Optimasi Prosedural Generasi Peta pada Game Rogue-like Menggunakan Teori Graf dan Algoritma *Kruskal* untuk Menjamin Rute yang Adil dan Bervariasi

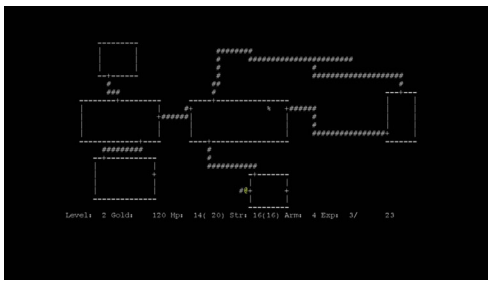
Edward David Rumahorbo - 13524036
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: edwardrumahorbo1@gmail.com , 13524036@std.stei.itb.ac.id

***Abstract*—Fondasi dari sebuah gim *roguelike* adalah *map generation*, yang mana akan terus terjadi ketika seorang pemain gagal dalam nyelesaikan suatu level. Akan tetapi, *map generation* terkadang tidak memberikan pemain sebuah keadilan pada setiap generasinya. Penelitian ini bertujuan untuk mengoptimasi *map generation* pada sebuah gim bertema *roguelike*.**

Keywords—Roguelike; Graf; Algoritma Kruskal; Keadilan; Variasi

I. PENDAHULUAN

Roguelike adalah sebuah tema gim yang memberikan pengalaman terhadap mekanisme *permadeath* dan generasi *map* secara prosedural. Cikal bakal perkembangan gim *roguelike* berasal dari sebuah gim bernama *Rogue* pada tahun 1980. Proses generasi *map* dilakukan menggunakan suatu algoritma yang disebut *Procedural Content Generation* (PCG). PCG sudah menjadi elemen desain inti perkembangan dunia gim, khususnya untuk gim *roguelike*. PCG memungkinkan sebuah gim *roguelike* menghasilkan generasi *map*, *level*, dan *item* yang beragam pada setiap sesi permainannya. Poin inilah yang mendukung tingkat *replayability*—kemauan pemain untuk kembali memainkan gim tersebut—pada sebuah gim *roguelike*. Akan tetapi, algoritma ini juga memungkinkan terjadinya generasi yang tidak seimbang dan adil apa setiap iterasi permainannya. Tingkat variasi dan keadilan rute permainan menjadi dilema tersendiri untuk gim bertema *roguelike*.

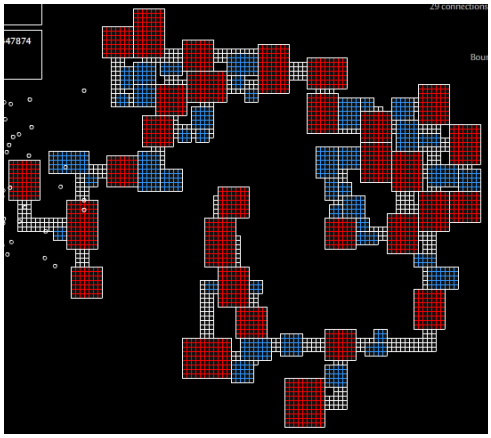


Gambar 1. Gim Rogue (1980)

Sumber : https://www.youtube.com/watch?v=vxF1osPkplA

Struktur *map* yang terlalu teratur memberikan pengalaman yang membosankan untuk para pemain, karena langkah penyelesaian permainan akan menjadi terlalu mudah ditebak dan menurunkan tingkat replayability untuk gim tersebut. Sebaliknya, struktur *map* yang terlalu acak menciptakan suatu pengalaman permainan yang terasa tidak ada untuk setiap generasi *map*-nya. Pemain mungkin dihadapkan pada suatu kondisi di mana akan sangat sulit untuk mencapai suatu area karena jalur yang panjang dan berbelit, tetapi daerah lain bisa dicapai dengan sangat mudah tanpa memerlukan upaya lebih.

Kedua masalah tersebut bisa dipecahkan menggunakan dua konsep, yaitu keadilan rute (*route fairness*) dan variasi rute (*route variation*). Keadilan yang dimaksud adalah tidak adanya perbedaan kesulitan yang ekstrem pada generasi *map* yang berbeda. Sedangkan rute yang variatif dimaksudkan agar para pemain bisa berseksplorasi dan beradaptasi pada perubahan rute jalur pada setiap generasi *map*-nya. Konsep ini tidak hanya meningkatkan pengalaman bermain yang adil, tetapi juga meningkatkan *replayability* pada gim tersebut.



