Aplikasi Algoritma *Backtracking* dalam Menentukan Susunan Penempatan Ninja pada Graf Deploy dalam Game Ninja Heroes

Razzan Daksana Yoni – 13521087
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung, Indonesia
E-mail (gmail): 13521087@std.stei.itb.ac.id

***Abstract*—Ninja Heroes New Era adalah salah satu mobile online RPG, permainan ini bergenre adventure, pemain harus mempersiapkan *hero* utama mereka dan hal-hal lain yang membantu dalam memperkuat *hero* mereka seperti *jutsu*, *equipment,* dan *deploy*. *Deploy* adalah graf yang menyediakan tempat untuk menaruh *hero* dan mempunyai keunikan yaitu ketika suatu kombinasi hero terdapat pada graf tersebut akan mendapat tambahan *stat* dan juga ketika ada pipa penghubung berwarna sama saling bertetanggaan juga akan mendapat tambahan *stat*. Untuk itu, pemain dapat mengaplikasikan algoritma *brute-force* yang untuk mendapatkan letak penempatan *hero* yang tepat agar mendapatkan *stat* yang maksimum.**

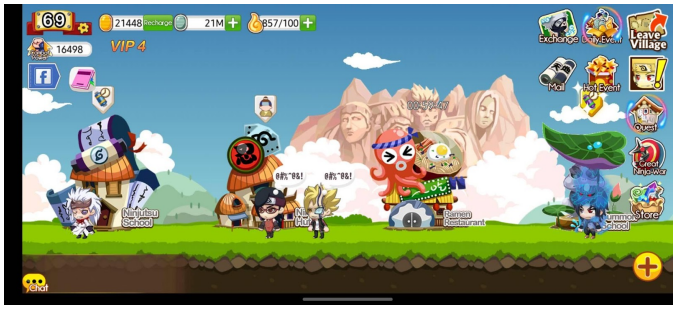
Keywords— Ninja Heroes NewEra, *deploy*, graf, Brute Force, Backtracking

I. Pendahuluan

Ninja Heroes NewEra merupakan permainan *online RPG* yang dikembangkan oleh Kageherostudio dan dirilis pada Agustus 2022. Pemain harus menyusun strategi dan *hero* yang akan digunakan, seperti *chakra,* ninjutsu *skill, deploy*. Pemain harus menentukan kombinasi yang terbaik untuk melawan musuh. Pada makalah ini, hanya difokuskan untuk mencari kombinasi dan penempatan *hero* pada graf *deploy* yang terbaik berdasarkan *hero* yang dimiliki.

Untuk menentukan *deploy* pemain perlu menyiapkan setidaknya 15 *hero* untuk memaksimalkan *stat*. Setiap kombinasi *hero* biasanya punya karakteristik unik yaitu menambah *stat* ataupun menjadi *trigger attack* yang dapat membantu *hero* utama ketika melawan musuh. Semakin tinggi *stat* yang didapat semakin mungkin juga untuk memenangkan pertandingan.[1]

Untuk itu diperlukan suatu algoritma yang dapat membantu menyelesaikan permasalahan tersebut dan penulis menemukan beberapa solusi yaitu *Brute Force* dan *Backtracking*. Namun, pada makalah ini lebih ditekankan pada algoritma *Backtracking* karena merupakan ekstensi dari algoritma *Brute Force*. Penulis juga sudah menentukan rentetan instruksi yang dapat menyelesaikan permasalahan dengan menggunakan algoritma *Backtracking* yaitu dengan mencari solusi persoalan, fungsi pembangkit, dan fungsi pembatas. Dengan itu penulis dapat menuangkan algoritma tersebut pada persoalan dengan menjadikannya ke dalam sebuah makalah.



