27;
        default:
            return 0;
    }
}
```

Berikut langkah-langkah untuk penilaian Elk :

1. Cari semua lokasi Elk dan catat lokasinya.
2. Untuk setiap elk, hitung kelompok elk dengan panjang terpanjang
3. Untuk kelompok elk terpanjang, hapus elk pada daftar lokasi elk.
4. Hitung poin elk bedasarkan panjang kelompok elk yang ada

```
public int gradingElk() {
    List<Tile> listElk = new ArrayList<>();
    for (Tile tile : this.listTile) {
        if (tile.getAnimalPlaced().contains(s:"ELK")) {
            listElk.add(new Tile(tile));
        }
    }
    int point = 0;
    for (Tile Elk : listElk) {
        int currentX = Elk.getX();
        int currentY = Elk.getY();
        List<Tile> longestElk = new ArrayList<>();
        List<Tile> longestLocal = new ArrayList<>();

        // KIRI KANAN
        longestLocal.clear();
        int offsetX = 0;
        int offsetY = 0;
        while (!isEmpty(currentX - offsetX,currentY - offsetY) && getTile(currentX - offsetX, currentY - offsetY).getAnimalPlaced().equals(anObject:"ELK")) {
            longestLocal.add(getTile(currentX - offsetX, currentY - offsetY));
            offsetX += 2;
        }
        offsetX = 2;
        while (!isEmpty(currentX + offsetX,currentY - offsetY) && getTile(currentX + offsetX, currentY - offsetY).getAnimalPlaced().equals(anObject:"ELK")) {
            longestLocal.add(getTile(currentX + offsetX, currentY - offsetY));
            offsetX += 2;
        }
        if (longestElk.size() < longestLocal.size()) {
            longestElk = new ArrayList<>(longestLocal);
        }
```

```
// SERONG KANAN
longestLocal.clear();
offsetX = 0;
offsetY = 0;
while (!isEmpty(currentX - offsetX,currentY - offsetY) && getTile(currentX - offsetX, currentY - offsetY).getAnimalPlaced().equals(anObject:"ELK")) {
    longestLocal.add(getTile(currentX - offsetX, currentY - offsetY));
    offsetX += 1;
    offsetY += 3;
}
offsetX = 1;
offsetY = 3;
while (!isEmpty(currentX + offsetX,currentY + offsetY) && getTile(currentX + offsetX, currentY + offsetY).getAnimalPlaced().equals(anObject:"ELK")) {
    longestLocal.add(getTile(currentX + offsetX, currentY + offsetY));
    offsetX += 1;
    offsetY += 3;
}
if (longestElk.size() < longestLocal.size()) {
    longestElk = new ArrayList<>(longestLocal);
}

// SERONG KIRI
longestLocal.clear();
offsetX = 0;
offsetY = 0;
while (!isEmpty(currentX - offsetX,currentY + offsetY) && getTile(currentX - offsetX, currentY + offsetY).getAnimalPlaced().equals(anObject:"ELK")) {
    longestLocal.add(getTile(currentX - offsetX, currentY + offsetY));
    offsetX += 1;
    offsetY += 3;
}
offsetX = 1;
offsetY = 3;
while (!isEmpty(currentX + offsetX,currentY - offsetY) && getTile(currentX + offsetX, currentY - offsetY).getAnimalPlaced().equals(anObject:"ELK")) {
    longestLocal.add(getTile(currentX + offsetX, currentY - offsetY));
    offsetX += 1;
    offsetY += 3;
}

if (longestElk.size() < longestLocal.size()) {
    longestElk = new ArrayList<>(longestLocal);
}
```

```
        // Grading
        switch (longestElk.size()) {
            case 1:
                point += 2;
                break;
            case 2:
                point += 5;
                break;
            case 3:
                point += 9;
                break;
            case 4:
                point += 13;
                break;
            default:
                break;
        }
        listElk.removeAll(longestElk);
    }
    return point;
}
```

Berikut langkah-langkah untuk penilaian Elk :

1. Cari semua lokasi Salmon dan catat lokasinya.
2. Untuk setiap salmon, cari salmon lain yang bersisian dengannya
3. Untuk pasangan salmon tersebut. Cari kemungkinan pergerakan salmon
4. Secara rekursif, evaluasi kemungkinan pergerakan salmon dan cari kelompok salmon dengan panjang salmon terbesar.