Gambar 2. Pengecekan Konektivitas pada *Map*

Sumber : https://www.gridsagegames.com/blog/gsg-content/uploads/2014/06/mapgen_complex.png

II. DASAR TEORI

A. *Graf*

1) Definisi

Graf G adalah pasangan $G = (V, E)$, yang dalam hal ini V adalah kumpulan tidak kosong dari simpul-simpul (vertices) dan E adalah himpunan sisi (edges) yang menghubungkan sepasang simpul, yang dapat ditulis dalam notasi berikut :

$$V = \{v_1, v_2, v_3, \dots, v_n\}$$

$$E = \{e_1, e_2, e_3, \dots, e_n\}$$

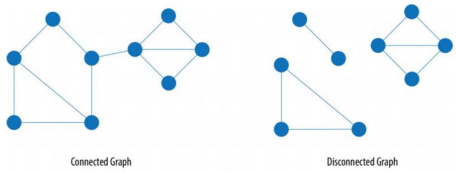
Jumlah sisi yang terhubung dengan sebuah simpul disebut sebagai derajat ($d(v)$), sedangkan kedua simpul yang terhubung langsung melalui sebuah sisi disebut sebagai simpul yang bertetangga.

2) Lintasan (Path) dan Sirkuit (Circuit)

Lintasan dengan panjang n dengan simpul awal v_0 dan simpul akhir v_n di dalam graf G merupakan barisan selang-seling simpul-simpul dan sisi-sisi yang berbentuk $v_0, e_1, v_1, e_2, v_2, \dots, v_{n-1}, e_n, v_n$ sedemikian sehingga e_i adalah sisi-sisi dari graf G. Sedangkan sirkuit (*circuit*) merupakan sebuah lintasan yang berawal dari sebuah simpul v_0 dan kembali ke simpul v_0 di dalam graf G yang melewati barisan selang-seling simpul-simpul dan sisi-sisi.

3) Graf Terhubung (Connected Graph) dan Graf Lengkap (Complete Graph)

G disebut sebagai sebuah graf terhubung (*connected graph*) jika untuk setiap pasang *simpul* v_i dan v_j dalam himpunan V terdapat lintasan dari v_i ke v_j. Sedangkan, graf lengkap adalah graf yang setiap simpulnya bertetangga dengan semua simpul lain yang ada pada graf.

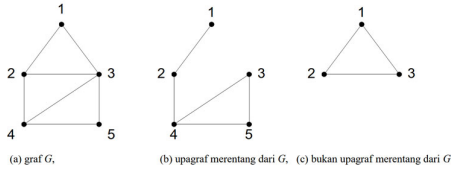


Gambar 3. Graf Terhubung dan Tidak Terhubung

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

4) Upagraf (Subgraph) dan Upagraf Merentang (Spanning Subgraph)

Graf $G_1 = (V_1, E_1)$ dikatakan sebagai upagraf (*subgraph*) dari sebuah graf G jika $V_1 \subseteq V$ dan $E_1 \subseteq E$. Upagraf merentang (*spanning subgraph*) G_1 adalah upagraf yang mengandung semua simpul dari G.

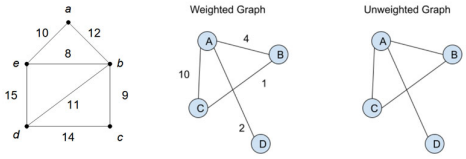


Gambar 4. Upagraf

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

5) Graf Berbobot (Weighted Graph)

Graf berbobot (*weighted graph*) adalah graf yang setiap sisinya memiliki sebuah harga (bobot).