Gambar 1. Tampilan Laman Utama Ninja Heroes NewEra
(Sumber : Arsip Penulis)

II. Landasan Teori

A. *Brute Force*

Algoritma *brute force* merupakan salah satu metode untuk menyelesaikan suatu persoalan. Jika diartikan secara etimologi, *brute* berarti murni secara fisik dan *force* berarti kekuatan, gaya, memaksa. Oleh karena itu, algoritma *brute force* pada umumnya tidak "cerdas" karena hanya mengandalkan tenaga daripada otak. Dalam ranah penyelesaian suatu masalah, algoritma *brute force* berarti memecahkan masalah menggunakan pendekata yang lempang (*straightforward*). Biasanya algoritma *brute force* didasarkan pada pernyataan pada persoalan dan definisi atau konsep yang dilibatkan. Karena itulah *brute force* mudah digunakan dan hampir seluruh permasalahan dapat diselesaikan dengan algoritma *brute force*.

Meskipun algoritma *brute force* bukanlah algoritma yang cerdas, mangkus, dan sangkil akan tetapi, algoritma *brute force* dapat mudah dicapai penyelesaiannya terkadang algoritma *brute force* bisa menjadi landasan atau pembanding sehingga mendapatkan algoritma mutakhir yang lebih baik seperti mencari elemen terbesar/terkecil, pencarian

berurutun, menghitung nilai dari suatu pangkat bilangan bulat, menghitung faktorial, perkalian matriks dan masih banyak lagi.[2]

Namun, ada beberapa kelemahan algoritma *brute force* yaitu jarang sekali menghasilkan algoritma yang sangkil dan mangkus. Karena solusinya yang sangat lempang dibutuhkan volume komputasi yang besar dan waktu yang lama. Dan juga terkadang algoritma *brute force* disebut sebagai algoritma yang naif karena tidak menggunakan kecerdasan untuk menyelesaikan permasalahannya.

Salah satu contoh penerapan pada algoritma *brute force* yang sering digunakan adalah *exhaustive search* yaitu teknik pencarian solusi untuk persoalan kombinatorik seperti permutasi, kombinasi, atau himpunan bagian dari sebuah himpunan. Langkah – langkagh di dalam *exhaustive search*

1. Enumerasi (*list*) setiap kemungkinan solusi dengan cara yang sistematis
2. Evaluasi setiap kemungkinan solusi satu per satu, simpan solusi terbaik yang ditemukan sampai sejauh ini
3. Bila pencarian berakhir, umumkan solusi terbaik

Adapun beberapa contoh *exhaustive search* yaitu *travelling salesperson problem*, *knapsack problem,* dan enkripsi. Dikarenakan besarnya volume komputasi ada sebuah perbaikan kinerja dalam *exhaustive search* yaitu teknik heuristic yaitu mempercepat dalam pencarian solusi tanpa harus mengeksplorasi kemungkinan solusi secara penuh.

Hal ini yang akan menjadi landasan untuk algoritma yang akan dibahas pada penyelesaian persoalan pada makalah ini yaitu algoritma *backtracking.*[3]

B. Backtracking

Algoritma *backtracking* adalah algoritma yang memperbaiki algoritma *exhaustive search* (salah satu penerapan algoritma *brute* force). Pada *exhaustive search*, semua kemungkinan solusi dieksplorasi dan dievaluasi satu per satu, sedangkan pada algoritma *backtracking*, hanya pilihan yang mengarah ke solusi yang dieksplorasi, pilihan yang tidak mengarah ke solusi tidak dipertimbangkan lagi yaitu dengan memangkas (*pruning*) simpul-simpul yang tidak megnarah ke solusi yaitu dengan cara memberi sebuah fungsi pembatas untuk membunuh *node* yang tidak mengarah ke solusi. Algoritma *backtracking* pertama kali diperkenalkan oleh D. H. Lehmer pada tahun 1950 kemudia R.J Walker, Golomb, dan Baumert menyajikan uraian umum tentang algoritma *backtracking*. [4]

Algoritma *backtracking* dapat dipandang sebagai salah satu dari dua hal berikut yaitu

1. Sebagai sebuah fase di dalam algoritma traversal (*Depth First Search*) DFS yaitu mencari sebuah solusi secara mendalam terlebih dahulu dan menjadi prioritas dalam mencari sebuah penyelesaian
2. Sebagai sebuah metode pemecahan masalah yang mangkus, terstruktur, dan sistematis, baik untuk persoalan optimasi maupun non-optimasi.

