BAB 2
LANDASAN TEORI

2.1 Mobile Legends

Mobile Legends: Bang Bang (MLBB) adalah sebuah permainan *multiplayer online battle arena* (MOBA) yang dikembangkan oleh Moonton, dirilis pada tahun 2016 untuk platform *mobile*, *Android* dan *iOS* [2]. *Mobile Legends* menjadi salah satu *game* yang paling populer di Asia Tenggara, terutama di negara seperti Indonesia, Malaysia, dan Filipina, dengan jutaan pemain aktif setiap harinya.

Mobile Legends memiliki mekanika permainan yang mirip dengan *game* MOBA lainnya seperti *League of Legends* dan *Dota 2*, tetapi dikemas untuk perangkat *mobile* dengan waktu permainan yang lebih singkat dan kontrol yang lebih sederhana.

2.1.1 Elemen Utama dalam Mobile Legends

Mode permainan utama dalam *Mobile Legends* adalah pertandingan 5v5, di mana dua tim, masing-masing terdiri dari lima pemain, bertarung untuk menghancurkan markas utama musuh (*turret*). Peta yang digunakan dalam *game* ini terbagi menjadi tiga jalur utama atau biasa disebut *lane*: *Top*, *Mid*, dan *Bottom*. Setiap jalur dilengkapi dengan *turret* dan markas yang harus dihancurkan oleh tim lawan.

Hero adalah karakter yang dapat dimainkan oleh pemain, dan masing-masing *hero* memiliki peran spesifik dalam tim, seperti *Tank*, *Marksman*, *Mage*, *Assassin*, *Fighter*, dan *Support*. Pemain memilih *hero* dengan kemampuan yang sesuai dengan gaya permainan dan peran tim. Strategi dan kerja sama tim sangat penting dalam *Mobile Legends*. Setiap pemain harus memahami peran mereka di tim dan berkolaborasi untuk memenangkan pertandingan. Komunikasi yang baik dan penguasaan mekanik *hero* sangat diperlukan untuk mencapai kemenangan.

Salah satu fitur penting dalam mode kompetitif *Mobile Legends* adalah *Draft-Pick*. Pada mode ini, kedua tim akan secara bergiliran memilih *hero* yang akan dimainkan selama pertandingan. Terdapat juga fase *ban*, di mana setiap tim dapat melarang beberapa *hero* untuk digunakan oleh kedua belah pihak. Fitur ini memungkinkan adanya elemen strategi yang lebih mendalam, di mana tim dapat merencanakan komposisi *hero* yang sinergis sekaligus membatasi pilihan lawan.

Mobile Legends telah menjadi *game* kompetitif yang populer di dunia *esports*. Turnamen besar seperti *Mobile Legends Professional League* (MPL) dan *Mobile Legends World Championship* (M1, M2, M3, dll.) diadakan setiap tahunnya dengan hadiah uang tunai yang besar dan diikuti oleh tim-tim terbaik dari berbagai negara. Pertumbuhan ekosistem *esports Mobile Legends* memberikan peluang bagi pemain profesional untuk berkarier dalam industri *game*.

2.1.2 Aspek Teknologi dalam Mobile Legends

Mobile Legends dibangun menggunakan mesin *game* yang memungkinkan grafis tinggi dan *gameplay* yang halus pada perangkat *mobile*. *Game* ini dirancang untuk dapat berjalan di berbagai jenis perangkat dengan spesifikasi yang berbeda-beda, dari yang rendah hingga tinggi. Salah satu aspek teknologi yang penting dalam *Mobile Legends* adalah sistem *matchmaking*, yang mencocokkan pemain berdasarkan tingkat keahlian mereka. Algoritma ini memastikan bahwa pemain mendapatkan lawan yang seimbang, sehingga permainan menjadi adil dan menyenangkan.

Moonton secara rutin merilis *patch* atau pembaruan, yang mencakup peningkatan fitur, perbaikan *bug*, penyesuaian *hero*, serta konten baru seperti *hero*, *skin*, dan mode permainan. *Mobile Legends* menggunakan *bot* atau karakter yang dikendalikan oleh kecerdasan buatan dalam beberapa mode permainan, seperti ketika ada pemain yang terputus dari permainan atau dalam mode latihan. *Bot* ini membantu menjaga alur permainan agar tetap berjalan meski ada gangguan.

Moonton juga menggunakan teknologi *big data* dan *AI* untuk menganalisis perilaku pemain, pola permainan, dan keseimbangan *game*. Data ini digunakan untuk meningkatkan kualitas permainan dan membuat pembaruan yang lebih akurat dan tepat sasaran.

