Optimasi Penargetan Prediktif Menggunakan Pattern Matching Boyer-Moore untuk Robocode

Ahmad Ibrahim - 13523089
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13523089@std.stei.itb.ac.id

***Abstract*—Robocode menyediakan sebuah lingkungan kompetitif yang ideal karena keberhasilan sangat bergantung pada kemampuan manuver dan akurasi penargetan. Sistem penargetan konvensional seperti linear atau sirkular seringkali tidak efektif melawan lawan yang mampu bergerak secara dinamis dan tak terduga. Paper ini mengusulkan sebuah sistem penargetan prediktif adaptif pada bot bernama "meong" untuk mengatasi tantangan tersebut. Metode yang diimplementasikan adalah pencocokan pola (*pattern matching*) multi-panjang untuk mengidentifikasi sekuens pergerakan lawan yang berulang dari data historis. Untuk memastikan performa *realtime*, pencarian pola ini dioptimalkan menggunakan algoritma Boyer-Moore. Keunikan dari sistem ini terletak pada kemampuannya untuk menyimpan dan memuat kembali basis data pola pergerakan antar pertandingan (memori persisten), memungkinkan bot untuk "belajar" dan menjadi lebih efektif melawan lawan yang sama di masa depan. Hasil implementasi menunjukkan bahwa pendekatan ini mampu meningkatkan akurasi tembakan secara signifikan dengan memprediksi posisi lawan berdasarkan perilaku historis mereka, bukan hanya berdasarkan status sesaat.**

Kata Kunci: Robocode, Penargetan Prediktif, Pencocokan Pola, Algoritma Boyer-Moore.

I. PENDAHULUAN

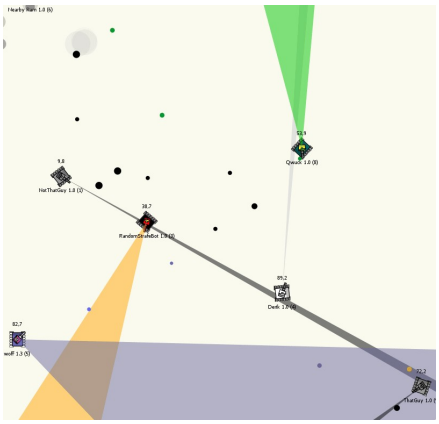
Robocode adalah sebuah permainan dan lingkungan pengembangan yang dirancang oleh IBM, di mana pemain mengkodekan robot tank otonom (*bot*) untuk bertarung di dalam sebuah arena. Platform ini telah lama menjadi sarana populer untuk eksplorasi konsep-konsep kecerdasan buatan (AI), seperti logika fuzzy, jaringan saraf, dan algoritma genetika. Dalam lingkungan yang dinamis ini, performa sebuah bot secara fundamental ditentukan oleh dua komponen utama yang saling bersinergi: strategi pergerakan untuk bertahan hidup dan sistem penargetan untuk mengeliminasi lawan. Keduanya harus bekerja secara efisien di bawah batasan komputasi waktu-nyata untuk mencapai keunggulan kompetitif.

Tantangan utama dalam merancang sistem penargetan yang efektif di Robocode adalah sifat lawan yang cerdas dan non-deterministik. Bot lawan secara aktif berusaha menghindari tembakan, mengubah arah, kecepatan, dan akselerasi secara tak terduga. Akibatnya, metode penargetan sederhana yang hanya mengekstrapolasi posisi lawan secara linear atau sirkular memiliki tingkat keberhasilan yang rendah. Untuk dapat mengenai target yang bergerak dinamis, sebuah bot memerlukan kemampuan untuk memprediksi posisi lawan di masa depan dengan akurasi tinggi. Prediksi ini tidak dapat hanya didasarkan pada status sesaat (posisi, kecepatan, dan arah saat ini), tetapi harus mampu mengenali pola perilaku yang lebih kompleks dari lawannya.

