Laporan Tugas Besar IF4070

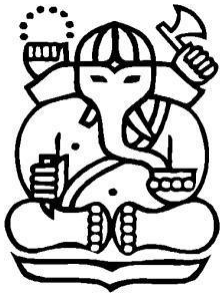
Graf Pengetahuan dan Sistem RAG

oleh :

Imanuel Sebastian Girsang / 13522108

Muhammad Neo Cicero Koda / 13522108

Derwin Rustanly / 13522115



PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA INSTITUT TEKNOLOGI BANDUNG

2025

Bab I

Domain Persoalan

Bab ini menjelaskan tentang *domain* yang dipilih pada tugas ini agar pembaca memiliki konteks yang diperlukan untuk memahami bab-bab selanjutnya.

1.1 Latar Belakang Pemilihan Domain

Gambar I.1 Permainan Clash Royale

Kami memilih *domain* Clash Royale karena alasan yang cukup sederhana. Kami bertiga merupakan pecinta *game* Clash Royale dan kami lumayan *expert* dalam permainan tersebut (sekitar 8000 *trophy*). Tidak hanya itu, kami juga lumayan kompetitif karena kami tidak hanya bermain untuk mencari kesenangan, tetapi kami juga bermain untuk menang. Berbagai video mengenai cara-cara untuk mendapatkan keuntungan dalam permainan, *deck building strategies*, dan *meta* (Most Effective Tactic Available) analysis telah kami pelajari selama bertahun-tahun bermain.

Sebagai pemain yang serius, kami sering menghadapi tantangan dalam membangun *deck* yang optimal. Pertanyaan seperti, "Apakah deck ini sudah *balance*?", "Kartu apa yang kurang dari *deck* ini?", atau "Kenapa *deck* ini sering kalah?" adalah pertanyaan yang sering muncul. Meskipun kami sudah berpengalaman, terkadang tetap sulit untuk mengidentifikasi kelemahan deck secara objektif, terutama ketika mencoba *archetype* baru atau bereksperimen dengan kombinasi kartu yang baru.

Dari pengalaman pribadi ini, kami melihat kesempatan untuk membuat sistem yang dapat membantu pemain, baik pemula maupun yang sudah *experienced*, untuk menganalisis dan memperbaiki deck mereka secara sistematis.

Tidak hanya itu, Clash Royale sebagai domain memiliki beberapa keunggulan praktis untuk tugas ini:

- ***Familiar*:** Sebagai pemain aktif, kami memahami seluk-beluk dari permainan ini.
- ***Rich knowledge base*:** Terdapat banyak sumber daya (YouTube Guides, *wiki*, RoyaleAPI) yang dapat kami gunakan untuk validasi dan sumber pengetahuan.
- **Struktur jelas:** Properti kartu, *archetypes*, dan konsep-konsep strategi memiliki definisi yang jelas.
- **Nilai praktis:** Sistem yang dibuat dapat benar-benar berguna untuk pemain lain.

1.2 Penjelasan Clash Royale

Clash Royale adalah permainan *real-time mobile strategy* yang dikembangkan dan dipublikasikan oleh Supercell pada tahun 2016. Sejak diluncurkan, Clash Royale telah menjadi salah satu permainan *mobile* paling populer dengan lebih dari 100 juta *download* dan ekosistem *e-sports* yang aktif dengan turnamen global berhadiah jutaan dolar.

1.2.1 Konsep Dasar Permainan

Gambar I.2 Contoh *gameplay* Clash Royale

Sistem *gameplay* dalam Clash Royale menggunakan *player versus player* (PvP), di mana dua pemain bertarung dalam arena *real-time* dengan tujuan menghancurkan menara lawan sambil mempertahankan menara sendiri. Setiap pertandingan berlangsung selama tiga menit dengan kemungkinan tambahan tiga menit *overtime* jika diperlukan.

1.2.2 Arena dan Menara

Berikut merupakan penjelasan terkait arena dan menara pada *clash royale*:

- Setiap pemain memiliki 3 menara, 1 King Tower (di tengah) dan 2 Princess Tower (di kiri dan kanan).
- Menghancurkan 1 Princess Tower memberikan 1 *crown* dan membuka akses ke King Tower.
- Menghancurkan King Tower langsung memberikan 3 *crowns* dan kemenangan instan.
- Pemain dengan *crown* terbanyak di akhir waktu memenangkan pertandingan.
- Jika hingga waktu habis tidak ada *tower* yang dihancurkan, maka pemain dengan darah terendah di salah satu *tower*-nya akan otomatis kalah.

