Strategi Optimasi Produksi dalam Game Satisfactory Menggunakan Teori Graf sebagai Model Rantai Produksi

Wafiq Hibban Robbany - 13524016
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: robbany.wafiq@gmail.com , 13524016@std.stei.itb.ac.id

***Abstract*—*Game Satisfactory* merupakan simulasi pembangunan pabrik yang menuntut pemain untuk merancang sistem produksi kompleks dan efisien. Setiap produk yang dihasilkan memerlukan kombinasi bahan baku dan komponen antara yang diolah melalui berbagai mesin dalam suatu alur kerja terstruktur. Dalam makalah ini, penulis mengkaji bagaimana proses produksi dalam *Satisfactory* dapat dimodelkan menggunakan teori graf sebagai pendekatan utama. Pemodelan ini memungkinkan visualisasi dan analisis jalur produksi secara sistematis, termasuk identifikasi bottleneck serta optimasi throughput. Dengan pendekatan matematika diskrit, khususnya teori graf, pemetaan dan evaluasi sistem produksi menjadi lebih terstruktur dan efisien. Hasilnya menunjukkan bahwa pendekatan ini dapat membantu pemain maupun perancang sistem dalam memahami, mengoptimalkan, dan mengembangkan jalur produksi yang lebih efektif.**

Keywords—Satisfactory; Rantai Produksi; Teori Graf; DAG; Graf Berarah Asiklik Berbobot; Directed Acyclic Graph.

I. PENDAHULUAN

Game Satisfactory adalah permainan berbasis simulasi dan eksplorasi di mana pemain berperan sebagai insinyur yang membangun pabrik otomatis di dunia asing. Salah satu tantangan utama dalam permainan ini adalah membangun rantai produksi yang efisien dan terintegrasi, dari bahan mentah hingga produk akhir. Kompleksitas permainan meningkat seiring dengan bertambahnya jenis bahan dan proses manufaktur yang saling bergantung satu sama lain.

Dalam dunia nyata, sistem produksi seperti ini dapat dianalisis menggunakan pendekatan ilmiah, salah satunya adalah melalui teori graf. Teori graf merupakan bagian dari matematika diskrit yang digunakan untuk merepresentasikan dan menganalisis hubungan antar entitas. Dalam konteks produksi, entitas tersebut mencakup bahan baku, produk antara, mesin, dan produk akhir, sedangkan hubungan antar entitas menggambarkan alur transformasi produksi.

Tujuan dari makalah ini adalah untuk membangun model representasi sistem produksi dalam *Satisfactory* menggunakan graf berarah asiklik berbobot, serta menganalisis jalur produksi dari sudut pandang optimasi. Dengan menyusun graf yang merepresentasikan alur produksi, pemain dapat mengidentifikasi titik-titik kritis (bottleneck), memperkirakan kebutuhan bahan, serta merancang sistem yang lebih efisien secara sumber daya maupun waktu.

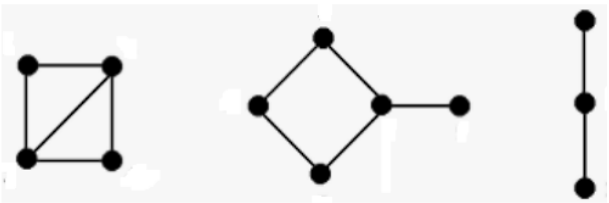
II. LANDASAN TEORI

A. Teori Graf

Graf dalam matematika dan ilmu komputer adalah struktur yang terdiri dari titik-titik (simpul/vertex) yang dihubungkan oleh garis (sisi/edge). Graf digunakan untuk merepresentasikan objek dan hubungan antar objek. Sebuah graf G dapat didefinisikan sebagai G = (V, E), yang dalam hal ini V merupakan himpunan tidak-kosong dari simpul-simpul { v1 , v2 , ... , vn }, sedangkan E merupakan himpunan sisi (edges) yang menghubungkan sepasang simpul {e1 , e2 , ... , en }.