II. DASAR TEORI

A. *Robocode*

Robocode adalah sebuah lingkungan simulasi pertempuran tank yang bersifat *open-source* dan berbasis Java, diciptakan oleh Mathew Nelson. Platform ini memungkinkan pengembang untuk merancang, mengimplementasikan, dan menguji agen-agen perangkat lunak otonom—dikenal sebagai *bot*—dalam sebuah arena pertempuran virtual. Setiap bot beroperasi berdasarkan serangkaian aturan fisika yang telah ditentukan dan dieksekusi dalam siklus waktu diskrit yang disebut *tick*. Dalam setiap *tick*, bot dapat mengevaluasi lingkungannya dan mengambil tindakan. Aturan fisika ini mencakup kecepatan maksimum (8.0 unit/tick), akselerasi (1.0 unit/tick²), deselerasi (2.0 unit/tick²), laju rotasi bodi dan senjata, serta fisika proyektil di mana kecepatan peluru berbanding terbalik dengan daya tembaknya (kecepatan = 20 - 3 * daya_tembak).



Gambar 1. Permainan robocode

Interaksi dengan lingkungan dan bot lain terjadi melalui model pemrograman berbasis peristiwa (*event-driven programming*). Bot menerima informasi sensorik melalui pemanggilan metode seperti OnScannedBot (ketika mendeteksi lawan), OnHitByBullet (ketika terkena tembakan), atau OnWallHit (ketika menabrak dinding). Berdasarkan peristiwa ini, bot harus merespons secara *real-time* dengan mengeluarkan perintah seperti bergerak (SetAhead), memutar radar (SetTurnRadarRight), atau menembak (SetFire). Keterbatasan waktu komputasi per *tick* dan kompleksitas interaksi fisika menjadikan Robocode sebagai platform yang ideal dan menantang untuk riset dalam bidang kecerdasan buatan, khususnya dalam pengujian strategi adaptif, algoritma prediktif, dan pengambilan keputusan dalam kondisi tidak pasti.

B. Sistem Penargetan (Targeting System)

Sistem penargetan adalah komponen krusial yang menentukan kemampuan ofensif sebuah bot. Tujuannya adalah untuk menembakkan proyektil sedemikian rupa sehingga jalur proyektil dan jalur target bertemu pada titik dan waktu yang sama. Mengingat proyektil membutuhkan waktu untuk mencapai target (waktu = jarak / kecepatan_peluru), sistem harus dapat memprediksi di mana lawan akan berada pada saat proyektil tiba, bukan di mana ia berada saat ini.

Metode-metode awal dan paling sederhana dalam penargetan Robocode didasarkan pada asumsi-asumsi pergerakan yang kaku. Dua yang paling umum adalah:

- Linear Targeting: Metode ini mengasumsikan lawan akan terus bergerak dalam garis lurus dengan kecepatan dan arah yang konstan. Posisi masa depan dihitung dengan melakukan ekstrapolasi linear dari vektor kecepatan lawan saat ini. Secara konseptual, ini seperti membidik di depan target yang sedang berlari. Meskipun implementasinya sederhana, kelemahannya sangat signifikan: metode ini hampir selalu gagal total melawan bot yang menggunakan manuver dasar seperti mengubah arah atau berhenti sejenak.
- Circular Targeting: Sebagai perbaikan dari penargetan linear, metode ini mengasumsikan lawan bergerak dalam lintasan melingkar konstan. Dengan mengukur perubahan arah lawan antar *tick* untuk menghitung kecepatan sudut (*angular velocity*), bot dapat memprediksi posisi masa depan di sepanjang busur lingkaran. Ini lebih efektif melawan bot yang cenderung berputar mengelilingi lawan, tetapi masih mudah dikalahkan. Bot yang menggunakan gerakan *oscillating* (bolak-balik) atau strategi "Stop-and-Go" akan dengan mudah mematahkan asumsi sirkular ini.

Kegagalan metode konvensional ini menunjukkan bahwa asumsi pergerakan yang statis tidak memadai untuk lingkungan yang kompetitif.

Menyadari keterbatasan metode konvensional, penargetan prediktif muncul sebagai pendekatan yang lebih canggih. Alih-alih mengandalkan asumsi pergerakan yang kaku, sistem ini bekerja berdasarkan hipotesis bahwa pergerakan lawan, meskipun terlihat acak, sesungguhnya tidak demikian. Pergerakan tersebut mengandung pola-pola yang dapat dikenali dan dieksploitasi, yang bersumber dari: (1) kebiasaan desain pemrogram, (2) artefak algoritma yang digunakan (misalnya, perilaku algoritma penghindar dinding di dekat sudut), atau (3) respons terkondisi terhadap stimulus tertentu (misalnya, selalu bergerak ke kiri setelah terkena tembakan).