```
public List<Pair<Integer,Integer>> possibleSalmonMove(Tile before, Tile current){
    List<Pair<Integer,Integer>> nextMove = new ArrayList<Pair<Integer,Integer>>();

    // find relation tile before and current
    int currentX = current.getX();
    int currentY = current.getY();
    if(before.getY() == current.getY() && before.getX() == current.getX() - 2) {
        // before di kiri current
        nextMove.add(new Pair<Integer,Integer>(currentX + 2, currentY)); // kanan
        nextMove.add(new Pair<Integer,Integer>(currentX + 1, currentY - 3)); // bawah kanan
        nextMove.add(new Pair<Integer,Integer>(currentX + 1, currentY + 3)); // atas kanan
        return nextMove;
    } else if(before.getY() == current.getY() && before.getX() == current.getX() + 2) {
        // before di kanan current
        nextMove.add(new Pair<Integer,Integer>(currentX - 2, currentY)); // kiri
        nextMove.add(new Pair<Integer,Integer>(currentX - 1, currentY - 3)); // bawah kiri
        nextMove.add(new Pair<Integer,Integer>(currentX - 1, currentY + 3)); // atas kiri
        return nextMove;
    } else if(before.getY() == current.getY() + 3 && before.getX() == current.getX() - 1) {
        // before di atas kiri current
        nextMove.add(new Pair<Integer,Integer>(currentX + 2, currentY)); // kanan
        nextMove.add(new Pair<Integer,Integer>(currentX + 1, currentY - 3)); // bawah kanan
        nextMove.add(new Pair<Integer,Integer>(currentX - 1, currentY - 3)); // bawah kiri
        return nextMove;
    } else if(before.getY() == current.getY() + 3 && before.getX() == current.getX() + 1) {
        // before di atas kanan current
        nextMove.add(new Pair<Integer,Integer>(currentX - 2, currentY)); // kiri
        nextMove.add(new Pair<Integer,Integer>(currentX - 1, currentY - 3)); // bawah kiri
        nextMove.add(new Pair<Integer,Integer>(currentX + 1, currentY - 3)); // bawah kanan
        return nextMove;
    } else if(before.getY() == current.getY() - 3 && before.getX() == current.getX() - 1) {
        // before di bawah kiri current
        nextMove.add(new Pair<Integer,Integer>(currentX + 2, currentY)); // kanan
        nextMove.add(new Pair<Integer,Integer>(currentX + 1, currentY + 3)); // atas kanan
        nextMove.add(new Pair<Integer,Integer>(currentX - 1, currentY + 3)); // atas kiri
        return nextMove;
    } else if(before.getY() == current.getY() - 3 && before.getX() == current.getX() + 1) {
        // before di bawah kanan current
        nextMove.add(new Pair<Integer,Integer>(currentX - 2, currentY)); // kiri
        nextMove.add(new Pair<Integer,Integer>(currentX - 1, currentY + 3)); // atas kiri
        nextMove.add(new Pair<Integer,Integer>(currentX + 1, currentY + 3)); // atas kanan
        return nextMove;
    }
```

```
public List<Tile> findNearestSalmon(Tile salmon, List<Tile> listSalmon) {
    int currentX = salmon.getX();
    int currentY = salmon.getY();

    List<Tile> nearestSalmon = new ArrayList<>();
    for(Tile tile: listSalmon) {
        // KIRI
        if (currentX - 2 == tile.getX() && currentY == tile.getY()) {
            nearestSalmon.add(tile);
            continue;
        }
        // KANAN
        if (currentX + 2 == tile.getX() && currentY == tile.getY()) {
            nearestSalmon.add(tile);
            continue;
        }
        // ATAS KIRI
        if (currentX - 1 == tile.getX() && currentY + 3 == tile.getY()) {
            nearestSalmon.add(tile);
            continue;
        }
        // ATAS KANAN
        if (currentX + 1 == tile.getX() && currentY + 3 == tile.getY()) {
            nearestSalmon.add(tile);
            continue;
        }
        // BAWAH KIRI
        if (currentX - 1 == tile.getX() && currentY - 3 == tile.getY()) {
            nearestSalmon.add(tile);
            continue;
        }
        // BAWAH KANAN
        if (currentX + 1 == tile.getX() && currentY - 3 == tile.getY()) {
            nearestSalmon.add(tile);
            continue;
        }