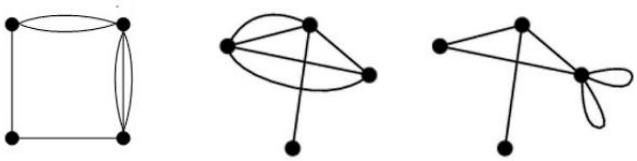
Graf dapat dibagi menjadi berbagai jenis berdasarkan beberapa kategori. Berdasarkan ada tidaknya gelang graf digolongkan menjadi dua jenis, yaitu:

1. Graf Sederhana (*simple graph*)
   Graf yang tidak mengandung gelang maupun sisi ganda dinamakan graf sederhana.



Gambar 1. Contoh Graf Sederhana
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

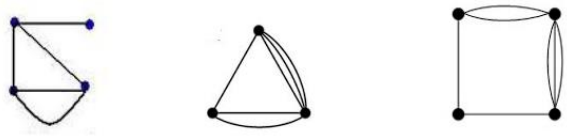
2. Graf Tak-sederhana (*unsimple-graph*)
   Graf yang mengandung sisi ganda atau gelang dinamakan graf taksederhana (*unsimple graph*).



Gambar 2. Contoh Graf Tak-sederhana
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

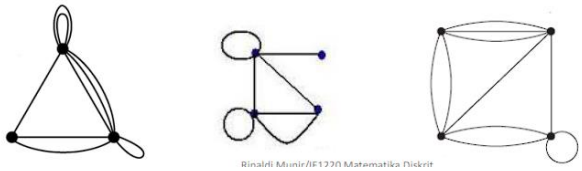
Graf tak-sederhana dibedakan lagi menjadi dua:

a. Graf Ganda
Graf yang mengandung sisi ganda



Gambar 3. Contoh Graf Ganda
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

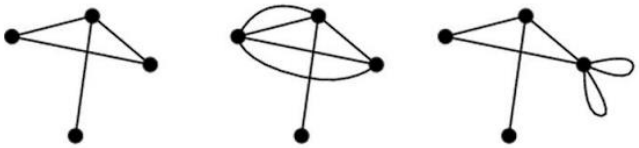
b. Graf Semu
Graf yang mengandung sisi gelang



Gambar 4. Contoh Graf Semu
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

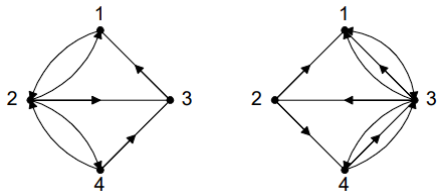
Selain itu, jenis graf dibagi berdasarkan orientasi arah pada sisi, ada dua jenis, yaitu:

1. Graf Tak-berarah (*undirected graph*)
Graf yang sisinya tidak mempunyai orientasi arah disebut graf tak-berarah.



Gambar 5. Contoh Graf Tak-berarah
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

2. Graf Berarah (*directed graph* atau *digraph*)
Graf yang setiap sisinya diberikan orientasi arah disebut sebagai graf berarah.



Gambar 6. Contoh Graf Berarah
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

Selain penggolongan jenis graf berdasarkan kategori, graf memiliki beberapa terminology, antara lain:

1. Ketetanggaan (Adjacent)
Dua buah simpul dikatakan bertetangga bila keduanya terhubung langsung.

2. Bersisian (Incidency)
Untuk sembarang sisi e = (v_j, v_k) dikatakan e bersisian dengan simpul v_j , atau e bersisian dengan simpul v_k.

3. Simpul Terpencil (Isolated Vertex)
Simpul terpencil ialah simpul yang tidak mempunyai sisi yang bersisian dengannya.

4. Graf Kosong (null graph atau empty graph)
Graf yang himpunan sisinya merupakan himpunan kosong (N_n).

5. Derajat (Degree)
Derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut. Notasi: d(v)

6. Lintasan (Path)
Lintasan yang panjangnya n dari simpul awal v0 ke simpul tujuan vn di dalam graf G ialah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk v0 , e1 , v1 , e2 , v2 ,... , vn –1 , en , vn sedemikian sehingga e1 = (v0 , v1), e2 = (v1 , v2), ... , en = (vn-1 , vn) adalah sisi-sisi dari graf G.