Dengan merekam histori pergerakan lawan secara detail, sistem dapat mencari kemiripan antara situasi saat ini dengan situasi yang pernah terjadi di masa lalu. Jika pola yang cocok ditemukan, sistem memprediksi bahwa lawan akan mengulangi tindakan yang mengikuti pola tersebut di masa lalu. Pendekatan inilah yang menjadi inti dari penelitian ini, mengubah masalah penargetan dari masalah fisika sederhana menjadi masalah pengenalan pola.

C. Pencocokan Pola (Pattern Matching)

Pencocokan pola adalah proses menemukan kemunculan sebuah pola (sekuens data yang lebih pendek) di dalam kumpulan data yang lebih besar yang disebut teks (atau histori). Dalam konteks Robocode, "teks" adalah seluruh riwayat pergerakan lawan yang telah direkam selama pertandingan, dan "pola" adalah sekuens pergerakan terbaru dari lawan tersebut.

Untuk memungkinkan pencocokan, data pergerakan yang bersifat kontinu harus diubah menjadi data diskrit melalui proses yang disebut diskritisasi. Atribut seperti kecepatan sudut (misalnya, 0.015 rad/tick) atau kecepatan (misalnya, 7.89 unit/tick) diubah menjadi nilai-nilai integer dalam rentang terbatas. Sebagai contoh, akselerasi hanya dikategorikan menjadi tiga nilai: -1 (melambat), 0 (konstan), atau 1 (berakselerasi). Proses ini sangat penting karena pencocokan pola pada data kontinu hampir tidak mungkin dilakukan secara efisien. Setelah diskritisasi, setiap momen (tick) dalam pergerakan lawan dapat direpresentasikan sebagai sebuah state tunggal, misalnya: State = {KecepatanSudut: 5, Kecepatan: 8, Akselerasi: 1}.

Dengan demikian, histori pergerakan lawan menjadi sebuah sekuens panjang dari *state-state* ini. Ketika bot ingin menembak, ia mengambil beberapa *state* terakhir (misalnya, 15 *tick* terakhir) sebagai "pola" dan mencarinya di dalam "teks" histori. Jika ditemukan, sekuens *state* yang mengikuti kemunculan historis tersebut menjadi dasar prediksi yang sangat kuat untuk pergerakan lawan di masa depan.

D. Algoritma Boyer-Moore

Untuk mengimplementasikan pencocokan pola secara efisien di bawah batasan waktu Robocode, diperlukan sebuah algoritma pencarian string yang cepat. Algoritma naif yang membandingkan pola karakter per karakter dari awal teks memiliki kompleksitas waktu yang buruk. Algoritma Boyer-Moore, yang dirancang oleh Robert S. Boyer dan J Strother Moore, menawarkan solusi yang jauh lebih optimal. Keunggulan performanya berasal dari dua ide cerdas:

Pencocokan dari Kanan ke Kiri: Berbeda dengan pendekatan intuitif, Boyer-Moore membandingkan pola dengan jendela teks dari karakter terakhir pola, bukan dari yang pertama.

Jika terjadi ketidakcocokan, algoritma ini tidak hanya menggeser jendela satu posisi ke depan. Sebaliknya, ia menggunakan dua heuristik untuk melakukan loncatan besar di dalam teks, melewatkan banyak perbandingan yang tidak perlu:

Ketika ketidakcocokan terjadi pada sebuah karakter di teks, algoritma akan menggeser pola ke depan hingga karakter tersebut sejajar dengan kemunculan terakhirnya di dalam pola. Jika karakter tersebut tidak ada dalam pola sama sekali, pola dapat digeser melewati posisi karakter tersebut sepenuhnya, menghasilkan loncatan yang sangat besar.

Heuristik Sufiks Baik (Good Suffix Heuristic): Jika ketidakcocokan terjadi setelah beberapa karakter di akhir pola sudah cocok (membentuk sebuah sufiks), heuristik ini akan menggeser pola hingga kemunculan berikutnya dari sufiks yang cocok tersebut di dalam pola itu sendiri.

Kombinasi kedua heuristik ini, terutama pada alfabet yang besar (seperti kemungkinan kombinasi state gerakan), membuat Boyer-Moore memiliki performa rata-rata yang sangat cepat, menjadikannya pilihan ideal untuk lingkungan Robocode.