1.2.3 Deck dan Kartu

Gambar I.3. Contoh kartu di Clash Royale

Berikut merupakan penjelasan terkait *deck* dan kartu pada *clash royale*:

- Setiap pemain membuat *deck* yang terdiri dari 8 kartu sebelum pertandingan.
- Kartu dibagi menjadi 4 kategori utama:
  - **Troops**: Unit yang dapat bergerak dan menyerang.
  - **Buildings**: Bangunan yang ditempatkan di arena.
  - **Spells**: Efek instan yang memberikan *damage* atau *utility*.
  - **Champion**: Troop khusus yang dapat memiliki *ability* untuk digunakan dalam arena.
- Pada awal permainan, tangan pemain berisi 4 kartu acak dari *deck* mereka.
- Setelah kartu dimainkan, kartu berikutnya dari *deck* akan muncul (*card rotation*).
- Kartu tersebut hanya dapat muncul kembali apabila nantinya pemain telah mendapatkan 7 kartu lainnya pada tangan (1 *cycle*).

1.2.4 Elixir System

Sumber daya utama yang diperlukan untuk memainkan kartu adalah *elixir*. Berikut merupakan penjelasan terkait *elixir*:

- Setiap pemain memiliki *bar elixir* yang maksimal berisi 10 *elixir*.
- Elixir akan terisi ulangsecara otomatis tergantung dengan periode dalam permainan:
  - *Normal time*: 1 *elixir* per ~3.6 detik.
  - *Double elixir*/*overtime*: 1 *elixir* per ~1.8 detik.
  - *Triple elixir* / 1 menit terakhir dalam *overtime* : 1 *elixir* per ~ 1.2 detik.
- Setiap kartu memiliki biaya *elixir* yang berbeda (1-10 *elixir*).

1.3 Mekanisme Permainan Clash Royale

Berikut merupakan penjelasan terkait mode permainan utama Clash Royale.

1.3.1 Pre-Game

Fase *pre-game* meliputi masa sebelum pemain bertarung dengan musuh, isinya adalah:

- Pemain memilih 8 kartu untuk mereka gunakan dalam pertandingan.
- Pemain mencari musuh untuk bertarung dengannya.
- Apabila musuh telah ditemukan, pemain dapat kemudian melihat *trophy* dan nama dari lawan. Akan tetapi, mereka tidak dapat melihat *deck*-nya.

1.3.2 In-Game:

Fase *in-game* meliputi masa saat pemain bertarung dengan musuh, isinya adalah:

- Pertandingan dimulai dengan *starting hand* (4 kartu acak).
- Pemain dapat menaruh kartu di arena dengan menyetuh lokasi yang diinginkan.
- Kartu yang dimainkan akan memakan *elixir* sesuai *cost*-nya.

- Troops bergerak otomatis menuju target terdekat yang mereka dapat lihat.
- Buildings tetap di lokasi penempatan selama *lifetime* mereka.
- Spells memberikan efek instan di area yang dipilih.
- Pemain saling bertukar serangan dengan tujuan mendapatkan *crown* lebih banyak dari musuh.

1.3.3 Victory Conditions

Agar salah satu pemain dinyatakan menang, pemain harus memenuhi salah satu dari tiga kondisi berikut:

- Menghancurkan King Tower musuh
- Mendapatkan lebih banyak *crown* dari musuh
- Jika pertandingan memasuki **overtime**, pemenang ditentukan oleh ***tower* yang memiliki sisa HP lebih tinggi** daripada milik lawan (atau jika salah satu pemain berhasil mendapatkan 1 *crown* lebih dulu).

1.4 Pembatasan Domain Clash Royale

Untuk menjaga kompleksitas agar sesuai dengan tujuan pembelajaran dan *scope* tugas, kami membatasi domain pada aspek yang akan dijelaskan pada bagian-bagian selanjutnya.

1.4.1 Tujuan Utama

Clash Royale merupakan permainan yang sangat kompleks dan terdapat banyak sekali hal yang bisa dieksplor pada permainan ini. Hal-hal seperti penempatan kartu, strategi untuk mendapatkan *positive elixir trade,* dan mekanisme *kiting* tidak menjadi hal yang esensial untuk kemenangan. Namun, satu-satunya hal yang kami fokuskan pada sistem ini adalah membentuk *deck*. Kami tidak mempertimbangkan pula bagaimana suatu *deck* berinteraksi dengan *deck* lain. Sebab, secara natural, seluruh *deck* memiliki kekuatan dan kelemahan ketika menghadapi komposisi tertentu.

1.4.2 Data dan Properti Kartu

Data yang digunakan pada sistem yang dibuat meliputi:

- **Informasi dasar**: *elixir cost*, *damage*, *hitpoints*, dan *range*.
- **Sifat kategorikal**: *card type* (Troop/Building/Spell), *rarity*, *target type*, *damage type*.
- **Flag *boolean***: *splash damage*, *flying*, *spawns units*.