2.2 Django

Django adalah salah satu *framework* berbasis *Python* yang digunakan untuk pengembangan aplikasi *web*. *Django* pertama kali dikembangkan oleh Adrian Holovaty dan Simon Willison pada tahun 2005 sebagai bagian dari proyek internal di sebuah perusahaan media. *Framework* ini diciptakan untuk membantu pengembang membangun aplikasi *web* dengan cepat dan efisien, sekaligus mengikuti prinsip "*Don't Repeat Yourself*" (DRY) untuk mengurangi

duplikasi kode.

Django dikenal sebagai *framework* yang bersifat *full-stack* karena menyediakan berbagai fitur bawaan seperti sistem autentikasi, *ORM* (*Object-Relational Mapping*), manajemen admin, serta sistem *routing* yang terstruktur. Dengan berbagai fitur ini, pengembang dapat fokus pada logika bisnis tanpa perlu membangun berbagai komponen dasar dari awal. *Django* juga dirancang dengan prinsip keamanan yang kuat, seperti perlindungan terhadap serangan *SQL injection*, *cross-site scripting* (XSS), dan *cross-site request forgery* (CSRF).

2.2.1 Cara Kerja Django

Django bekerja dengan menyediakan kerangka lengkap untuk membangun aplikasi *web*. *Django* menggunakan pola desain *Model-View-Template* (MVT), di mana setiap bagian memiliki peran spesifik dalam pengelolaan data, tampilan, dan logika bisnis aplikasi.

1. Model

   Model merupakan bagian dari *Django* yang bertugas mengelola struktur data dan interaksi dengan *database* menggunakan *Django ORM*. Dengan adanya *model*, pengembang dapat dengan mudah mendefinisikan tabel *database* menggunakan kode *Python* tanpa harus menulis perintah *SQL* secara manual.

2. View

   View bertanggung jawab untuk memproses permintaan pengguna dan mengembalikan respons yang sesuai. *View* dalam *Django* bisa berupa fungsi atau kelas yang mengolah data dari *model* dan menampilkannya melalui *template*.

3. Template

   Template adalah sistem yang digunakan *Django* untuk mengatur tampilan aplikasi. *Template* ini memungkinkan pemisahan antara logika pemrograman dan tampilan, sehingga kode lebih rapi dan mudah dikelola.

Django menggunakan sistem *routing* berbasis *URLconf*, yang memungkinkan pengembang mendefinisikan pola *URL* yang akan diarahkan ke fungsi tampilan tertentu. *Django* menangani permintaan HTTP dengan

mencocokkan *URL* yang diterima dengan pola yang telah ditentukan di dalam konfigurasi *URL*. Jika ditemukan kecocokan, *Django* akan menjalankan fungsi tampilan yang sesuai dan mengembalikan respons dalam bentuk *HTML*, *JSON*, atau format lain yang dibutuhkan.

Django menyediakan dekorator `@urlpatterns` untuk mendefinisikan *URL routing*, sehingga setiap permintaan HTTP dapat diarahkan ke fungsi *view* yang relevan. Fungsi *view* ini kemudian bertugas untuk mengambil data yang diperlukan, memprosesnya, dan mengembalikan respons ke pengguna sesuai kebutuhan aplikasi.

Selain itu, *Django* juga mendukung *middleware* yang memungkinkan eksekusi kode sebelum atau sesudah permintaan diproses oleh *view*. *Middleware* dapat digunakan untuk berbagai keperluan, seperti autentikasi pengguna, *logging*, dan pengelolaan sesi. Dengan adanya *middleware*, pengembang dapat menambahkan fitur tambahan tanpa perlu mengubah kode dalam *view* secara langsung.

2.2.2 Komponen Penting dalam Django

Django memiliki beberapa komponen penting yang sering digunakan dalam pengembangan aplikasi, yaitu:

1. WSGI dan ASGI Django

   Mendukung *WSGI* untuk aplikasi berbasis sinkron dan *ASGI* untuk menangani komunikasi asinkron, seperti *WebSockets* dan *HTTP/2*.

2. Django Template Engine

   Sistem *template* yang digunakan *Django* untuk memisahkan logika aplikasi dari tampilan dengan menyisipkan variabel dan kontrol alur ke dalam *HTML*.

3. Django Middleware

   Komponen ini yang memungkinkan pemrosesan permintaan dan respons sebelum mencapai *view function*, seperti autentikasi dan pengelolaan sesi.

4. Django ORM

Django dilengkapi dengan *Object-Relational Mapping* (ORM) yang memudahkan interaksi dengan *database* tanpa harus menulis *query SQL* secara langsung.