Gambar 5. Graf Berbobot dan Tidak Berbobot

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

B. *Pohon*

1) Definisi

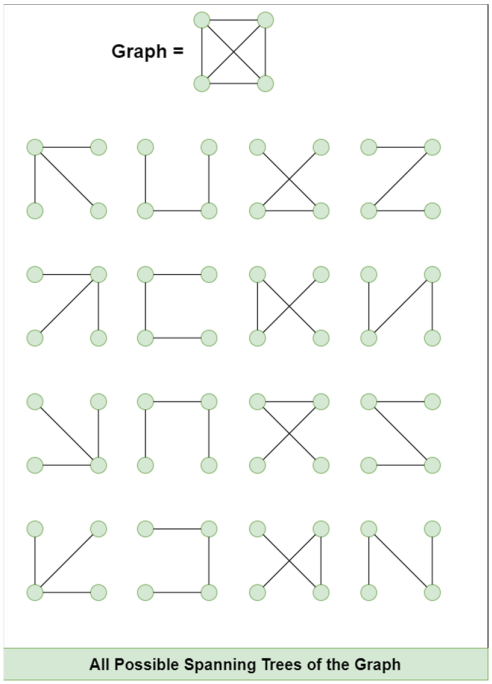
Pohon adalah sebuah graf tak-berarah dan terhubung yang tidak memiliki sirkuit. Misalkan sebuah graf $G = (V, E)$

merupakan graf tak-berarah sederhana dengan jumlah simpul sebanyak n, maka dapat dikatakan bahwa

- *G* adalah sebuah pohon
- Setiap pasang simpul di G terhubung dengan dengan lintasan tunggal
- *G* terhubung dan memiliki $m = n - 1$ buah sisi
- *G* tidak mengandung sirkuit
- Penambahan satu sisi pada graf *G* akan membuat hanya satu sirkuit
- *G* terhubung dan semua sisinya adalah jembatan.

2) Pohon Merentang (Spanning Tree) dan Minimum Spanning Tree (MST)

Pohon merentang dari sebuah graf terhubung adalah upagraf merentang yang berupa pohon. Pohon merentang dari sebuah graf *G* didapat dengan memutus sirkuit yang ada pada graf tersebut. Suatu graf terhubung-berbobot mungkin saja memiliki lebih dari satu *spanning tree*. *Spanning tree* yang memiliki bobot minimum dinamakan pohon merentang minimum (*minimum spanning tree*) atau dapat disingkat menjadi MST. MST merepresentasikan cara paling efektif untuk menghubungkan semua simpul yang ada pada suatu graf.

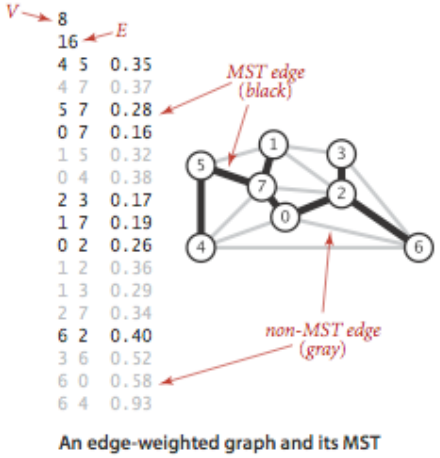


Gambar 6. Pohon Merentang

Sumber : https://translate.google.com/translate?u=https://www.geeksforgeeks.org/dsa/spanning-tree/&hl=id&sl=en&tl=id&client=imgs

3) Algoritma Kruskal

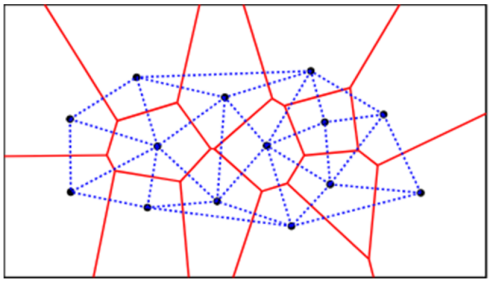
Algoritma Kruskal merupakan salah satu algoritma yang dapat digunakan untuk mencari MST dari sebuah graf. Untuk menemukan MST pada sebuah graf, pertama-tama, urutkan semua sisi pada graf tersebut berdasarkan bobotnya dari yang terkecil ke yang terbesar. Awalnya pohon *T* kosong. Kemudian, pilih sisi dengan bobot minimum yang tidak membentuk sirkuit di T dan tambahkan sisi tersebut ke dalam *T*. ulangilah langkah tersebut sebanyak $n - 1$ kali.



Gambar 7. MST pada Sebuah Graf

C. Delaunay Triangulation

Bayangkan, anda berada pada suatu ruangan gelap bersama teman-teman anda dan masing-masing orang membawa sebuah senter. Semua orang yang ada di ruangan itu kemudian menyalakan senternya masing-masing. *Delaunay triangulation* adalah cara untuk menghubungkan orang-orang yang ada di ruangan tersebut menjadi jaringan segitiga sedemikian sehingga hubungannya terasa paling "alami" dan lokal. *Delaunay triangulation* dari sekumpulan simpul pada suatu graf tanpa sisi adalah cara untuk membagi area menjadi jaring-jaring segitiga dengan setiap simpulnya menjadi sudut dari segitiga-segitiga tersebut.