Adapun data yang diabaikan pada sistem yang dibuat meliputi:

- **Level kartu**: Semua kartu diasumsikan berada pada *tournament standard* (Level 11).
- **Mekansime *evolution***: *Evolution* tidak dipertimbangkan.
- ***Balance changes***: Statistik kartu yang digunakan adalah statistik konstan dan tidak di-*update* secara berkala.
- **Interaksi detil**: Hal seperti *stun duration*, *knockback distance*, *spawn timing*, dan kecepatan gerak tidak dimodelkan.

Justifikasi: Kami fokus pada analisis yang independen dari level kartu atau *micro-interactions*. Tujuannya adalah menganalisis struktur *deck* dan bukan mensimulasikan *gameplay* detil.

1.4.3 Sumber Data

Sumner utama yang kami gunakan dalam membangun *knowledge graph* meliputi:

- **RoyaleAPI** (https://royaleapi.com): Digunakan untuk mendapat statistik kartu.
- **Clash Royale Wiki**: Digunakan utnuk mendapat interkasi antarkatu.
- **Pengalaman pribadi**: Diambil dari pengalaman bermain pribadi kami bertiga.
- **YouTube**: Konten edukatif dari pemain tingkat tinggi (SirTag, B-Rad, BestNA, dll.).
- **Top ladder decks**: Analisis *deck* yang digunakan oleh Top 100 pemain global.

Untuk metode koleksi data, kami melakukan *web scraping* dari RoyaleAPI untuk mendapat properti kartu. Beberapa sumber lain kami cek untuk *cross-reference*. Validasi manual hasil analisis *deck* dilakukan dengan *in-game testing*.

1.5 Kecocokan Domain dengan Pengembangan KBS

Domain Clash Royale Deck Analysis sangat cocok untuk dikembangkan menjadi sistem berbasis pengetahuan. Bagian-bagian selanjutnya akan menjelaskan alasan kecocokan tersebut.

1.5.1 Aturan yang Eksplisit dan Dapat Diartikulasikan

Berbeda dengan permainan yang berbasis kemampuan mekanis (seperti permainan FPS), Clash Royale memiliki ***strategic layer*** dengan aturan-aturan yang dapat dijelaskan secara verbal.

Berikut merupakan contoh cara untuk mendefinisikan *deck archetype* dalam Clash Royale:

- "Beatdown Deck HARUS memiliki *heavy tank* dengan *elixir cost* $\geq$5 dan HP $>$ 2000"
- "Cycle Deck HARUS memiliki *average elixir* $\leq$ 3.3 dan setidaknya 3 kartu dengan *elixir cost* berupa 1-2"
- "Siege Deck HARUS menggunakan X-Bow atau Mortar sebagai *primary win condition*"

Selain itu, Berikut merupakan contoh cara untuk mendefinisikan aturan komposisi *deck* dalam Clash Royale:

- "Deck tanpa kartu yang bisa menargetkan pasukan udara PASTI rentan terhadap *air-heavy decks*"
- "Deck dengan lebih dari 4 *spells* KURANG memiliki pasukan untuk *defense*"
- "Deck dengan *average elixir* > 4.8 TERLALU memiliki rotasi kartu yang terlalu lambat"

Aturan-aturan ini bersifat ***deterministic*** **dan** ***explainable*****,** setidaknya untuk META. Pemain profesional dapat menjelaskan alasan di balik setiap *rule*. Selain itu, skema aturan ini memungkinkan *translation* langsung ke *knowledge base*. Ini sangat kontras dengan domain yang memerlukan *machine learning* karena aturan yang terlalu kompleks atau *implicit*.

1.5.2 Domain Kaya Pengetahuan

Clash Royale memiliki *ecosystem* yang kaya dengan pengetahuan. Top 100 pemain dunia secara terbuka membagikan *deck* yang mereka gunakan. Banyak kreator konten yang membuat tutorial menggunakan suatu *deck*. Tutorial buat suatu *deck archetype* tertentu juga sering dibuat. Sumber daya dari komunitas seperti RoyaleAPI dan *subreddit* Clash Royale juga berfungsi sebagai sumber pengetahuan. Data turnamen, seperti dari Clash Royale League (CRL), dapat digunakan untuk memvalidasi jenis kombinasi *deck* yang berkinerja tinggi dan yang kurang baik.

1.5.3 Kegunaan Praktis

Sistem yang dikembangkan memiliki kegunaan praktis untuk berbagai jenis pemain. Untuk pemain pemula, sistem ini memberikan bahan ajar tentang dasar-dasar membangun *deck* yang baik dan alasan di balik kombinasi-kombinasi kartu. Pemain menengah dapat menggunakan sistem ini untuk menyempurnakan *deck* mereka atau untuk mengeksplorasi *archetype* baru. Untuk pemain lanjut, sistem ini dapat digunakan untuk memvalidasi ide *deck* yang bersifat eksperimental secara cepat.