Algoritma *backtracking* memiliki properti umum dalam menyelesaikan sebuah permasalahan. Properti umum tersebut yaitu

1. Solusi persoalan

   Solusi dinyatakan sebagai vektor dengan *n-tuple*: X = (x1, x2, …, xn), xi ∈ Si. Umumnya S1 = S2 = … = Sn.

   Contoh: pada persoalan 1/0 *knapsack* Si = {0,1}, xi = 0 atau 1

2. Fungsi pembangkit nilai xk

   Dinyatakan sebagai predikat T()

   T(x[1], x[2], …, x[k-1]) membangkitkan nilai untuk xk, yang merupakan komponen vektor solusi.

3. Fungsi pembatas (*bounding function*)

   Dinyatakan sebagai predikat B(x1, x2, …, xk). B bernilai *true* jika (x1, x2, …, xk) mengarah ke solusi. Mengarah ke solusi artinya tidak melanggar kendala (*constraints*)

Pengorganisasian Solusi

1. Semua kemungkinan solusi dari persoalan disebut ruang solusi (*solution space*).
2. Tinjau *knapsack* 0/1 untuk n=3.
3. Solusi persoalan dinyatakan sebagai X = (x1, x2, x3) dengan xi ∈ {0,1}.
4. Ruang solusinya adalah

   {(0,0,0), (0,1,0), (0,0,1), (1,0,0),

   (1,1,0), (1,0,1), (0,1,1), (1,1,1)}
5. Ruang solusi diorganisasikan ke dalam struktur pohon berakar
6. Tiap simpul pohon menyatakan status (*state*) persoalan, sedangkan sisi (cabang) dilabeli dengan nilai-nilai xi.
7. Lintasan dari akar ke daun menyatakan solusi yang mungkin.
8. Seluruh lintasan dari akar ke daun membentuk ruang solusi.
9. Pengorganisasian pohon ruang solusi diacu sebagai pohon ruang status (*state space tree*)

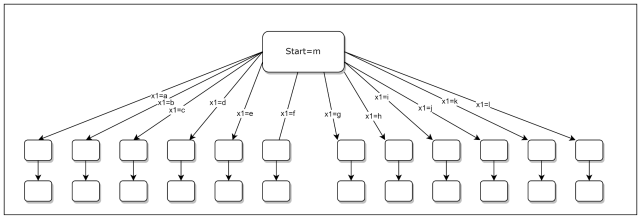
III. Implementasi dan Pembahasan

A. Penyelesaian penempatan hero pada graf deploy Ninja Heroes New Era dengan algoritma Brute Force

Salah satu strategi algoritma yang sangat lempang dan paling mudah diterapkan untuk menyelesaikan permasalahan dalam menempatakan heroes pada graf deploy pada *game Ninja Heroes New Era* adalah *brute force*. Algoritma *brute force* yang menjadi penyelesaian penempatan *heroes* pada graf *deploy* tidak menggunakan heuristik sama sekali dan menggunakan *exhaustive search* dengan mengenumerasikan seluruh kemungkinan sehingga semua kemungkinan tersebut

telah dikomputasikan dan akan didapatkan hasil yang optimal yang diinginkan. Tahapan algoritma *brute force* dalam menyelesaikan permasalahan dalam makalah ini sebagai berikut

1. Masukkan ninja yang akan menjadi ninja utama yang akan digunakan dalam pertandingan
2. Masukkan ninja yang akan menjadi ninja pembantu yang akan dihitung juga dalam penentuan penempatan ninja pada graf *deploy*
3. Enumerasikan semua kemungkinan yang mungkin terjadi
4. Hitung secara traversal total pipa yang tersambung pada graf
5. Bandingkan dengan solusi sebelumnnya apakah sudah maksimal atau tidak
6. Jika sudah tidak ada lagi berhenti, solusi didapatkan
7. Jika tidak lanjut ke kemungkinan selanjutnya
8. Lanjut ke (4)