7. Siklus (Cycle) atau Sirkuit (Circuit)
Lintasan yang berawal dan berakhir pada simpul yang sama disebut sirkuit atau siklus.

8. Kerterhubungan (Connected)
Dua buah simpul v1 dan simpul v2 disebut terhubung jika terdapat lintasan dari v1 ke v2 . G disebut graf terhubung (connected graph) jika untuk setiap pasang simpul vi dan vj dalam himpunan V terdapat lintasan dari vi ke vj . Jika tidak, maka G disebut graf tak-terhubung (disconnected graph).

B. Directed Acyclic Graph (DAG)

Directed Acyclic Graph atau Graf berarah asiklik adalah sebuah graf berarah yang tidak memiliki siklus [2]. Dengan kata lain, tidak ada lintasan tertutup yang dimulai dan berakhir pada simpul yang sama mengikuti arah sisi.

Pada sistem produksi yang kompleks, seperti dalam game *Satisfactory,* setiap produk memiliki ketergantungan terhadap satu atau lebih bahan baku atau produk antara. Alur ini dapat direpresentasikan secara alami dalam bentuk DAG, di mana simpul menyatakan bahan baku atau produk dan sisi berarah menyatakan hubungan “dibutuhkan untuk produksi”.

Dalam makalah ini, struktur DAG diberikan tambahan bobot pada setiap sisi atau dapat disebut Weighted Directed Acyclic Graph. Bobot ini merepresentasikan jumlah unit produk yang dibutuhkan sebagai bahan baku dalam proses produksi. Dengan demikian, setiap sisi tidak hanya menunjukkan ketergantungan antar simpul, tetapi juga

menyatakan kuantitas yang diperlukan dari simpul sumber ke simpul tujuan.

C. *Model Rantai Produksi*

Model rantai produksi merupakan representasi struktural dari alur pembuatan suatu produk, mulai dari bahan mentah hingga produk akhir, melalui serangkaian tahapan produksi. Setiap tahapan melibatkan konsumsi satu atau lebih bahan untuk menghasilkan item baru, yang mungkin menjadi bahan bagi tahapan berikutnya. Dalam konteks permainan *Satisfactory*, model ini direpresentasikan sebagai graf berarah tak bersiklus, di mana simpul-simpul mewakili item dan sisi-sisi berarah menunjukkan ketergantungan bahan dalam proses produksi.

Pemodelan ini dirancang untuk memudahkan pemain merencanakan dan menghitung kebutuhan sistem produksi agar dapat mencapai kondisi 100% *efficiency*, yaitu kondisi di mana seluruh mesin dalam pabrik berjalan tanpa henti, mendapat suplai bahan yang cukup, serta menghasilkan produk secara stabil dan konsisten.

Dengan efisiensi penuh, pemain dapat mengoptimalkan penggunaan daya dan menjaga kestabilan sistem secara keseluruhan. Hal ini menjadi sangat penting karena sistem pembangkit listrik pada permainan Satisfactory juga memiliki keterbatasan kapasitas. Ketidakstabilan dalam sistem produksi, seperti kekurangan bahan atau lonjakan konsumsi mendadak, dapat menimbulkan fluktuasi konsumsi daya yang sulit diprediksi. Jika konsumsi daya melebihi kapasitas yang tersedia maka seluruh sistem pabrik akan mengalami *shutdown* secara otomatis sehingga diperlukan intervensi manual untuk pemulihannya.

III. PEMBAHASAN DAN IMPLEMENTASI

Pada bagian ini dijelaskan bagaimana pemodelan rantai produksi dalam permainan *Satisfactory* diimplementasikan menggunakan bahasa pemrograman Python dengan pendekatan berbasis Graf Berarah Asiklik Berbobot (Weighted DAG). Program ini dibangun dengan memanfaatkan beberapa pustaka Python seperti:

- **networkx** untuk pemodelan graf dan relasi antar item produksi,
- **matplotlib** untuk visualisasi struktur produksi,
- dan **json** untuk membaca struktur data dari berkas resep permainan.