Bab II

Graf Pengetahuan

Bab ini menjelaskan tentang graf pengetahuan (*knowledge graph*) yang telah disusun untuk tugas ini.

2.1 Pemanfaatan secara Garis Besar

Sistem yang dibuat bertujuan untuk menyajikan informasi terkait strategi dalam permainan Clash Royale. Pada sistem ini, *knowledge graph* digunakan sebagai basis data terstruktur yang menyimpan informasi tentang kartu yang ada, sifat-sifatnya, serta hubungan antarkartu. Data kartu seperti jumlah *hitpoints*, *damage*, dan *elixir cost* disimpan agar model tidak halusinasi ketika menjawab pertanyaan. Hubungan yang sebelumnya bersifat implisit, seperti kartu apa yang mengalahkan kartu lain dan kartu apa yang bekerja sama secara baik dengan kartu lain, juga dapat dibuat menjadi eksplisit. Dalam sistem RAG, graf ini digunakan oleh komponen *retriever* menggunakan kueri Cypher untuk mengambil konteks yang relevan sebelum jawaban disusun oleh Generator.

2.2 Deskripsi Label dan Simpul

Deskripsi label dan simpul dapat dilihat pada file schema.py. Berikut merupakan penjelasan detil tentang label dan simpul yang ada.

Tabel II.1 berikut berisi deksripsi detil tentang setiap *node* yang ada dalam sistem.

Tabel II.1 Deskripsi *node* pada sistem

Node	Deskripsi	Properti	Deskripsi Properti
Card	Berisi informasi tentang suatu kartu beserta statistiknya	name	Nama kartu
		elixir	Harga *elixir* kartu
		type	Tipe kartu (*troop*, *building*, dll.)
		rarity	Tingkat *rarity* kartu (*common*, *rare*, dll.)
		arena	Arena kartu tersebut pertama ditemukan
		transport	Metode pergerakan kartu

		hitpoints	Jumlah *hitpoints*
		damage	Jumlah *damage* yang diberikan kartu
		dps	Jumlah *damage per second* kartu
		description	Deskripsi kartu
		level11_stats	Statistik kartu pada level 11
Rarity	Tingkat *rarity* kartu (*common*, *rare*, dll.)	name	Nama *rarity*
		color	Warna *rarity*
Arena	Arena kartu tersebut pertama kali dapat diperoleh	name	Nama arena
		number	Nomor arena
		trophy_requirements	Jumlah piala minimum untuk mencapai arena
Target	Jenis kartu yang ditargetkan	name	Nama target
Archetype	Klasifikasi *archetype* suatu *deck*	name	Nama *archetype*
		description	Deskripsi
		avg_elixir_range	Rentang *elixir archetype*
Type	Kategori tipe kartu (*troop*, *spell*, dll.)	name	Nama tipe

Tabel II.2 berisi hubungan antar-*node* yang berlaku pada sistem.

Tabel II.2 Deskripsi *relationships* pada sistem

Relationship	**From Label**	**To Label**	**Deskripsi**	**Extra Properties**
Has Rarity	Card	Rarity	Menunjukkan *rarity* yang dimiliki kartu	-

				-
Unlocks In	Card	Arena	Menunjukkan pada arena apa suatu kartu mulai dapat diperoleh	-
Can Hit	Card	Target	Mendefinisikan jenis kartu lain yang dapat ditargetkan oleh Kartu	-
Has Type	Card	Type	Mendefinisikan tipe kartu (*troop*, *building*, dll.)	-
Counters	Card	Card	Mendefinisikan kartu from_label yang merupakan lawan baik (*counter*) dari kartu to_label	**synergy_type:** Bagaimana kedua kartu bekerja sama. Contohnya, Golem dan Night Witch bersinergi *tank-support* **strength:** Tingkat kekuatan kombinasi
Fits Archetype	Card	Archetype	Mendefinisikan *archetype* yang cocok untuk suatu kartu	**role:** Mendefinisikan peran kartu dalam *archetype*. Contohnya, Golem adalah *main-tank* dalam *beatdown*.

2.3 Keterkaitan dengan Ontologi

Graf pengetahuan yang dibuat pada proyek ini memanfaatkan sistem berbasis pengetahuan yang sudah dirancang pada proyek sebelumnya. Dalam Prolog, suatu entitas didefinisikan

menggunakan predikat. Dalam Neo4j, predikat tersebut diubah menjadi Node yang memiliiki Properties.. Sebagai contoh, dalam Prolog, untuk mendefinisikan kartu Giant yang berupa *troop* dengan harga *elixir* 5, ditulis *card(giant, troop)* dan *elixir_cost(giant, 5*). Dalam Neo4j, hal ini dilakukan dengan membuat Node *:Card* dengan properti berupa *{name: "Giant", type: "troop", elixir:5}*.