Gambar 2. Ilustrasi pohon pada implementasi algoritma *brute force*
(Sumber : Arsip Penulis)

Dapat dilihat bahwa semua kemungkinan yang terjadi selalu diperhitungkan dan ini akan memerlukan biaya yang sangat besar. Ada penyelesaian naif dengan menggunakan algoritma *brute force* yaitu dengan menggunakan konsep *multithreading* akan tetapi ini hanya mempercepat secara waktu tapi biaya akan tetap sama karena total biaya untuk komputasi yang disebar ke setiap *state*-nya akan sama. Oleh karena itu penulis memikirkan ide lain yaitu dengan menggunakan algoritma *Backtracking* yaitu dengan mematikan kemungkinan yang tidak mencapai solusi yang tepat.

B. Penyelesaian penempatan hero pada graf deploy Ninja Heroes New Era dengan algoritma Backtracking

Salah satu algoritma yang menjadi pendekatan dalam menyelesaikan permasalahan ini adalah menggunakan algoritma yang disebut *Backtracking*. Algoritma ini merupakan algoritma perbaikan dari algoritma *Brute Force* yang telah diberikan pada bagian sebelumnnya. Pemetaan persoalan ke dalam algoritma *backtracking* adalah sebagai berikut

1. Solusi dari persoalan penempatan hero dalam graf *deploy* dinyatakan dalam *list of Hero*
2. Ruang solusi dari permasalahan adalah didapatkan hasil total penyambungan pipa maksimum]
3. Simpul dari pohon ruang status merupakan pilihan hero pada simpul tertentu
4. Sisi dari pohon ruang status adalah bagian kosong yang akan diisi dengan salah satu hero yang belum diaokasikan
5. Fungsi pembangkit dari permasalahan adalah fungsi yang mengisi graf dengan salah satu hero yang belum dialokasikan
6. Fungsi pembatas atau *bounding function* pada permasalahan ini adalah fungsi yang mengecek apakah node pada level ke-n total pipa tersambungnya lebih atau sama dengan total pipa tersambung minimum

Berdasarkan hasil pemetaan persoalan, didapatkan salah satu pemecahan masalah dengan metode algoritma *backtracking* dapat diselesaikan dalam beberapa proses yakni sebagai berikut

1. Masukkan ninja yang akan menjadi ninja utama yang akan digunakan dalam pertandingan
2. Masukkan ninja yang akan menjadi ninja pembantu yang akan dihitung juga dalam penentuan penempatan ninja pada graf *deploy*
3. Cari nilai minimum yang mungkin untuk menghubungkan pipa pada graf *deploy* yang akan menjadi fungsi pembatas atau *bounding function* yaitu dengan menghitung banyak warna yang sama dengan posisi yang bersebrangan contoh dilakukan perhitungan nilai minimum pada bagian samping dengan warna merah maka akan dicari nilai pipa tersambung minimum dengan menghitung semua pipa yang berwarna merah di bagian kiri dan juga semua pipa berwarna merah di bagian kanan. Lalu jika ada satu hero yang sama memiliki pipa bagian kiri berwarna merah dan bagian kanan juga berwarna merah beri sebuah *flag boolean* untuk menandakan ada hero yang sama agar komputasi lebih baik. Jika sudah dihitung total pipa yang berwarna merah pada bagian samping bagi dua dengan *floor function*
4. Tentukan apakah total minimum yang didapat merupakan bagian samping atau atas bawah
5. Enumerasikan semua kemungkinan yang mungkin terjadi
6. Bagi graf secara kolom maupun secara baris
7. Jika pada pencarian nilai minimum didapatkan adalah bagian samping maka komputasi graf secara baris saja, juga sebaliknya jika didapatkan bagian atas bawah maka komputasi graf secara kolom saja
8. Jika total pipa yang tersambung kurang dari n pipa tersambung minimum maka bunuh node ini akan tetapi, jika total pipa yang tersambung lebih atau sama dengan n pipa tersambung minimum maka lanjutkan komputasi berikutnya
9. Komputasikan secara baris dan kolom pada kasus yang sedang didapatkan

10. Jika didapatkan total pipa tersambung lebih dari total pipa minimum jadikan sebagai solusi terbaik saat ini
11. Lanjutkan ke (6) sehingga tidak ada lagi kasus yang akan dicari.