Tujuan dari implementasi ini adalah untuk memungkinkan pemain menghitung kebutuhan bahan mentah, produk antara, dan jumlah mesin secara akurat berdasarkan jumlah produk akhir yang ingin dihasilkan setiap menit. Seluruh fungsi dalam program dibagi secara modular agar mempermudah pengelolaan dan pengembangan lebih lanjut.

A. *Pemrosesan dan Pembentukan Graf Produksi*

Langkah pertama dalam program adalah membaca file “data1.0.json” yang berisi semua resep produksi dan item-item di dalam permainan. Data ini kemudian digunakan untuk membentuk struktur graf yang merepresentasikan alur produksi.

```python
with open('data1.0.json', 'r', encoding='utf-8') as f:
    data = json.load(f)

G = nx.DiGraph()
item_names = {}
producible_items = {}
recipe_machines = {}
recipes = data.get('recipes', {})
items_data = data.get('items', {})

for recipe_key, recipe in recipes.items():
    if (not recipe.get('inMachine', False) or recipe.get('alternate', False)
        or 'Desc_Converter_C' in recipe.get('producedIn', [])):
        continue
    products = recipe.get('products', [])
    ingredients = recipe.get('ingredients', [])

    if products:
        product = products[0]['item']
        product_name = recipe['name']
        produced_in = recipe.get('producedIn', [])
        producible_items[product] = product_name
        item_names[product] = product_name
        machine_name = produced_in[0] if produced_in else "Unknown Machine"
        recipe_machines[product] = machine_name
        output_quantity = products[0]['amount']

        for ing in ingredients:
            src = ing['item']
            dst = product
            weight = ing['amount']
            G.add_edge(src, dst, weight=weight, output_qty=output_quantity)

for item_key, item_data in items_data.items():
    if item_key not in item_names:
        item_names[item_key] = item_data.get('name', item_key)

G_labeled = nx.relabel_nodes(G, lambda n: item_names.get(n, n))

```

Gambar 7. Dokumentasi Source Code
Sumber: Dokumen Penulis

Pada bagian ini, setiap item direpresentasikan sebagai simpul (node), sedangkan relasi antara bahan baku dan produk hasil produksi direpresentasikan sebagai sisi berarah (edge). Setiap sisi memuat atribut tambahan berupa bobot (weight), yaitu jumlah bahan yang diperlukan untuk satu kali proses produksi, serta output_qty, yaitu jumlah produk yang dihasilkan dalam satu batch. Agar hasil pemodelan akurat dan konsisten, hanya resep-resep yang diproses menggunakan mesin, bukan merupakan alternate recipe, dan tidak melibatkan converter yang dimasukkan ke dalam graf.

Setelah seluruh relasi antar item dimasukkan ke dalam graf, dilakukan proses pelabelan ulang (relabeling) simpul dengan nama-nama item yang lebih mudah dibaca agar graf yang dihasilkan dapat divisualisasikan secara lebih informatif.

B. *Perhitungan Kebutuhan Produksi*

Setelah struktur graf berhasil dibentuk, langkah berikutnya adalah menghitung jumlah kebutuhan bahan mentah dan produk antara yang dibutuhkan untuk memproduksi suatu produk akhir dalam jumlah tertentu per menit. Hal ini bertujuan agar pemain dapat mengetahui secara pasti jumlah bahan dan sumber daya yang diperlukan sebagai suplai untuk

sistem agar produksi dapat berjalan lancer tanpa hambatan (*bottleneck*).

```python
def calculate_requirements(target_item, quantity=1, requirements=None):
    """Menghitung kebutuhan total bahan baku untuk memproduksi item target dengan
    rasio produksi yang benar"""
    if requirements is None:
        requirements = {}

    if target_item not in G_labeled or G_labeled.in_degree(target_item) == 0:
        requirements[target_item] = requirements.get(target_item, 0) + quantity
        return requirements

    original_key = None
    for key, name in item_names.items():
        if name == target_item:
            original_key = key
            break

    output_ratio = 1.0
    if original_key:
        for recipe in recipes.values():
            if recipe.get('products') and recipe['products'][0]['item'] == original_key:
                if not recipe.get('alternate', False) and recipe.get('inMachine', False):
                    output_ratio = recipe['products'][0]['amount']
                    break

    for pred in G_labeled.predecessors(target_item):
        edge_data = G_labeled[pred][target_item]
        input_needed = edge_data['weight']
        batches_needed = quantity / output_ratio
        total_input_needed = input_needed * batches_needed
        calculate_requirements(pred, total_input_needed, requirements)

    return requirements
```