Gambar 8. *Delaunay Triangulation*

Sumber :
https://www.cs.umd.edu/class/spring2020/cmsc754/Lects/cmsc754-spring2020-lects.pdf

D. Perhitungan Keadilan dan Variasi

1) Keadilan

Perhitungan keadilan (*fairness*) pada graf akan dilakukan menggunakan standar deviasi dari panjang jalur (σPL). Untuk melakukannya, pertama-tama, akan dicari sisi terpendek dari simpul awal ke setiap simpul lain. Kumpulan panjang jalur tersebut kemudian akan dianalisis untuk dihitung σPL-nya. Nilai σPL yang rendah menandakan panjang jalur yang relatif seragam. Keseragaman ini mengindikasikan *fairness* pada graf, karena tidak ada simpul yang jaraknya terlalu jauh dari simpul lainnya.

2) Variasi

Perhitungan variasi (*variety*) akan dilakukan dengan menggunakan rata-rata jarak Jaccard (*Jaccard distance*). Untuk dua graf G_1 dan G_2 dengan himpunan sisi dari masing-masing graf adalah E_1 dan E_2, jarak Jaccard dapat dihitung dengan menggunakan persamaan :

$$J(E1, E2) = 1 - \frac{|E1 \cap E2|}{|E1 \cup E2|}$$

Jarak Jaccard digunakan untuk menilai tingkat perbedaan antara 2 buah *map*. Nilai *J* berkisar dari 0 hingga 1. Semakin nilai *J* mendekati 1, maka tingkat perbedaan struktur antara kedua graf semakin tinggi.

III. Implementasi Teori Graf dan Algoritma Kruskal

A. Representasi Data dan Inisiasi

Keterhubungan antar ruangan pada *map* akan direpresentasikan dalam bentuk graf dengan menggunakan *library* networkx pada Python. Simpul pada graf merepresentasikan satu ruangan yang masing-masing memiliki koordinat (*x*,*y*) yang berbeda. Proses inisiasi dimulai dengan membuat variabel-variabel global sebagai berikut :

- JUMLAH_PETA_GENERASI (*n*) : banyaknya peta yang akan digenerasi pada program
- JUMLAH_RUANGAN (*nv*): banyaknya simpul pada graf yang merepresentasikan jumlah ruangan
- JARAK_MINIMUM_ANTAR_RUANGAN (*min_dis)* : jarak minimum antar ruangan, karena setiap ruangan memiliki dimensi luas.
- PERSENTASE_SISI_EKSTRA (*k*) : jumlah persentase sisi non-MST yang ditambahkan untuk membentuk sirkuit
- UKURAN_GRID_X (*x*) : lebar area penempatan ruangan
- UKURAN_GRID_Y (*y*) : tinggi area penempatan ruangan

Setelah inisiasi variabel global dilakukan, algoritma akan mulai melakukan iterasi sebanyak JUMLAH_RUANGAN kali untuk menentukan posisi acak (*x*,*y*) dari ruangan-ruangan yang ada pada map serta menghubungkannya dengan sisi-sisi sesuai dengan MST dan PERSENTASE_SISI_EKSTRA. Langkah ini diulang hingga JUMLAH_PETA_GENERASI kali untuk menguji tingkat keadilan dan variasi dari banyak generasi peta.

B. Algoritma dalam Generasi Peta

Algoritma ini dilakukan dalam 3 tahapan utama untuk memastikan generasi peta yang adil dan variatif, sehingga meningkatkan *replayability* dari gim tersebut :

1) Membuat Graf

Awalnya, suatu graf *G* akan dibuat dengan penambahkan simpul-simpul untuk setiap ruangan yang ada. Kemudian, simpul $u,v \in V$ akan dihubungkan satu dengan yang lainnya menggunakan *Delaunay Triangulation*. Setelah terbentuk suatu graf, sisi-sisi pada *G* kemudian diurutkan berdasarkan bobotnya.