Penulis telah mengimplementasikan penyelesaian dengan algoritma *backtracking* tersebut dengan bahasa python yaitu sebagai berikut

Pipa minimum yang harus didapatkan

```
def get_minima(main_heroes: list, deploy_heroes: list):
    sum_minima = 0
    top_down = False

    for i in Color:
        for j in range(2):
            sum = 0
            same_color = False
            for k in range(len(deploy_heroes)):
                if j == 0:
                    sum += 1 if deploy_heroes[k].left == i else 0
                    sum += 1 if deploy_heroes[k].right == i else 0
                    if deploy_heroes[k].left == deploy_heroes[k].right:
                        same_color = True
                else:
                    sum += 1 if deploy_heroes[k].top == i else 0
                    sum += 1 if deploy_heroes[k].bottom == i else 0
                    if deploy_heroes[k].top == deploy_heroes[k].bottom:
                        same_color = True

            if sum_minima == 0:
                sum_minima = sum // 2
            elif sum // 2 > sum_minima:
                sum = sum // 2
                sum -= 1 if same_color else 0
                sum_minima = sum
                top_down = True if j == 1 else False

    sum_main_hero = 0
    for i in range(2):
        if main_heroes[i].right == main_heroes[i + 1].left:
            sum_main_hero += 1
    print("minimum cost was found", sum_minima, top_down)
return sum_minima, sum_main_hero, top_down
```

Algoritma *backtracking* dalam penentuan graf deploy

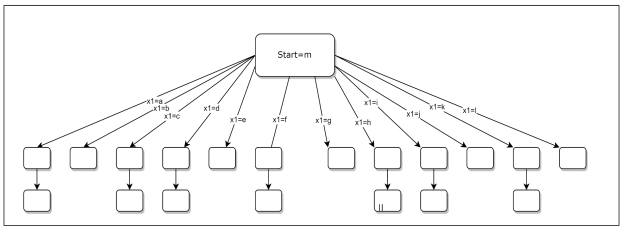
```
def get_deploy_backtrack(
        main_heroes: list, deploy_heroes: list, sum_minima: int, sum_main_hero, top_down: bool
):
    result = []
    sum_result = 0
    perm = permutations(deploy_heroes, 12)
    sum_continue = 0
    for combination in perm:
        if sum_continue > 0:
            # print(f"skipped {sum_continue} combinations")
            sum_continue -= 1
            continue
        row_1 = combination[0:5]
                row_2 = [combination[5]] + main_heroes + [combination[6]]
        row_3 = combination[7:12]
        rows = [row_1, row_2, row_3]

        col_1 = [row_1[0], row_2[0], row_3[0]]
        col_2 = [row_1[1], row_2[1], row_3[1]]
        col_3 = [row_1[2], row_2[2], row_3[2]]
        col_4 = [row_1[3], row_2[3], row_3[3]]
        col_5 = [row_1[4], row_2[4], row_3[4]]
        cols = [col_1, col_2, col_3, col_4, col_5]

        sum = 0
        if top_down:
            for col in cols:
                sum += 1 if col[1].top == col[0].bottom else 0
                sum += 1 if col[1].bottom == col[2].top else 0
        else:
            for row in range(len(rows)):
                for i in range(len(rows[row])):
                    if i < len(rows[row]) - 1:
                            sum += 1 if rows[row][i].right == rows[row][i + 1].left else 0

        if sum >= (sum_minima + sum_main_hero if not(top_down) else 0):
            # compute
            if top_down:
                for row in range(len(rows)):
                    for i in range(len(rows[row])-1):
                            sum += 1 if rows[row][i].right == rows[row][i + 1].left else 0
            else:
                for col in cols:
                    sum += 1 if col[1].top == col[0].bottom else 0
                    sum += 1 if col[1].bottom == col[2].top else 0

            if sum > sum_result:
                sum_result = sum
                result = rows[:]
            else:
                        sum_continue = math.factorial(sum_minima + sum_main_hero if not(top_down) else 0 - sum - 1) - 1
            if sum_continue < 0:
                sum_continue = 0
            continue
    return sum_result, result
```