Gambar 8. Dokumentasi Source Code
Sumber: Dokumen Penulis

Fungsi yang bertanggungjawab pada langkah ini adalah calculate_requirements(target_item, quantity), yang memiliki peran utama dalam menelusuri jalur produksi dari produk akhir hingga ke bahan mentah. Tujuannya adalah untuk menghitung secara menyeluruh jumlah tiap komponen yang dibutuhkan agar dapat mempertahankan laju produksi tertentu.

Perhitungan dilakukan dengan pendekatan *bottom-up traversal*, yaitu menjelajahi graf dari simpul produk akhir ke simpul-simpul penyusunnya, termasuk produk antara dan bahan mentah. Pada setiap tahapan, perhitungan jumlah bahan dilakukan berdasarkan rasio yang diperoleh dari resep produksi yang sesuai. Langkah-langkah utama yang dilakukan sebagai berikut:

1. Cek apakah item target memiliki simpul pendahulu (*predecessor*). Jika tidak, maka item tersebut dikategorikan sebagai bahan mentah dan langsung dicatat ke daftar kebutuhan (*requirements*).
2. Jika item bukan merupakan bahan mentah atau merupakan hasil dari proses produksi, maka program akan mencari resep yang memproduksi item tersebut dalam struktur data JSON. Informasi yang diambil adalah jumlah produksi yang dihasilkan dan bahan yang dibutuhkan per *batch*.
3. Untuk setiap pendahulu pada graf, fungsi akan melakukan rekursi untuk menghitung bahan dari setiap pendahulu hingga seluruh jalur produksi dari produk akhir ke bahan mentah.

C. Estimasi Jumlah Mesin Produksi

Setelah kebutuhan setiap item per menit dihitung, langkah berikutnya adalah memperkirakan jumlah mesin produksi yang diperlukan untuk mencapai laju produksi tersebut. Hal ini penting agar seluruh sistem produksi berjalan dengan efisiensi 100%, di mana setiap mesin dapat terus beroperasi tanpa jeda karena kekurangan bahan maupun kelebihan kapasitas.

```python
def calculate_machine_count(item_name, quantity_per_minute):
    original_key = None
    for key, name in item_names.items():
        if name == item_name:
            original_key = key
            break

    if original_key:
        for recipe in recipes.values():
            if recipe.get('products') and recipe['products'][0]['item'] == original_key:
                if not recipe.get('alternate', False) and recipe.get('inMachine', False):
                    time_per_cycle = recipe['time']
                    output_per_cycle = recipe['products'][0]['amount']
                    return (time_per_cycle * quantity_per_minute) / (60 * output_per_cycle)
    return 0.0
```

Gambar 9. Dokumentasi Source Code
Sumber: Dokumen Penulis

Fungsi calculate_machine_count(item_name, quantity_per_minute) digunakan untuk menghitung jumlah mesin yang dibutuhkan untuk memproduksi suatu item dengan target kuantitas tertentu per minute. Perhitungan memperhitungkan variable waktu siklus produksi (*time*) dan jumlah produksi yang dihasilkan per siklus (*amount*) pada data JSON. Persamaan yang dirumuskan sebagai berikut:

$$Machine\_count = (time * quantity/minute) / (60 * amount) \ldots \quad (1)$$

Dengan *quantity/minute* merupakan jumlah item yang ingin diproduksi tiap menit.