E. Memori Persisten

Konsep memori persisten merujuk pada kemampuan sistem untuk menyimpan data di luar siklus eksekusi program dan memuatnya kembali di sesi berikutnya. Dalam penelitian ini, histori pergerakan setiap lawan di-serialisasi ke dalam sebuah file eksternal berformat JSON pada akhir setiap pertandingan. Format JSON dipilih karena strukturnya yang mudah dibaca manusia (memudahkan debugging) dan kemudahan *parsing* di berbagai platform.

Ketika bot memulai pertandingan baru, ia akan memuat kembali data historis tersebut ke dalam memori. Mekanisme ini secara efektif mengubah bot dari agen yang menderita "amnesia" setiap ronde menjadi agen yang memiliki "memori jangka panjang" dan kemampuan untuk belajar lintas pertandingan (*meta-learning*). Hal ini secara signifikan meningkatkan efektivitasnya, terutama dalam format turnamen multi-ronde. Basis data polanya menjadi semakin kaya dan akurat seiring berjalannya waktu, memungkinkan bot untuk membangun "profil perilaku" untuk setiap lawan yang pernah dihadapinya dan mengeksploitasi kebiasaan mereka dengan lebih presisi.

III. IMPLEMENTASI

Bab ini menguraikan secara rinci arsitektur perangkat lunak dan detail implementasi dari bot "Meong". Pembahasan mencakup struktur umum bot, alur kerja akuisisi data, mekanisme penargetan prediktif, sistem pergerakan, dan modul memori persisten yang menjadi inti dari penelitian ini. Seluruh implementasi dilakukan dalam bahasa C# menggunakan API Robocode Tank Royale, dengan fokus pada modularitas, efisiensi, dan kemampuan adaptasi.

A. Arsitektur Umum Bot

Arsitektur "meong" dirancang secara modular untuk memisahkan fungsi-fungsi utama: penargetan, pergerakan, dan manajemen data. Pemisahan ini merupakan keputusan desain fundamental yang memungkinkan pengembangan, pengujian, dan iterasi yang lebih mudah untuk setiap komponen secara independen. Sebagai contoh, jika bug terdeteksi pada logika penargetan, pengembang dapat mengisolasinya tanpa perlu mempertanyakan integritas modul pergerakan. Fleksibilitas ini juga membuka jalan untuk peningkatan di masa depan, seperti mengganti sistem pergerakan Anti-Gravitasi dengan algoritma lain tanpa menulis ulang seluruh bot.

Struktur umum bot dapat dibagi menjadi empat modul utama yang saling berinteraksi:

- Modul Akuisisi dan Representasi Data: Bertanggung jawab untuk mengumpulkan data mentah dari arena, mengubahnya menjadi format yang terstruktur (State), dan menyimpannya dalam objek EnemyData. Modul ini berfungsi sebagai fondasi bagi modul penargetan.
- Modul Penargetan Prediktif: Merupakan inti dari sistem ofensif bot. Modul ini menggunakan data yang telah diolah untuk melakukan pencocokan pola multi-panjang, mensimulasikan pergerakan masa depan lawan, dan menentukan sudut tembak yang optimal.
- Modul Pergerakan Anti-Gravitasi: Sistem defensif yang bertugas untuk memposisikan bot di lokasi teraman di arena dengan menghindari peluru dan menjaga jarak strategis dari lawan.
- Modul Memori Persisten: Komponen yang memungkinkan bot untuk menyimpan dan memuat data historis antar pertandingan, memberikan kemampuan belajar jangka panjang.

Interaksi antar modul ini sepenuhnya digerakkan oleh peristiwa (*event-driven*). OnScannedBot menjadi pemicu utama ("detak jantung") untuk seluruh alur kerja ofensif, memicu serangkaian tindakan berantai mulai dari akuisisi data hingga eksekusi penargetan. Sementara itu, OnTick secara kontinu memicu modul pergerakan defensif, memastikan bot selalu waspada dan reaktif terhadap perubahan lingkungan setiap saat. Peristiwa OnGameEnded secara khusus memicu modul memori persisten untuk melakukan serialisasi dan menyimpan hasil pembelajaran ronde tersebut.

B. Modul Akuisisi dan Representasi Data

Akurasi sistem prediktif sangat bergantung pada kualitas dan representasi data yang digunakan. Modul ini bertugas untuk mengubah data sensorik kontinu dari game engine menjadi data diskrit yang dapat dianalisis secara komputasional.