2) Menentukan MST dengan menggunakan Algoritma Kruskal

Penentuan MST digunakan untuk mencari sisi-sisi dengan bobot paling rendah dalam graf lengkap tersebut. Ini bertujuan untuk menemukan koneksi paling efektif untuk menghubungkan setiap ruangan yang ada. Awalnya, semua sisi pada graf akan didata dan diurutkan berdasarkan bobotnya. Kemudian, dengan menggunakan Algoritma Kruskal, dibentuk suatu MST yang sifatnya masih kaku karena kurangnya variasi.

3) Penambahan Sisi non-MST

Setelah terbentuk sebuah MST dari graf lengkap-berbobot tersebut, akan ditambahkan beberapa sisi non-MST untuk meningkatkan variasi dari *map* yang ada. Sisi-sisi dengan bobot terkecil yang tidak termasuk ke dalam MST akan ditambahkan ke dalam *map* untuk menambah rute-rute alternatif menuju suatu ruangan. Pemilihan sisi-sisi termurah bertujuan untuk mengurangi dampak negatif yang selanjutnya akan berpengaruh pada tingkat keadilan pada generasi peta.

C. Skenario Pengujian

Pengujian akan dilakukan untuk mengukur tingkat keadilan dan variasi dari generasi peta sesuai dengan algoritma yang sebelumnya sudah dipaparkan dengan melihat bagaimana perubahan nilai PERSENTASE_SISI_EKSTRA mempengaruhi perubahan nilai σPL, rata-rata panjang jalur utama, rata-rata titik keputusan di jalur, dan jarak Jaccard antar generasi peta.

Pada pengujian ini, PERSENTASE_SISI_EKSTRA akan menjadi variabel bebas, sedangkan JUMLAH_PETA_GENERASI, JUMLAH_RUANGAN, JARAK_MINIMUM_ANTAR_RUANGAN, UKURAN_GRID_X, dan UKURAN_GRID_Y akan menjadi variabel kontrol. Pengujian ini akan dilakukan menggunakan bahasa Python. *Library* yang akan digunakan adalah numpy, scipy.spatial, networkx, dan matplotlib.pyplot.

```python
import random
import time
import itertools
from collections import Counter
import math

import numpy as np
from scipy.spatial import Delaunay
import networkx as nx
import matplotlib.pyplot as plt

# --- KONFIGURASI EKSPERIMEN ---
CONFIG = {
    "JUMLAH_PETA_GENERASI": 500,
    "JUMLAH_RUANGAN": 15,
    "JARAK_MINIMUM_ANTAR_RUANGAN": 80,
    "PERSENTASE_SISI_EKSTRA": 0.05,
    "UKURAN_GRID_X": 800,
    "UKURAN_GRID_Y": 600,
    "SIMPAN_GAMBAR_PETA": True,
}
```

Gambar 9. *Library* dan Konfigurasi Program

Sumber : Dokumentasi Pribadi

Pengujian akan dimulai dengan membuat sebuah graf yang mengandung *nv* buah simpul.

```python
def bangun_graf_awal(points):
    """Membangun graf awal dari titik-titik ruangan menggunakan Delaunay Triangulation."""
    if points.shape[0] < 4:
        print("PERINGATAN: Terlalu sedikit titik untuk Delaunay Triangulation. Membuat graf manual.")
        graf = nx.Graph()
        for i in range(len(points)):
            graf.add_node(i, pos=points[i])
        for i in range(len(points)):
            for j in range(i + 1, len(points)):
                graf.add_edge(i, j, weight=random.uniform(0.1, 1.0))
        return graf

    tri = Delaunay(points)
    graf = nx.Graph()
    for i in range(len(points)):
        graf.add_node(i, pos=points[i])
    for simplex in tri.simplices:
        for i, j in itertools.combinations(simplex, 2):
            graf.add_edge(i, j, weight=random.uniform(0.1, 1.0))
    return graf
```