D. Visualisasi Rantai Produksi Menggunakan DAG

Setelah proses perhitungan jumlah bahan baku serta estimasi kebutuhan jumlah mesin produksi dilakukan, tahap selanjutnya adalah memvisualisasikan seluruh alur rantai produksi dalam bentuk graf berarah. Visualisasi ini dirancang agar menyerupai flowchart horizontal, di mana aliran proses produksi ditata dari kiri ke kanan, dimulai dari bahan mentah, dilanjutkan ke produk antara, dan berakhir pada produk akhir. Representasi visual semacam ini sangat membantu dalam memahami struktur produksi secara menyeluruh, terutama ketika jalur produksi melibatkan banyak komponen dan tahap manufaktur.

Fungsi yang bertanggung jawab untuk menghasilkan visualisasi tersebut adalah visualize_production_chain(target_item, target_quantity). Fungsi ini menerima dua parameter utama: nama produk yang ingin diproduksi (*target_item*), dan jumlah unit yang ingin dihasilkan per menit (*target_quantity*). Berdasarkan masukan tersebut, fungsi ini akan menyusun subgraf dari keseluruhan graf produksi yang hanya mencakup jalur produksi relevan terhadap produk tersebut.

Fungsi ini menghasilkan grafik visual berupa DAG dari sistem produksi untuk suatu jenis produk. Informasi yang ditampilkan mencakup:

- Jumlah bahan mentah dan antara yang dibutuhkan per menit, dihitung dari hasil propagasi kebutuhan yang dilakukan fungsi calculate_requirements().
- Nama dan jumlah mesin yang digunakan untuk memproduksi setiap produk antara maupun akhir.

- Jumlah bahan yang diperlukan pada setiap sisi, berdasarkan perhitungan jumlah batch yang dibutuhkan dikalikan dengan jumlah bahan per batch. Informasi ini ditampilkan di tengah panah yang menghubungkan dua simpul.

Untuk memperjelas struktur produksi, setiap simpul pada graf diberi warna sesuai dengan kategori item pada simpul. Cokelat untuk bahan mentah (*raw material*), yaitu item yang tidak diproduksi oleh mesin lain. Oranye untuk produk antara, yaitu item yang diproduksi oleh suatu mesin dan digunakan kembali sebagai bahan untuk membuat produk lain. Dan hijau untuk produk akhir, yaitu produk target yang telah ditentukan oleh pengguna sebagai hasil akhir dari proses produksi.

Gambar 10. Visualisasi Rantai Produksi
Sumber: Dokumen Penulis

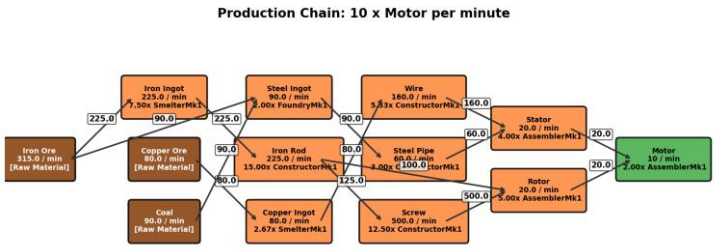
E. Eksekusi Program

Setelah seluruh fungsi telah dirancang, langkah terakhir adalah mengeksekusi program untuk mensimulasikan rantai produksi berdasarkan input pengguna

```
Masukkan nama item yang ingin diproduksi: Motor
Berapa Motor yang ingin diproduksi per menit? 10
```

Gambar 12. Output Eksekusi Program
Sumber: Dokumen Penulis

Pengguna diminta memasukkan nama produk target dan jumlah yang ingin diproduksi dalam semenit



Gambar 12. Visualisasi Rantai Produksi
Sumber: Dokumen Penulis

Setelah menghitung, hasil visualisasi akan ditampilkan dalam bentuk DAG. Pada **Gambar 12**, dapat diamati bahwa untuk memproduksi 10 Motor per menit, dibutuhkan sejumlah bahan mentah utama, antara lain 315 unit Iron Ore, 80 unit Copper Ore, dan 90 unit Coal. Selain kebutuhan bahan, system juga membutuhkan sejumlah mesin produksi. Misalnya, dibutuhkan 2 unit Assembler yang dikonfigurasi untuk merakit Motor, serta 7.5 unit SmelterMk1 untuk melebur Iron Ore menjadi Iron Ingot.