1. Pengumpulan Data

   Setiap kali radar bot mendeteksi lawan, *event* OnScannedBot akan terpicu. Di dalam *event handler* inilah data-data vital lawan seperti posisi (X, Y), arah (radian), dan kecepatan (unit/tick) diambil. Namun, data mentah ini kurang informatif untuk prediksi. Status sesaat hanya memberi tahu "di mana lawan sekarang", bukan "apa yang sedang coba dilakukan lawan". Oleh karena itu, dihitunglah atribut-atribut

turunan yang lebih deskriptif terhadap *perubahan* perilaku, yang jauh lebih prediktif.

```
EnemyData data = enemyDataByName[botName];
        data.LastX = e.X;
        data.LastY = e.Y;
        data.IsAlive = true;
        double currentDirection =
toRad(NormalizeRelativeAngle(e.Direction));
        double angularVelocity =
data.HasPrevious ?

(currentDirection - data.LastDirection +
Math.PI) % (2 * Math.PI) - Math.PI : 0;
        data.LastDirection =
currentDirection;
        double currentSpeed = e.Speed;
        double acceleration =
data.HasPrevious ? currentSpeed -
data.LastSpeed : 0;
        data.LastSpeed = currentSpeed;
        data.HasPrevious = true;
```

2. Diskritisasi dan Representasi State

Atribut-atribut yang telah dihitung kemudian didiskritisasi untuk mengubahnya menjadi nilai integer yang dapat dibandingkan. Proses pencocokan pola pada data kontinu secara langsung sangatlah sulit dan rentan terhadap kesalahan presisi floating-point. Diskritisasi menyederhanakan masalah ini dengan mengelompokkan nilai-nilai yang serupa. Proses ini diimplementasikan dalam konstruktor struct State:

- AngularVelocity dikalikan dengan faktor skalar (dalam kode, 512). Pilihan faktor ini adalah sebuah kompromi: nilai yang terlalu kecil akan membuat semua belokan tampak sama, sedangkan nilai yang terlalu besar akan menciptakan terlalu banyak *state* unik (sebuah bentuk dari "kutukan dimensi"), sehingga mempersulit penemuan pola. Nilai 512 terbukti memberikan keseimbangan yang baik antara presisi dan generalisasi. Sebagai contoh, perubahan arah sebesar 0.02 radian akan menjadi nilai diskrit (int)(0.02 * 512) = 10.
- Speed dibulatkan ke nilai integer terdekat. Ini secara efektif mengelompokkan kecepatan yang sedikit berbeda (misalnya, 7.8 dan 7.9 unit/tick) ke dalam kategori yang sama (8), mengasumsikan bahwa fluktuasi kecil tersebut bukan merupakan perubahan manuver yang disengaja.
- Acceleration dikategorikan secara agresif menjadi tiga nilai: -1 (melambat), 0 (konstan), dan 1 (berakselerasi). Penggunaan nilai ambang batas (threshold) membuat kategorisasi ini kuat terhadap *noise* sensorik. Ini menangkap keputusan makro lawan (mempercepat, memperlambat, atau mempertahankan laju) daripada fluktuasi kecepatan minor.

Setiap *tick*, sebuah objek State baru yang merepresentasikan kondisi lawan pada saat itu dibuat dan ditambahkan ke dalam CurrentGameMovementHistory. Daftar ini secara efektif merupakan rekaman jejak digital dari perilaku lawan dalam pertandingan saat ini.

```
public struct State : IEquatable<State>
{
    public int AngularVelocity { get; set; }
    public int Speed { get; set; }
    public int Acceleration { get; set; }

    public State(double angularVelocity,
double speed, double acceleration)
    {
        AngularVelocity =
(int)(angularVelocity * 512);
        Speed = (int)Math.Round(speed);
        double threshold = 0.1;
        if (acceleration < -threshold)
Acceleration = -1;
        else if (acceleration > threshold)
Acceleration = 1;
        else Acceleration = 0;
    }

    public bool Equals(State other) =>
AngularVelocity == other.AngularVelocity &&
Speed == other.Speed && Acceleration ==
other.Acceleration;
    public override bool Equals(object obj)
=> obj is State other && Equals(other);
    public override int GetHashCode() =>
HashCode.Combine(AngularVelocity, Speed,
Acceleration);
}
```

C. Modul Penargetan Prediktif

Modul ini adalah inti dari kecerdasan bot. Alur kerjanya mengeksekusi serangkaian langkah komputasi canggih setiap kali OnScannedBot terpicu.