Gambar 10. Pembentukan Graf Awal

Sumber : Dokumentasi Pribadi

Akan tetapi, pembentukan simpul memiliki ketentuan. Satu simpul dengan simpul lainnya harus memiliki jarak minimal sebesar *min_dis* satuan.

```python
def tempatkan_ruangan_dengan_jarak(jumlah: int, grid_x: int, grid_y: int, jarak_minimum: float):
    """
    Menghasilkan koordinat acak untuk ruangan dengan memastikan jarak minimum.
    Menggunakan metode 'rejection sampling' untuk menempatkan simpul satu per satu.
    """
    points = []
    max_attempts_per_point = 100

    for i in range(jumlah):
        is_point_placed = False
        for _ in range(max_attempts_per_point):
            candidate_x = random.uniform(0, grid_x)
            candidate_y = random.uniform(0, grid_y)

            is_valid = True
            for p in points:
                dist = math.hypot(candidate_x - p[0], candidate_y - p[1])
                if dist < jarak_minimum:
                    is_valid = False
                    break

            if is_valid:
                points.append((candidate_x, candidate_y))
                is_point_placed = True
                break

        if not is_point_placed:
            print(f"PERINGATAN: Gagal menempatkan simpul ke-{i+1} setelah {max_attempts_per_point} percobaan. "
                  "Area mungkin terlalu padat. Coba kurangi JARAK_MINIMUM atau JUMLAH_RUANGAN.")

    return np.array(points)
```

Gambar 11. Pengecekan Jarak Minimum antar Simpul

Sumber : Dokumentasi Pribadi

Setelah graf tersebut terbentuk, Program akan mulai menentukan MST dari graf tersebut dan menambahkan beberapa sisi sesuai dengan nilai *k*.

```python
def jalankan_kruskal(graf):
    """Menjalankan algoritma Kruskal untuk menemukan Minimum Spanning Tree (MST)."""
    mst = nx.minimum_spanning_tree(graf, algorithm='kruskal')
    for node, data in graf.nodes(data=True):
        mst.nodes[node]['pos'] = data['pos']
    return mst

def tambah_sisi_ekstra(graf_mst, graf_awal, persentase):
    """Menambahkan beberapa sisi kembali ke MST untuk menciptakan siklus/rute alternatif."""
    sisi_non_mst = list(set(graf_awal.edges()) - set(graf_mst.edges()))
    sisi_non_mst.sort(key=lambda e: graf_awal.edges[e]['weight'])
    jumlah_sisi_untuk_ditambah = int(len(sisi_non_mst) * persentase)
    for i in range(min(jumlah_sisi_untuk_ditambah, len(sisi_non_mst))):
        u, v = sisi_non_mst[i]
        graf_mst.add_edge(u, v)
    return graf_mst
```

Gambar 12. Penentuan MST dan Penambahan Sisi

Sumber : Dokumentasi Pribadi

Telah terbentuk sebuah graf yang mengandung MST dan beberapa tambahan sisi untuk menciptakan variasi pada peta. Selanjutnya, program akan menghitung metrik keadilan dan variasi, yaitu σPL, rata-rata panjang jalur utama, rata-rata titik keputusan di jalur, dan jarak Jaccard antar generasi peta.

```python
def hitung_metrik_keadilan(peta, awal, akhir):
    try:
        jalur_terpendek_nodes = nx.shortest_path(peta, source=awal, target=akhir)
        panjang_jalur = len(jalur_terpendek_nodes) - 1
        titik_keputusan = sum(1 for node in jalur_terpendek_nodes if peta.degree(node) > 2)
        return panjang_jalur, titik_keputusan
    except nx.NetworkXNoPath:
        return -1, -1

def hitung_jaccard_distance(set_sisi1, set_sisi2):
    irisan = len(set_sisi1.intersection(set_sisi2))
    gabungan = len(set_sisi1.union(set_sisi2))
    if gabungan == 0: return 0.0
    return 1 - (irisan / gabungan)
```

Gambar 13. Perhitungan Metrik Keadilan dan Variasi

Sumber : Dokumentasi Pribadi

D. Hasil Pengujiam

Pengujian akan menggunakan nilai k = {0, 0.05, 0.10, 0.15, 0.20}. Nilai $k = 0$ berfungsi sebagai basis yang

merepresentasikan MST tanpa tambahan sisi. Perlu dicatat bahwa program akan selalu memberikan hasil yang berbeda pada setiap eksekusinya.

- Saat $k = 0$

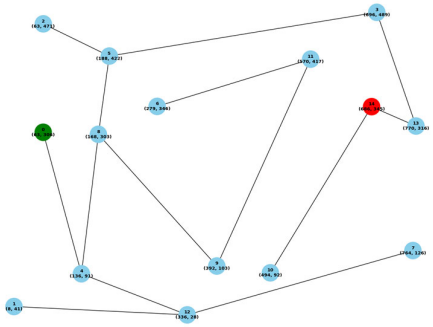
```
--- Hasil Analisis dan Statistik ---

1. Metrik Keadilan (Fairness):
  - Rata-rata Panjang Jalur Utama: 3.89
  - Standar Deviasi Panjang Jalur: 2.08
  - Rata-rata Titik Keputusan di Jalur: 1.84

2. Metrik Variasi (Variety):
  - Rata-rata Jarak Jaccard antar peta: 0.9263

  - Rata-rata Distribusi Derajat Simpul:
    - Derajat 1: 6.08 ruangan
    - Derajat 2: 5.73 ruangan
    - Derajat 3: 2.42 ruangan
    - Derajat 4: 0.66 ruangan
    - Derajat 5: 0.10 ruangan
    - Derajat 6: 0.01 ruangan
```