Gambar 3. Ilustrasi pohon pada implementasi algoritma *backtracking*
(Sumber : Arsip Penulis)

C. *Hasil Pengujian*

Algoritma *backtracking* dan *brute force* diuji untuk menyelesaikan permasalahan pemetaan hero pada graf *deploy Ninja Heroes New Era* pada beberapa kasus acak. Lengkap dari program percobaan akan diberikan pada referensi.

```
Hero,Kiri,Kanan,Atas,Bawah
Rinegan Sasuke,4,1,4,3
Six Path Madara,1,1,1,1
Boruto Karma,1,2,3,2
Kabuto Yakushi,1,1,3,3
Kakashi,1,2,4,1
Shikamaru,2,3,1,4
Tsunade,3,3,3,3
Naruto,1,3,4,3
Iruka,2,4,3,1
Sakura,4,2,2,1
Orochimaru,2,4,1,1
Boruto,1,2,3,4
Karin,2,3,1,3
Jiraiya,3,4,2,1
Kabuto,1,2,3,2
```

Setelah dikomputasi menggunakan algoritma *backtracking* didapatkan hasil

```
Total pipa tersambung : 19

Penempatan Hero pada graf
Boruto, Shikamaru, Kakashi, Tsunade,
Jiraiya,
Naruto, Rinegan Sasuke, Six Path Madara,
Boruto Karma, Karin,
Kabuto Yakushi, Kabuto, Orochimaru,
Sakura, Iruka,
Waktu eksekusi: 1572.1926774978638 detik
```

Setelah dikomputasi menggunakan algoritma *backtracking* didapatkan hasil

```
Total pipa tersambung : 19

Penempatan Hero pada graf
Boruto, Shikamaru, Kakashi, Tsunade,
Jiraiya,
Naruto, Rinegan Sasuke, Six Path Madara,
Boruto Karma, Karin,
Kabuto Yakushi, Kabuto, Orochimaru,
Sakura, Iruka,
Waktu eksekusi: 2014.1645687397543 detik
```

Gambar 4. Hasil pada graf *deploy Ninja Heroes New Era*
(Sumber : Arsip Penulis)

Dapat dilihat bahwa hasil yang didapatkan sama akan tetapi, perbandingan waktu pada kedua algoritma lumayan dekat sekitar 786:1007 dengan selisih waktu sekitar tujuh menit hal ini dikarenakan pasangan pipa memiliki total yang sedikit oleh karena itu tingginya *rate* untuk tidak terjadi pruning. Namun ini akan sangat membantu ketika total pasangan pipa minimum yang semakin banyak.

D. *Analisis Kompleksitas Waktu dan Ruang*

Setelah didapatkan solusi perlu juga untuk menganalisis kompleksitas waktu dan ruang untuk menentukan apakah solusi tersebut akan lebih baik daripada brute force atau tidak. Untuk itu dalam menentukan kompleksitas waktu yang perlu dibutuhkan dengan menentukan banyak operasi yang dilakukan.

Pada algoritma *brute force* di-enumerasikan setiap kemungkinan dan didapatkan sebanyak n!. Pada makalah ini n bersifat tetap yaitu sebesar 12! akan tetapi penulis mempertimbangkan agar tetap menjadi sebagai nilai n agar merepresentasikan kemungkinan untuk segala kasus. Oleh karena itu, pada algoritma *brute force* didapatkan bahwa kompleksitas waktu nya dalam notasi Big-O adalah sebesar