1. Pembentukan Pola Multi-Panjang: Untuk meningkatkan fleksibilitas, sistem menerapkan strategi pencarian iteratif dengan pendekatan "terbaik-dahulu" (*best-first*). Pencarian dimulai dengan pola terpanjang yang paling spesifik (misalnya, 30 *tick* terakhir). Pola yang panjang cenderung lebih unik dan prediksinya lebih dapat diandalkan karena ia menangkap konteks manuver yang lebih besar. Jika tidak ditemukan kecocokan, panjang pola dikurangi satu per satu (len--) dan pencarian diulang. Proses ini merupakan bentuk *graceful degradation* (penurunan performa yang anggun): jika prediksi yang sangat spesifik tidak tersedia, sistem akan mundur ke prediksi yang lebih umum (pola lebih pendek)

2. daripada tidak memberikan prediksi sama sekali.
3. Pencarian Pola dengan Boyer-Moore: Untuk setiap panjang pola yang diuji, sekuens State terbaru diambil. Sekuens ini kemudian dicari dalam gabungan data historis lintas pertandingan (HistoricalMovement) dan data pertandingan saat ini. Metode BoyerMoore.Search() digunakan untuk tugas ini. Pilihan algoritma ini sangat penting untuk performa. "Alfabet" dalam kasus ini adalah himpunan semua kemungkinan kombinasi State, yang bisa sangat besar. Algoritma Boyer-Moore, khususnya dengan *Bad Character Heuristic*-nya, sangat efisien pada alfabet besar karena mampu melakukan loncatan besar dalam data teks, secara signifikan mengurangi jumlah perbandingan yang diperlukan dibandingkan pencarian naif (O(n*m)) menjadi sub-linear pada kasus rata-rata.
4. Ekstraksi Sekuens Prediksi: Jika BoyerMoore.Search() berhasil menemukan indeks kecocokan dalam data histori, maka sekuens State yang berada tepat setelah pola yang cocok tersebut diambil. Sekuens ini dianggap sebagai *hipotesis* yang paling mungkin untuk pergerakan lawan selanjutnya. Ini adalah langkah krusial di mana data masa lalu digunakan untuk menginformasikan prediksi masa depan.
5. Simulasi Gerakan Probabilistik (Play It Forward): Berbekal hipotesis sekuens prediksi, bot menjalankan metode yang mirip dengan simulasi Monte Carlo. Sejumlah simulasi (SIMULATION_COUNT) dijalankan untuk memetakan "awan kemungkinan" posisi masa depan lawan. Setiap simulasi memproyeksikan pergerakan lawan *tick* demi *tick*, mengaplikasikan perubahan State dari sekuens prediksi. Jika sekuens prediksi habis sebelum proyektil diperkirakan tiba, simulasi akan terus berjalan dengan asumsi pergerakan terakhir, namun dengan bobot kepercayaan (weight) yang diturunkan secara eksponensial. Ini mencerminkan prinsip bahwa kepastian prediksi menurun seiring dengan bertambahnya horizon waktu.
6. Seleksi Sudut Tembak Berbasis Histogram: Setiap posisi akhir dari simulasi diubah menjadi sudut tembak relatif. Sudut-sudut ini kemudian digunakan untuk memberikan "suara" pada sebuah *array* (angleScores) yang berfungsi sebagai histogram. Bobot suara dari setiap simulasi memengaruhi besarnya kontribusi pada histogram. Setelah semua simulasi selesai, *bin* histogram dengan skor kumulatif tertinggi dipilih. Puncak histogram ini merepresentasikan arah yang paling mungkin di mana lawan akan berada, berdasarkan konsensus dari puluhan simulasi. Metode voting ini membuat keputusan akhir menjadi kuat terhadap simulasi tunggal yang mungkin menghasilkan prediksi menyimpang.

```
List<State> predictionSequence = new
List<State>();

        // Loop from the longest pattern length
down to the minimum
        for (int len = PATTERN_LENGTH; len >=
MIN_PATTERN_LENGTH; len--)
        {
            if
(data.CurrentGameMovementHistory.Count > len)
            {
                // 1. Define the pattern of
current length
                var pattern =
data.CurrentGameMovementHistory.GetRange(data.Cur
rentGameMovementHistory.Count - len,
len).ToArray();

                // 2. Combine historical and
current game data to search in
                var historyToSearch =
data.HistoricalMovement.Concat(data.CurrentGameMo
vementHistory.Take(data.CurrentGameMovementHistor
y.Count - len)).ToArray();

                int matchIndex =
BoyerMoore.Search(historyToSearch, pattern);

                // 3. If a match is found, use it
and stop searching for shorter patterns
                if (matchIndex != -1)
                {
                    int predictionStartIndex =
matchIndex + pattern.Length;
                    int predictionLength =
Math.Min(100, historyToSearch.Length -
predictionStartIndex);
                    if (predictionLength > 0)
                    {
                        predictionSequence =
historyToSearch.Skip(predictionStartIndex).Take(p
redictionLength).ToList();
                    }
                    break;
                }
            }
        }
```