Gambar 14. Statistik Graf saat $k = 0$
Sumber : Dokumentasi Pribadi



Gambar 15. Contoh Graf saat $k = 0$
Sumber : Dokumentasi Pribadi

- Saat $k = 0.05$

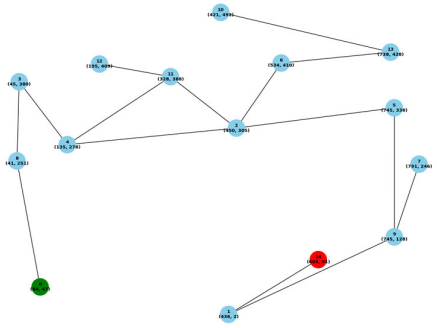
```
--- Hasil Analisis dan Statistik ---

1. Metrik Keadilan (Fairness):
  - Rata-rata Panjang Jalur Utama: 3.72
  - Standar Deviasi Panjang Jalur: 2.02
  - Rata-rata Titik Keputusan di Jalur: 2.05

2. Metrik Variasi (Variety):
  - Rata-rata Jarak Jaccard antar peta: 0.9215

  - Rata-rata Distribusi Derajat Simpul:
    - Derajat 1: 5.33 ruangan
    - Derajat 2: 5.73 ruangan
    - Derajat 3: 2.88 ruangan
    - Derajat 4: 0.92 ruangan
    - Derajat 5: 0.13 ruangan
    - Derajat 6: 0.01 ruangan
```

Gambar 16. Statistik Graf saat $k = 0.05$
Sumber : Dokumentasi Pribadi



Gambar 17. Contoh Graf saat $k = 0.05$
Sumber : Dokumentasi

- Saat $k = 0.10$

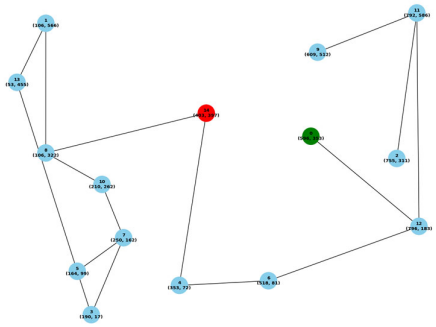
```
--- Hasil Analisis dan Statistik ---

1. Metrik Keadilan (Fairness):
  - Rata-rata Panjang Jalur Utama: 3.34
  - Standar Deviasi Panjang Jalur: 1.84
  - Rata-rata Titik Keputusan di Jalur: 2.21

2. Metrik Variasi (Variety):
  - Rata-rata Jarak Jaccard antar peta: 0.9159

  - Rata-rata Distribusi Derajat Simpul:
    - Derajat 1: 4.66 ruangan
    - Derajat 2: 5.46 ruangan
    - Derajat 3: 3.54 ruangan
    - Derajat 4: 1.10 ruangan
    - Derajat 5: 0.22 ruangan
    - Derajat 6: 0.01 ruangan
```