Sistem *rules* pada proyek sebelumnya diubah menjadi *relationship*. Dalam kata lain, pengetahuan yang sebelumnya implisit dan dihasilkan melalui inferensi di Prolog, dijadikan eksplisit/tersimpan dalam Neo4j untuk meningkatkan kinerja RAG. Sebagai contoh, dalam Prolog, hubungan seperti *counter* seringkali berupa aturan yang diturunkan pada saat ditanya. Dalam *knowledge graph*, logika ini dinalarkan sekali pada awal (*ingestion*) menggunakan *script* pada *relationship_rules.py* dan hasilnya disimpan secara permanen pada *knowledge graph*.

Beberapa aturan kompleks pada *rules.pl* tidak bisa disimpan sebagai *node* dan *relationship* pada graf. Sebagai contoh, logika analisis deck yang kompleks pada Prolog dipindahkan ke Python dalam modul DeckAnalyzer.

Bab III

Sistem RAG

3.1 Fungsionalitas Penting

Alur kerja dari sistem RAG dapat terbagi menjadi beberapa tahap yang masing-masing akan dijelaskan pada bagian berikutnya.

3.1.1 *Preprocessing*

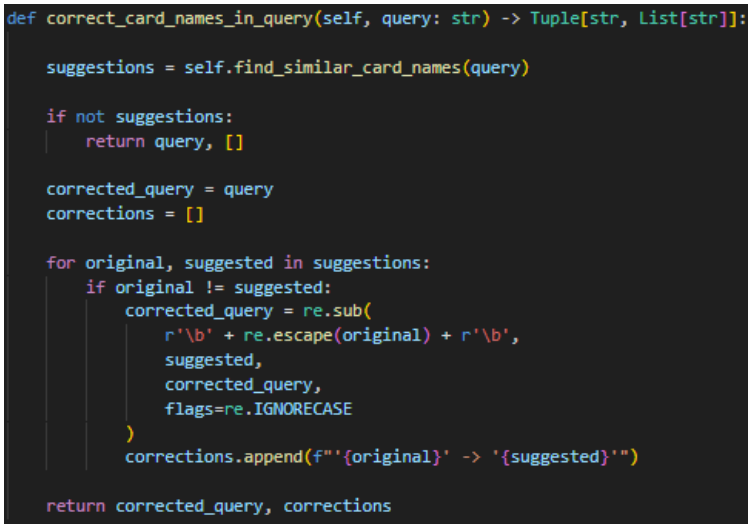
Tahap *preprocessing* digunakan untuk membersihkan teks dan juga menentukan maksud dari permintaan pengguna. Modul *QueryPreprocessor* pertama-tama akan melakukan *intent classification* (deteksi maksud) untuk membedakan antara pertanyaan umum Clash Royale dan permintaan analisis *deck*. Proses ini dilakukan oleh fungsi *is_deck_analysis_query* yang mencari kata kunci spesifik, seperti "analyze", "check", "validate", "deck", dan "rate", dalam masukan pengguna.

```
def is_deck_analysis_query(self, query: str) -> bool:

    deck_keywords = ['analyze', 'check', 'validate', 'deck', 'my deck', 'rate']
    query_lower = query.lower()
    return any(keyword in query_lower for keyword in deck_keywords)
```

Gambar III.1 Potongan kode untuk klasifikasi *intent* dari *query*

Selain klasifikasi *intent*, sistem juga melakukan normalisasi dan koreksi typo. Ketika pengguna menggunakan nama kartu yang tidak baku, singkatan, atau mengandung kesalahan penulisan, fungsi *correct_card_names_in_query* menggunakan *fuzzy matching* melalui pustaka *difflib* untuk membandingkan token dalam pertanyaan dengan daftar entitas kartu yang valid dalam *knowledge graph*. Jika terdapat kemiripan di atas ambang batas tertentu, sistem secara otomatis mengganti istilah tersebut dengan nama resmi agar Translator dapat menghasilkan *query* yang valid terhadap basis data. Sebagai contoh, ketika pengguna mengetik "Royal Recruit", sistem akan mengenal bahwa maksud pengguna adalah "Royal Recruits" dan memperbaiki *query* ke basis data.



```python
def correct_card_names_in_query(self, query: str) -> Tuple[str, List[str]]:

    suggestions = self.find_similar_card_names(query)

    if not suggestions:
        return query, []

    corrected_query = query
    corrections = []

    for original, suggested in suggestions:
        if original != suggested:
            corrected_query = re.sub(
                r'\b' + re.escape(original) + r'\b',
                suggested,
                corrected_query,
                flags=re.IGNORECASE
            )
            corrections.append(f"'{original}' -> '{suggested}'")

    return corrected_query, corrections
```

Gambar III.2 Potongan kode untuk normalisasi nama kartu dalam *query*

Untuk analisis *deck*, digunakan langkah *preprocessing* berupa ekstraksi dan resolusi alias agar sistem mampu mengisolasi daftar 8 kartu dari kalimat pengguna yang formatnya dapat beragam. Sistem dilengkapi dengan kamus alias untuk mengenali singkatan populer, seperti “MK” untuk Mega Knight dan “Ewiz” untuk Electro Wizard. Semua ini dilakukan untuk memastikan bahwa *deck analyzer* menerima masukan yang bersih.