5. Django Admin

   Antarmuka *Django Admin* ini sistem bawaan yang memungkinkan pengelolaan data dan *model* aplikasi dengan mudah tanpa perlu membangun sistem admin dari awal.

6. Django Forms

   Menyediakan sistem manajemen formulir yang memungkinkan validasi *input* pengguna dan pengelolaan data formulir secara otomatis.

7. Django Authentication System

   Django memiliki sistem autentikasi bawaan yang mencakup fitur *login*, *logout*, pembuatan pengguna, serta manajemen izin dan grup.

8. Django Caching

   Mendukung *caching* untuk meningkatkan performa aplikasi dengan menyimpan data yang sering diakses.

9. Django Rest Framework (DRF)

   Ekstensi populer yang memungkinkan pengembangan *API* berbasis *REST* dengan mudah.

2.3 Machine Learning

Machine Learning (ML) adalah cabang dari kecerdasan buatan (*Artificial Intelligence*) yang berfokus pada pengembangan algoritma yang memungkinkan komputer untuk ”belajar” dari data dan membuat prediksi atau keputusan tanpa harus diprogram secara eksplisit [11]. Dalam konteks sistem rekomendasi, *Machine Learning* memainkan peran penting dalam menganalisis pola data dan memberikan rekomendasi yang relevan kepada pengguna.

Content-Based Filtering (CBF) yang digunakan dalam penelitian ini merupakan salah satu pendekatan *Machine Learning* untuk sistem rekomendasi yang bekerja dengan menganalisis karakteristik item untuk memberikan rekomendasi berdasarkan kesamaan fitur. CBF termasuk dalam kategori

unsupervised learning atau lebih tepatnya pendekatan berbasis kesamaan (*similarity-based*), di mana sistem tidak memerlukan data berlabel untuk melatih model, melainkan menggunakan karakteristik intrinsik dari item itu sendiri.

CBF menggunakan pendekatan yang lebih sederhana dan interpretable. CBF dipilih dalam penelitian ini karena beberapa alasan:

1. Tidak memerlukan dataset yang sangat besar seperti *Deep Learning*,
2. Lebih mudah diinterpretasikan karena didasarkan pada perhitungan kesamaan fitur yang eksplisit,
3. lebih efisien dalam hal komputasi karena tidak memerlukan pelatihan model yang kompleks, dan cocok untuk data biner seperti fitur *hero* yang digunakan dalam sistem ini.

2.4 Algoritma Content-Based Filtering (CBF)

Content-Based Filtering (CBF) adalah pendekatan rekomendasi yang menggunakan karakteristik (*content*) dari suatu item untuk memberikan rekomendasi berdasarkan kecocokan dengan item yang menjadi acuan. Dalam konteks permainan *Mobile Legends*, CBF digunakan untuk merekomendasikan *hero counter* berdasarkan karakteristik kemampuan dan kelemahan *hero*—yaitu vektor fitur biner *has_** (kemampuan) dan *weak_to_** (kelemahan).

2.4.1 Cara Kerja CBF

CBF dalam sistem rekomendasi *counter-pick hero* bekerja dengan mencocokkan Vektor Kelemahan (V_{weak}) hero musuh dengan Vektor Kemampuan (V_{has}) hero kandidat *counter*. Tingkat kecocokan diukur menggunakan skor kesamaan $S(A,B)$ yang dihitung melalui operasi *dot product* antara V_{weak} (hero musuh) dan V_{has} (hero kandidat). Semakin tinggi nilai $S(A,B)$, maka semakin besar tingkat kecocokan hero tersebut sebagai *counter*.

Dot product dipilih karena fitur direpresentasikan sebagai *angka biner* (1 = ya, 0 = tidak) tanpa bobot. Setiap pasangan fitur yang cocok (musuh lemah terhadap fitur i dan kandidat memiliki fitur i) memberi kontribusi 1 ke skor. Skor $S(A,B)$ bernilai 0, 1, 2, 3, 4, 5, atau 6. Hero dengan **skor** ≥ 1 direkomendasikan sebagai *counter*; skor < 1 dianggap kurang mampu direkomendasikan.

Tahapan utama dalam cara kerja CBF adalah sebagai berikut:

1. Ekstraksi Fitur Hero (*Feature Engineering*)

   Setiap *hero* dianalisis berdasarkan mekanik *skill*-nya dan diubah menjadi data biner (1 = Punya, 0 = Tidak Punya). Fitur dibagi menjadi dua kategori:

   (a) Fitur Kemampuan (*has_**): Atribut yang dimiliki *hero*. Contoh: *has_burst*, *has_dps*, *has_hard_cc*, *has_anti_heal*, *has_anti_mobility*, *has_purify*.