Gambar 18. Statistik Graf saat $k = 0.10$
Sumber : Dokumentasi Pribadi



Gambar 18. Contoh Graf saat $k = 0.10$
Sumber : Dokumentasi Pribadi

- Saat $k = 0.15$

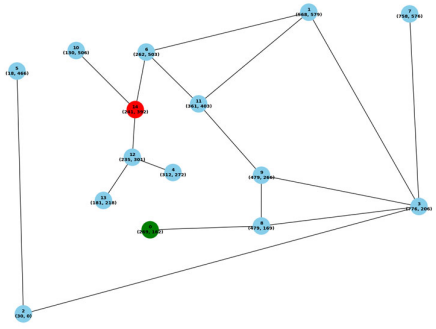
```
--- Hasil Analisis dan Statistik ---

1. Metrik Keadilan (Fairness):
  - Rata-rata Panjang Jalur Utama: 3.29
  - Standar Deviasi Panjang Jalur: 1.73
  - Rata-rata Titik Keputusan di Jalur: 2.38

2. Metrik Variasi (Variety):
  - Rata-rata Jarak Jaccard antar peta: 0.9104

  - Rata-rata Distribusi Derajat Simpul:
    - Derajat 1: 4.03 ruangan
    - Derajat 2: 5.36 ruangan
    - Derajat 3: 3.81 ruangan
    - Derajat 4: 1.41 ruangan
    - Derajat 5: 0.34 ruangan
    - Derajat 6: 0.04 ruangan
    - Derajat 7: 0.00 ruangan
```

Gambar 19. Statistik Graf saat $k = 0.15$
Sumber : Dokumentasi Pribadi



Gambar 20. Contoh Graf saat $k = 0.15$
Sumber : Dokumentasi Pribadi

- Saat $k = 0.20$

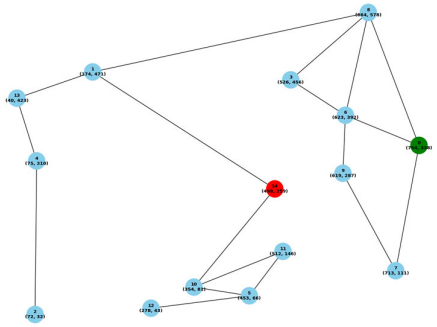
```
--- Hasil Analisis dan Statistik ---

1. Metrik Keadilan (Fairness):
  - Rata-rata Panjang Jalur Utama: 3.07
  - Standar Deviasi Panjang Jalur: 1.57
  - Rata-rata Titik Keputusan di Jalur: 2.36

2. Metrik Variasi (Variety):
  - Rata-rata Jarak Jaccard antar peta: 0.9047

  - Rata-rata Distribusi Derajat Simpul:
    - Derajat 1: 3.47 ruangan
    - Derajat 2: 5.17 ruangan
    - Derajat 3: 4.11 ruangan
    - Derajat 4: 1.73 ruangan
    - Derajat 5: 0.43 ruangan
    - Derajat 6: 0.08 ruangan
    - Derajat 7: 0.01 ruangan
```

Gambar 21. Statistik Graf saat $k = 0.20$
Sumber : Dokumentasi Pribadi



Gambar 22. Contoh Graf saat $k = 0.20$
Sumber : Dokumentasi Pribadi

E. Analisis Hasil Pengujian

Berdasarkan hasil pengujian yang telah dilakukan, didapatkan perubahan nilai variabel pada pemberian nilai k yang berbeda. Bertambahnya nilai k berbanding lurus dengan nilai rata-rata titik keputusan di jalur. Di sisi lain, nilai k berbanding terbalik dengan nilai σPL, rata-rata panjang jalur utama, dan jarak Jaccard antar generasi peta. Ini menandakan bahwa bertambahnya nilai k akan mengurangi tingkat variasi pada generasi peta di gim bertema *roguelike,* tetapi meningkatkan nilai keadilannya.

IV. KESIMPULAN

Salah satu faktor yang bisa menentukan keadilan dan variasi generasi peta pada sebuah gim bertema *roguelike* adalah persentase jumlah sisi non-MST yang ditambahkan untuk menciptakan jalur alternatif. Penentuan nilai k yang tepat akan menciptakan sebuah gim *roguelike* yang memiliki tingkat *replayability* yang tinggi, karena memiliki tinggal *fairness* dan *variety* yang seimbang.

REFERENCES

[1] Munir, Rinaldi. 2024. "Graf (Bagian 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf (Diakses pada 17 Juni 2025)

[2] Munir, Rinaldi. 2024. "Pohon (Bagian 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/23-Pohon-Bag1-2024.pdf (Diakses pada 17 Juni 2025)

[3] "Diving into procedural content generation, with WorldEngine — Smashing Magazine," *Smashing Magazine*, Mar. 28, 2016. https://www.smashingmagazine.com/2016/03/procedural-content-generation-introduction/ (Diakses pada 17 Juni 2025)

[4] Mount, David. 2020. "Computational Geometry". https://www.cs.umd.edu/class/spring2020/cmsc754/Lects/cmsc754-spring2020-lects.pdf (Diakses pada 18 Juni 2025)

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, 20 Juni 2025

Edward David Rumahorbo
13524036