3.1.2 Penerjemahan Bahasa Alami ke Cypher

Modul QueryTranslator digunakan untuk mengonversi pertanyaan pengguna menjadi sintaks Cypher agar dapat dieksekusi oleh Neo4j. Pendekatan yang dilakukan untuk penerjemahan adalah *schema-aware few-shot prompting*. Pertama, dibuat instruksi yang berisi deskripsi lengkap graf, seperti daftar *node* dan *relationship* yang tersedia. Beberapa contoh pasangan pertanyaan dan Cypher juga disediakan agar model mengetahui cara menangani berbagai kasus.

```
    def _build_prompt_template(self) -> PromptTemplate:


        raw_schema_desc = self.schema.get_schema_description()
        schema_desc = raw_schema_desc.replace("{", "{{").replace("}", "}}")

        examples = self.schema.get_cypher_examples()

        raw_examples_text = "\n\n".join([
            f"Question: {ex['question']}\nCypher: {ex['cypher']}"
            for ex in examples
        ])
        examples_text = raw_examples_text.replace("{", "{{").replace("}", "}}")

        template = """You are an expert Cypher query translator for a Clash Royale Knowledge Graph.

""" + schema_desc + """

1. Use MATCH for querying nodes and relationships
2. Use WHERE for filtering (prefer WHERE over inline {{}} for clarity)
3. Use RETURN to specify what data to return
4. For name matching, use "c.name = 'ExactName'" or "c.name CONTAINS 'PartialName'" for partial matches
5. Always return meaningful column names with AS keyword
6. For aggregations, use COUNT(), SUM(), AVG(), etc.
7. For multi-hop queries, chain multiple MATCH patterns
8. Order results when relevant using ORDER BY
9. Limit results if asking for "top" or "best" using LIMIT
10. **For Champion cards**: Always include c.level11_stats in RETURN clause to capture ability information
11. **For Champion queries**: Include c.rarity to identify if card is a champion


""" + examples_text + """

## Your Task:
Translate the following question into a valid Cypher query.
Output ONLY the Cypher query, no explanations or markdown formatting.

Question: {question}

Cypher:"""

        return PromptTemplate.from_template(template)
```

Gambar III.3 Fungsi untuk membuat *prompt* translasi ke Cypher yang berisi instruksi dan *few shot examples*

Setelah *prompt* diikirim dan LLM menghasilkan respons, *output* LLM dibersihkan dari hal seperti tulisan *markdown* atau tulisan seperti “Here is the query you asked for:”. Fungsi *_clean_cypher_output* digunakan untuk menghilangkan semua hal yang bukan kode.

```python
def _clean_cypher_output(self, raw_output: str) -> str:

    if hasattr(raw_output, 'content'):
        text = raw_output.content
    else:
        text = str(raw_output)

    text = text.replace("```cypher", "").replace("```", "").strip()

    if "Cypher:" in text:
        text = text.split("Cypher:")[-1].strip()

    lines = text.split("\n")
    cypher_lines = []
    for line in lines:
        line = line.strip()
        if line.startswith("//") or line.startswith("#"):
            continue
        if line:
            cypher_lines.append(line)

    return " ".join(cypher_lines)
```

Gambar III.4 Fungsi untuk membersihkan *output* Cypher dari LLM

3.1.3 *Information Retrieval*

Modul *KGRetriever* digunakan untuk melakukan eksekusi *query* Cypher terhadap basis data Neo4j. Sistem akan membuka sesi transaksi dengan *driver* Neo4j, menjalankan *query*, dan mengonversi hasil *record* graf menjadi JSON yang dapat diproses oleh komponen selanjutnya. Modul juga mencatat metrik kinerja, seperti waktu eksekusi.

```python
def retrieve(self, cypher_query: str) -> QueryResult:

    start_time = time.time()

    try:
        with self.driver.session() as session:
            result = session.run(cypher_query)
            data = [record.data() for record in result]
            execution_time = time.time() - start_time

            return QueryResult(
                data=data,
                cypher_query=cypher_query,
                execution_time=execution_time,
                error=None
            )
```