D. Modul Pergerakan

Untuk memastikan modul penargetan memiliki cukup waktu untuk bekerja, kelangsungan hidup bot adalah prioritas utama. "Meong" menggunakan sistem pergerakan Anti-Gravitasi, sebuah pendekatan standar emas di komunitas Robocode. Prinsipnya adalah memodelkan arena sebagai "medan gravitasi" di mana bot adalah partikel yang mencoba mencari titik dengan energi potensial terendah.

Setiap *tick*, bot mengevaluasi sejumlah titik kandidat di sekelilingnya. Setiap titik dievaluasi berdasarkan jumlah "gaya tolak" yang diterimanya dari berbagai sumber bahaya, dengan kekuatan gaya yang telah dikalibrasi:

- Bot Lawan: Menghasilkan gaya tolak yang berbanding terbalik dengan kuadrat jarak ($1/d^2$), mendorong bot untuk menjaga jarak aman—tidak

terlalu dekat untuk diseruduk, tidak terlalu jauh sehingga tembakan tidak efektif.

- Peluru Lawan: Setiap peluru yang terdeteksi menghasilkan "dinding" gaya tolak yang sangat kuat di sepanjang lintasan prediksinya. Kekuatan gaya ini harus menjadi yang dominan untuk memastikan penghindaran peluru menjadi prioritas tertinggi.
- Dinding dan Sudut Arena: Menghasilkan gaya tolak yang kuat untuk mencegah bot terjebak di sudut, yang merupakan posisi yang sangat rentan.

Sinergi antara modul ini dan modul penargetan sangat jelas: pergerakan defensif yang baik tidak hanya memperpanjang hidup bot, tetapi juga dapat memanipulasi lawan. Dengan terus bergerak dan menghindar, "Meong" memaksa lawannya untuk terus-menerus menghitung ulang solusi penargetan mereka, yang berpotensi menyebabkan mereka jatuh ke dalam pola gerakan yang lebih sederhana dan dapat diprediksi, sehingga menjadi target yang lebih mudah bagi sistem pencocokan pola "Meong".

```
    double bestX = X;
            double bestY = Y;
            double minGrav = double.PositiveInfinity;

            for (int i = 0; i < POINT_COUNT; i++)
            {
                double theta = (2 * Math.PI /
POINT_COUNT) * i;

                for (int u = 0; u < ITERATE_RADIUS;
u++) {
                    double r = Math.Sqrt(Math.Pow(u *
DELTA_RADIUS, 2) + Math.Pow(MIN_RADIUS, 2));

                    double x = X + r *
Math.Cos(theta);
                    double y = Y + r *
Math.Sin(theta);

                    if (x < MOVE_WALL_MARGIN || x >
ArenaWidth - MOVE_WALL_MARGIN ||
                        y < MOVE_WALL_MARGIN || y >
ArenaHeight - MOVE_WALL_MARGIN)
                    {
                        continue;
                    }

                    double grav = CalcGrav(x, y);
                    if (grav < minGrav ||
distance(X,Y,destX, destY) < 20)
                    {
                        minGrav = grav;
                        bestX = x;
                        bestY = y;
                    }
                }
            }

            if (minGrav < CalcGrav(destX, destY) *
GRAV_OVERRIDE_TRESHOLD)
            {
                destX = bestX;
                destY = bestY;
            }

            double turn = toRad(BearingTo(destX,
destY));
            SetTurnLeft(toDeg(Math.Tan(turn)));
            SetForward(DistanceTo(destX, destY) *
Math.Cos(turn));
```

E. Modul Memori Persisten

Kemampuan belajar jangka panjang, yang membedakan bot canggih dari bot biasa, diimplementasikan melalui modul ini.