    }

    return nearestSalmon;
}
```

```
public List<Tile> findSalmonRun(Tile salmonBefore, Tile salmonCurrent, List<Tile> listSalmon, List<Tile> salmonRun) {
    List<Pair<Integer,Integer>> possibleMove = possibleSalmonMove(salmonBefore, salmonCurrent);
    List<Tile> longestRun = new ArrayList<>();
    for(int i = 0; i < possibleMove.size(); i++) {
        int possibleX = possibleMove.get(i).getX();
        int possibleY = possibleMove.get(i).getY();
        if (!isEmpty(possibleX, possibleY) && getTileFromList(possibleX, possibleY, listSalmon).getAnimalPlaced().equals(anObject:"SALMON")) {
            List<Tile> recurrSalmonRun =
                findSalmonRun(
                    salmonCurrent,
                    getTileFromList(possibleX, possibleY, listSalmon),
                    listSalmon,
                    addSalmonRun(salmonRun, getTileFromList(possibleX, possibleY, listSalmon))
                    );
            if (recurrSalmonRun.size() >= longestRun.size()) {
                longestRun = new ArrayList<>(recurrSalmonRun);
            }
        }
    }
    return longestRun;
}
```

```
public int gradingSalmon() {
    int point = 0;
    int lengthLocal;
    int lengthGlobal = 0;
    List<Tile> listSalmon = new ArrayList<>();
    for (Tile tile : this.listTile) {
        if (tile.getAnimalPlaced().contains(s:"SALMON")) {
            listSalmon.add(new Tile(tile));
        }
    }

    for (Tile salmonBefore : listSalmon) {
        List<Tile> nearestSalmon = findNearestSalmon(salmonBefore, listSalmon); //only one or two
        int len = nearestSalmon.size();

        if (nearestSalmon.size() == 0) {
            point+=2;
        }
        Tile salmonCurrent = nearestSalmon.get(index:0);
        List<Tile> salmonRun = new ArrayList<>();
        salmonRun = findSalmonRun(salmonBefore, salmonCurrent, listSalmon, addSalmonRun(addSalmonRun(salmonRun, salmonBefore),salmonCurrent));

        int anotherSalmonLength = 0;
        if (nearestSalmon.size() == 2) {
            salmonCurrent = nearestSalmon.get(index:1);
            List<Tile> anotherSalmonRun = findSalmonRun(salmonBefore, salmonCurrent, listSalmon, addSalmonRun(addSalmonRun(salmonRun, salmonBefore), salmonCurrent));
            anotherSalmonLength = anotherSalmonRun.size() - 2;
        }

        switch(salmonRun.size() + anotherSalmonLength) {
            case 2:
                point += 5;
                break;
            case 3:
                point += 8;
                break;
            case 4:
                point += 12;
                break;
            case 5:
```

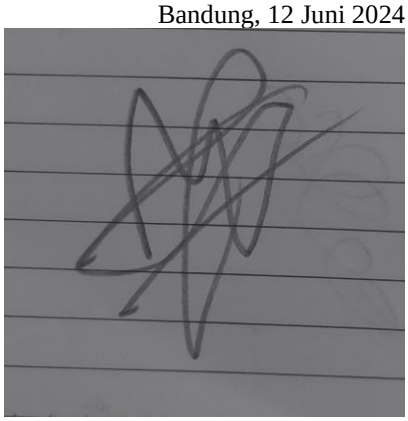
IV. KESIMPULAN

Melalui Algoritma Brute Force. Pemain dapat mencari langkah terbaik yang harus dilakukan pada giliran tersebut untuk mendapatkan hasil maksimal. Meskipun hasil yang diberikan selalu optimal. Algoritma Brute Force tidak selalu memberikan kemenanangan pada permainan *Cascadia* karena pada permainan *Cascadia* faktor kemenangan cukup banyak ditentukan oleh keberuntungan

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, 12 Juni 2024



Matthew Vladimir Hutabarat
13522093