Gambar III.5 Fungsi *retrieve* ke *knowledge graph*

3.1.4 Analisis Deck

Modul *DeckAnalyzer* digunakan untuk mengevaluasi komposisi *deck* yang diberikan pengguna. Setelah 8 kartu diekstrak dari *prompt* pengguna, sistem mengambil data statistik setiap kartu dari *knowledge graph*. Data tersebut digunakan untuk mengklasifikasi *deck* ke *archetype* tertentu. Sebagai contoh, jika terdapat X-Bow atau Mortar pada *deck*, *deck* akan diklasifikasi ke kategori Siege. Jika *deck* berisi *tank* besar seperti Golem, klasifikasinya adalah *beatdown*.

```python
def classify_archetype(self, deck: List[str], deck_data: List[Dict]) -> str:

    if any(self.is_siege_building(card) for card in deck):
        return 'siege'

    avg_elixir = self.calculate_avg_elixir(deck_data)
    if 0 < avg_elixir <= 3.0:
        return 'cycle'

    if any(self.is_heavy_tank(card_data) for card_data in deck_data):
        return 'beatdown'

    bridge_spam_cards = ['Magic Archer', 'Ram Rider', 'Lumberjack', 'Royal Ghost',
                         'Bandit', 'Prince', 'P.E.K.K.A.', 'Battle Ram']
    bridge_spam_count = sum(1 for card in deck if card in bridge_spam_cards)
    if bridge_spam_count >= 3:
        return 'bridge_spam'

    return 'No Archetype'
```

Gambar III.6 Fungsi klasifikasi *archetype*

Setelah diklasifikasi, sistem memeriksa komposisi *deck* terhadap aturan yang telah didefinisikan. Aturan ini akan mengecek komposisi seperti keberadaan *win condition* atau keberadaan *spell*. Sistem akan memberikan *warning* dengan tingkat urgensi berupa *strong* atau *weak*.

```python
def _check_general_warnings(self, deck: List[str], deck_data: List[Dict]) -> List[DeckWarning]:

    warnings = []


    if not any(self.is_wincon_card(deck[i], deck_data[i]) for i in range(len(deck))):
        warnings.append(DeckWarning(
            WarningLevel.STRONG,
            'No win condition - Deck has no clear path to tower damage.',
            'general'
        ))


    if not any(self.can_hit_air(card_data) for card_data in deck_data):
        warnings.append(DeckWarning(
            WarningLevel.STRONG,
            'No air defense - Vulnerable to air-heavy decks (Lava Hound, Balloon, etc.).',
            'general'
        ))
```

Gambar III.7 Potongan kode untuk mengecek komposisi *deck*

3.1.5 Generasi Jawaban

Pada tahap terakhir, jawaban akhir akan dibuat oleh modul *AnswerGenerator*. Modul ini akan menjelaskan JSON yang diterima dari *Retriever* dengan bahasa alami yang mudah dipahami pengguna. Pertama, data dari Neo4j disisipkan dalam *template* instruksi yang melarang LLM untuk berhalusinasi. Selain menghasilkan teks, sistem memiliki fungsionalitas *confidence scoring* untuk menghitung skor kepercayaan secara heuristik. Skor ini bukan dari LLM, tetapi dari jumlah data pendukung yang berhasil diambil dari graf. Semakin banyak *node* relevan yang ditemukan, semakin tinggi skor kepercayaan yang diberikan kepada pengguna. Sistem juga akan melakukan *source attribution* untuk memberikan transparansi jawaban.

```
## Instructions:
1. Use ONLY the information provided in the Graph Data below
2. **IMPORTANT**: Include ALL cards from the data - do not skip or omit any entries
3. **CRITICAL**: Only say "I don't have information about that" if NO data was retrieved at all (data is completely empty). If ANY data exists, provide a complete answer using that data.
4. **For basic card info questions**: If the data contains card properties (name, elixir, hp, damage, dps, description, etc.), provide a complete answer with ALL those details. Don't just list the name.
5. **For synergy/counter questions**:
   - If the data has 'synergy_type' or 'strength' fields, use those to explain WHY cards work well together
   - If these fields are missing, simply list the cards WITHOUT making up reasons (don't invent explanations like "complements X's slow movement")
6. **For counter relationships**: If effectiveness/reason fields exist, explain WHY. Otherwise, just list the counters.
7. Format card stats clearly (HP, Damage, DPS, Elixir cost, etc.)
8. Be concise but complete - mention EVERY card and ALL stats in the retrieved data
9. Cite specific numbers and facts from the data
10. For Champions: If a card has rarity='champion', check the 'stats' or 'level11_stats' field for ability information
   - Look for stats with pattern "stat_name (with Ability Name)": value
   - Extract and explain the ability name and its effect on stats

## FORMATTING RULES:
- Do NOT use markdown formatting (no **, ##, -, etc.)
- Use plain text only
- When listing multiple cards:
  * If synergy_type/strength/effectiveness fields exist: "Cards that work well with Giant: Dark Prince (tank synergy, high strength), Prince (beatdown synergy, medium strength)"
  * If NO synergy fields exist: Simply list "Cards that work well with Giant: Dark Prince, Prince, Mega Minion, Musketeer" WITHOUT parenthetical explanations
- Include ALL cards from the data, separated by commas
- Separate into readable sentences

## User Question:
{question}

## Graph Data Retrieved:
{data}
```