   (b) Fitur Kelemahan (*weak_to_**): Atribut yang membuat *hero* rentan. Contoh: *weak_to_burst*, *weak_to_dps*, *weak_to_hard_cc*, *weak_to_anti_heal*, *weak_to_anti_mobility*, *weak_to_purify*.

2. Representasi Hero dalam Vektor Fitur

   Setiap *hero* memiliki Vektor Kelemahan V_{weak} dan (sebagai calon *counter*) Vektor Kemampuan V_{has}. Urutan fitur dalam vektor konsisten: (*burst*, *dps*, *hard_cc*, *anti_heal*, *anti_mobility*, *purify*).

   (a) *Hero* Alucard (Musuh):

   $$V_{weak}(\text{Alucard}) = (0,1,1,1,0,0)$$

   (Lemah terhadap *dps*, *hard_cc*, dan *anti_heal*.)

   (b) *Hero* Baxia (Calon Counter):

   $$V_{has}(\text{Baxia}) = (0,1,1,0,0,0)$$

   (Memiliki *dps* dan *hard_cc*.)

   (c) *Hero* Saber (Calon Counter):

   $$V_{has}(\text{Saber}) = (1,0,1,0,0,0)$$

   (Memiliki *burst* dan *hard_cc*.)

3. Perhitungan Skor Kesamaan $S(A,B)$ menggunakan Vweak (Dot Product)

   Skor kesamaan antara hero musuh A dan calon *counter* B dihitung menggunakan metode *dot product* antara vektor kelemahan hero musuh $V_{weak}(A)$ dan vektor atribut yang dimiliki hero kandidat $V_{has}(B)$. Perhitungan

ini bertujuan untuk mengetahui sejauh mana atribut yang dimiliki hero B dapat mengeksploitasi kelemahan hero A.

$$S(A,B) = V_{weak}(A) \cdot V_{has}(B) \tag{2.1}$$

Secara matematis, perhitungan dot product tersebut dapat dituliskan sebagai berikut:

$$S(A,B) = \sum_{i=1}^{6} V_{weak}^{(A)}[i] \times V_{has}^{(B)}[i] \tag{2.2}$$

Karena setiap elemen pada vektor V_{weak} dan V_{has} bernilai biner (0 atau 1), maka nilai $S(A,B)$ berupa bilangan bulat dengan rentang 0 hingga 6. Semakin tinggi nilai $S(A,B)$, semakin besar tingkat kecocokan hero B sebagai *counter* bagi hero A.

Pada sistem ini, hero dengan nilai $S(A,B) \geq 1$ akan direkomendasikan sebagai hero *counter*, sedangkan hero dengan nilai $S(A,B) < 1$ tidak direkomendasikan.

4. Pemberian Rekomendasi

   Sistem menyaring *hero* berdasarkan *role* yang diminta pengguna. Untuk setiap calon *counter* di *role* tersebut dihitung $S(A,B)$. Hanya hero dengan skor ≥ 1 yang masuk rekomendasi. Hasil diurutkan berdasarkan skor menurun, dan lima *hero* teratas (top 5) beserta *justifikasi teks* ditampilkan kepada pengguna.

2.4.2 Komponen Penting dalam CBF

Komponen utama CBF dalam sistem ini:

1. Ekstraksi Fitur Hero (*Feature Engineering*)

   Fondasi sistem: fitur *hero* diekstraksi dari mekanik *skill* dan dinormalisasi menjadi data biner.

   (a) Vektor Kemampuan V_{has}: mendefinisikan apa yang bisa dilakukan *hero*.

(b) Vektor Kelemahan V_{weak}: mendefinisikan kelemahan *hero* terhadap fitur tertentu.

2. Normalisasi Data (*Feature Binarization*)

   Data kualitatif (deskripsi *skill*) dinormalisasi menjadi numerik biner (0 atau 1). Proses ini mengubah deskripsi tidak terstruktur menjadi vektor fitur yang dapat dihitung oleh sistem.

3. Metode Similarity: Skor $S(A,B)$ Vweak (Dot Product)

   Dot product antara V_{weak}(musuh) dan V_{has}(kandidat) digunakan untuk mengukur kecocokan. Semakin tinggi skor (maksimal 6), semakin cocok sebagai *counter*. Tidak ada bobot per fitur; setiap fitur berkontribusi sama (1 jika cocok, 0 jika tidak).

4. Model Rekomendasi

   Sistem menyaring *hero* berdasarkan *role* yang diminta. Kemudian dihitung $S(A,B)$ untuk setiap kandidat; hanya skor ≥ 1 yang direkomendasikan. Hasil diurutkan berdasarkan skor menurun, dan lima *hero* teratas ditampilkan beserta justifikasi teks.