Perhitungan 7.5 unit SmelterMk1 ini berarti dapat direalisasikan dengan 7 unit Smelter yang berjalan secara normal yaitu pada 100% *clock speed,* dan 1 unit SmelterMk1 tambahan yang dikonfigurasi pada 50% clock speed, pengaturan ini juga dikenal dengan istilah *underclock.* Pengaturan ini umum dilakukan pada permainan *Satisfactory* untuk mencapai efisiensi penuh.

Selanjutnya, alur produksi yang divisualisasikan menunjukkan tahapan transformasi dari bahan mentah hingga menjadi produk akhir. Misalnya, Iron Ore diproses menjadi Iron Ingot, lalu menjadi Iron Rod. Setelah itu, Iron Rod digunakan dalam dua jalur produksi yang berbeda. Sebagian Iron Rod diproses lebih lanjut menjadi Screw melalui Constructor, sementara sisanya langsung digunakan Bersama Screw untuk merakit Rotor. Hal ini mencerminkan bahwa dalam rantai produksi, satu bahan dapat memiliki peran ganda dalam tahapan perakitan berikutnya.

Visualisasi ini memberikan gambaran menyeluruh tentang rantai produksi serta keterkaitan antar komponen dan mesin dalam sistem yang optimal.

IV. KESIMPULAN

Melalui pemodelan rantai produksi dalam permainan *Satisfactory* menggunakan pendekatan graf berarah tak bersiklus (Directed Acyclic Graph/DAG), diperoleh sebuah sistem yang mampu memvisualisasikan dan menghitung seluruh kebutuhan produksi secara sistematis. Setiap simpul dalam graf merepresentasikan produk (baik mentah, antara, maupun akhir), sedangkan sisi menunjukkan hubungan ketergantungan bahan dengan bobot yang menyatakan jumlah yang dibutuhkan.

Pemanfaatan struktur graf ini tidak hanya mempermudah perencanaan produksi, tetapi juga memungkinkan pemain untuk mencapai *100% efficiency*, yaitu kondisi optimal di mana semua mesin bekerja secara kontinu, bahan baku tersedia tepat waktu, dan konsumsi daya stabil.

Model ini juga menunjukkan bahwa konsep-konsep dalam matematika diskrit, khususnya teori graf, dapat diterapkan secara langsung dalam konteks perencanaan dan optimalisasi produksi dalam dunia permainan maupun dunia nyata. Maka dari itu, pemodelan ini tidak hanya berguna sebagai alat bantu permainan, tetapi juga menjadi contoh konkret pemanfaatan teori graf dalam menyelesaikan permasalahan kompleks yang melibatkan rantai produksi.

V. LAMPIRAN

Source code dalam bahasa pemrograman Python dapat diakses melalui link berikut.

https://github.com/wafiqhibban/Rantai-Produksi-Satisfactory

VI. Ucapan Terima Kasih

Segala puji dan syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa atas segala rahmat, karunia, dan kemudahan yang telah diberikan, sehingga penulis dapat menyelesaikan makalah ini dengan baik.

Penulis menyampaikan terima kasih yang sebesar-besarnya kepada Pak Arrival Dwi Sentosa, S.Kom., M.T., selaku dosen pengampu mata kuliah IF1220 Matematika Diskrit kelas K-02, atas ilmu, arahan, dan bimbingan yang menjadi dasar dalam penyusunan makalah ini.

Ucapan terima kasih juga penulis sampaikan kepada Bapak dan Ibu penulis, atas doa dan dukungan yang tiada henti, baik secara moral maupun semangat, selama proses pengerjaan makalah.

References

[1] Munir, R. Matematika Diskrit. Bandung: Informatika 2025. Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/ pada tanggal 20 Juni 2025

[2] GeeksforGeeks. *Introduction to Directed Acyclic Graph (DAG)*. Diakses dari https://www.geeksforgeeks.org/dsa/introduction-to-directed-acyclic-graph/ pada tanggal 20 Juni 2025.

[3] NetworkX Developers. *DiGraph — NetworkX 3.3 documentation*. Diakses dari https://networkx.org/documentation/stable/reference/classes/digraph.html pada tanggal 20 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

Wafiq Hibban Robbany - 13524016