Gambar III.8 Instruksi yang diberikan kepada AnswerGenerator

```python
def _extract_sources(self, data: List[Dict[str, Any]]) -> List[str]:

    sources = set()

    for record in data:
        for key, value in record.items():
            if isinstance(value, str) and key.lower() in ['card', 'name', 'from', 'to']:
                sources.add(value)
            elif key == 'card' and isinstance(value, dict) and 'name' in value:
                sources.add(value['name'])

    return sorted(list(sources))
```

Gambar III.9 Fungsi ekstraksi sumber

```python
def _estimate_confidence(self, data: List[Dict[str, Any]], answer: str)

    if not data:
        return 0.0


    num_results = len(data)
    if num_results >= 5:
        base_confidence = 0.9
    elif num_results >= 3:
        base_confidence = 0.8
    elif num_results >= 1:
        base_confidence = 0.7
    else:
        base_confidence = 0.5


    if len(answer) < 20:
        base_confidence *= 0.8
    elif len(answer) > 100:
        base_confidence *= 1.1


    return min(base_confidence, 1.0)
```

Gambar III.10 Fungsi penghitungan *confidence*

3.2 Pemanfaatan Graf Pengetahuan

Selain untuk menyimpan informasi, graf pengetahuan memiliki penggunaan spesifik lain. Sistem memanfaatkan *knowledge graph* sebagai referensi untuk memvalidasi *input* pengguna. Sistem mengambil semua daftar kartu yang valid, menyimpannya dalam *cache*, dan menggunakannya untuk melakukan *fuzzy matching* agar tidak ada *query* yang salah eja.

Selain itu, sistem memanfaatkan sifat graf yang terhubung untuk memberikan jawaban yang lebih komprehensif. Fitur *retrieve_with_context* melakukan penelusuran tetangga relevan dari sebuah kartu. Sebagai contoh, jika pengguna bertanya tentang satu kartu, sistem juga akan mengambil data *counters*, *countered_by*, *synergies*, dan *archetype*.

```python
def get_all_card_names(self) -> List[str]:

    if self._card_names_cache is None:
        result = self.retriever.retrieve("MATCH (c:Card) RETURN c.name AS name")
        if not result.error and result.data:
            self._card_names_cache = [record['name'] for record in result.data]
        else:
            self._card_names_cache = []
    return self._card_names_cache
```

Gambar III.11 Fungsi untuk mengambil semua nama kartu

```python
def retrieve_with_context(self, cypher_query: str, card_name: Optional[str] = None) -> QueryResult:

    main_result = self.retrieve(cypher_query)

    if main_result.error or not card_name:
        return main_result

    try:
        context_data = self._fetch_card_context(card_name)
        if context_data and main_result.data:
            main_result.data[0]["_context"] = context_data
    except Exception:
        pass

    return main_result
```

Gambar III.12 Fungsi untuk mengambil semua nama kartu

Bab 4

Pembahasan

3.1 Hasil Sistem RAG

Ini Penjelasan

3.2 Perbandingan Sistem RAG dengan Sistem Berbasis Pengetahuan

Ini Penjelasan

Bab 5

Kesimpulan

Kesimpulannya ini bos

Referensi

Clash Royale Fandom. (t.t.). Clash Royale Wiki. Dalam *Fandom*. https://clashroyale.fandom.com

Haryadi, A. (2025). Graph RAG Starter Kit. Github. https://github.com/AbdiHaryadi/graph-rag-starter-kit/

Joppi. (2025). *Every Clash Royale Deck Type Explained* [Video]. YouTube. http://www.youtube.com/watch?v=pY6WANsKcUw

OOfro. (2022). *5 BIGGEST MISTAKES NOOBS MAKE IN CLASH ROYALE* [Video]. YouTube. https://www.youtube.com/watch?v=_8SlWCawQxI

Puppe, F. (1993). *Systematic Introduction to Expert Systems: Knowledge Representations and Problem-Solving Methods*. Berlin: Springer-Verlag.

Ryley. (2025) *TOP 10 Decks to Push Trophies in Clash Royale* [Video]. YouTube. https://www.youtube.com/watch?v=y1gcwirV6xg

Vlakx. (2025). *How to Build Your OWN Clash Royale Deck in 2025* [Video]. YouTube. http://www.youtube.com/watch?v=OLLPEpzAUTA

Lampiran

Matriks Kontribusi

Nama	NIM	Bagian
Imanuel Sebastian Girsang	13522058	
Muhammad Neo Cicero Koda	13522108	
Derwin Rustanly	13522115	