$$O(n!)$$

Pada algoritma *backtracking* di-enumerasikan setiap kemungkinan dan didapatkan sebanyak n!. Pada makalah ini n bersifat tetap yaitu sebesar 12! akan tetapi penulis mempertimbangkan agar tetap menjadi sebagai nilai n agar merepresentasikan kemungkinan untuk segala kasus. Namun, karena *backtracking* adalah sifat heuristik daripada *exhaustive search* adanya fungsi pembatas atau *bounding function* untuk mempercepat dalam pencapaian solusi setelah dilakukan kalkulasi fungsi pembatas dengan kompleksitas waktu sebesar

$$O((n-(m*2)-1)!)$$

dengan m adalah nilai minimum pipa yang perlu dicapai. Oleh karena itu, pada algoritma *backtracking* didapatkan bahwa kompleksitas waktu nya dalam notasi Big-O adalah sebesar

$$O(n! - (n-(m*2))!)$$

Setelah dilihat, dapat disimpulkan bahwa kompleksitas waktu algoritma *backtracking* lebih baik daripada algoritma *brute force*. Meskipun pada kasus m yang kecil tidak akan terlalu berpengaruh akan tetapi untuk m yang semakin besar perbedaan yang dirasakan akan semakin terasa.

Untuk menentukan kompleksitas ruang pada pada algoritma *brute force* dan algoritma *backtracking* dapat ditentukan dengan melihat banyaknya memori ketika melakukan komputasi algoritma tersebut. Dapat dilihat bahwa hanya diperlukan satu buat array untuk melakukan komputasi. Oleh karena itu didapatkan bahwa keduanya memiliki kompleksitas waktu yang sama yaitu sebesar.

$$O(n)$$

IV. Kesimpulan

Penentuan penempatan hero pada graf deploy pada *game* Ninja Heroes New Era dapat dicapai dengan beberapa algoritma alternatif yaitu dengan *brute force* dan *backtracking*. Diperlukan pengkalkulasian matematika dan konsep pembelajar algoritma yang baik untuk mendapatkan alternatif ini ataupun alternatif lain. Algoritma *brute force* dapat dilihat bahwa dapat mencapai hasil yang sudah pasti tepat meskipun dengan kompleksitas waktu yang sangat tinggi dan algoritma *backtracking* juga dapat mencapai hasil yang tepat sama seperti algoritma *brute force* dengan sedikit pengurangan kompleksitas waktu untuk melakukan komputasi yang tidak diperlukan.

Video Link at Youtube dan Github

Youtube: https://youtu.be/jf1zVhvX8gU

Github: https://github.com/razzanYoni/DeployNHNE.git

Ucapan Terima Kasih

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas berkat dan karunia-Nya, penulis dapat menyelesaikan makalah yang berjudul "Aplikasi Algoritma *Backtracking* dalam Menentukan Susunan Penempatan Ninja pada Graf Deploy dalam Game Ninja Heroes" dengan lancar. Penulis mengucapkan terimakasih sebesar-besarnya kepada bapak Dr. Ir. Rinaldi Munir, M.T., ibu Dr. Nur Ulfa Maulidevi ST,M.Sc, dan bapak Ir. Rila Mandala, M.Eng., Ph.D., sebagai pengampu dalam menjalani kuliah IF2211 Strategi Algoritma 2022/2023, atas arahan, bimbingan dan ilmu yang telah diajarkan kepada penulis dan teman-teman. Penulis juga mengucapkan terimakasih kepada orangtua, saudara, dan teman-teman yang selalu memberi dukungan sehingga penulis dapat menyelesaikan makalah ini dengan lancar. Penulis juga meminta maaf jika ada kekeliruan atau kesalahan yang terdapat pada makalah ini, baik sengaja maupun tidak disengaja. Penulis sangat terbuka terhadap masukan, kritikan, dan saran terhadap makalah ini. Akhir kata, penulis berharap semoga makalah ini dapat bermanfaat.

References

[1] https://gamefinity.id/game/review/ninja-heroes-new-era-turn-based-yang-bangkit-dari-kubur/

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf

[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag2.pdf

[4] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.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, 22 Mei 2023

Razzan Daksana Yoni 13521087