- Penyimpanan (SaveAllPatterns): Ketika sebuah pertandingan berakhir (OnGameEnded), bot melakukan proses serialisasi. Untuk setiap lawan yang dihadapi, seluruh riwayat pergerakan (HistoricalMovement yang dimuat sebelumnya digabung dengan CurrentGameMovementHistory yang baru direkam) diubah menjadi format string JSON. Penggunaan JSON memungkinkan data untuk mudah dibaca dan di-debug oleh manusia, serta bersifat portabel. String ini kemudian disimpan ke dalam sebuah file di direktori patterns/, dengan nama file sesuai dengan nama unik bot lawan.
- Pemuatan (LoadAllPatterns): Pada awal eksekusi (Run), bot memeriksa keberadaan direktori patterns/. Jika ada, ia akan membaca setiap file JSON, melakukan proses deserialisasi untuk mengubah string kembali menjadi struktur data List<State>, dan memuatnya ke dalam atribut HistoricalMovement dari EnemyData yang sesuai. Proses ini secara efektif mengatasi "masalah permulaan dingin" (*cold start problem*). Bot tidak lagi memulai setiap pertempuran dari nol, melainkan membawa seluruh "pengalaman"-nya dari pertempuran sebelumnya. Ini memungkinkannya untuk melakukan *meta-learning*, yaitu belajar tentang gaya pemrograman atau kebiasaan taktis dari seorang pembuat bot, bukan hanya perilaku bot itu sendiri dalam satu ronde.

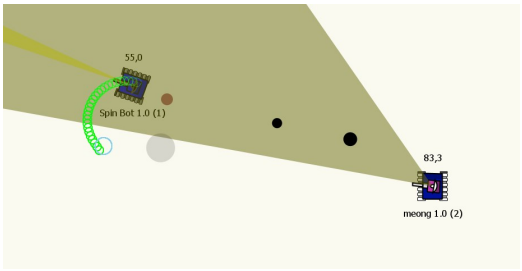
Dengan mekanisme ini, bot "Meong" tidak hanya bereaksi, tetapi juga belajar dan beradaptasi secara terus-menerus, menjadikannya lawan yang semakin tangguh dan sulit diprediksi seiring berjalannya waktu.

IV. KESIMPULAN

Penelitian ini berhasil merancang, mengimplementasikan, dan menguji sebuah sistem penargetan prediktif yang canggih dan adaptif untuk lingkungan kompetitif Robocode. Berdasarkan analisis dan implementasi yang telah diuraikan pada bab-bab sebelumnya, dapat ditarik beberapa kesimpulan utama yang saling menguatkan, yang tidak hanya relevan bagi Robocode tetapi juga bagi desain agen cerdas secara umum.

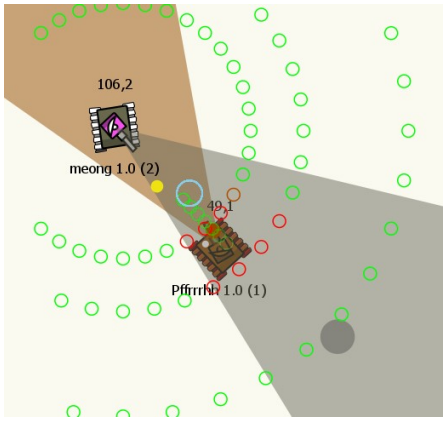
1. Pencocokan Pola sebagai Paradigma Penargetan yang Unggul: Sistem penargetan berbasis pencocokan pola terbukti secara konseptual dan praktis lebih superior dibandingkan metode penargetan konvensional. Metode linear dan sirkular secara inheren memperlakukan lawan sebagai objek fisika yang tunduk pada asumsi pergerakan sederhana. Sebaliknya, pendekatan berbasis pola secara fundamental mengubah cara pandang ini: ia

memperlakukan lawan sebagai agen cerdas dengan kebiasaan, kecenderungan, dan artefak algoritmik. Dengan menganalisis histori pergerakan, bot mampu memprediksi manuver-manuver non-trivial—seperti gerakan menghindar "Stop-and-Go" yang dirancang untuk mengelabui penargetan linear, osilasi maju-mundur yang mematahkan asumsi sirkular, atau bahkan pola spesifik "wall-smoothing" yang digunakan bot untuk menavigasi di dekat dinding. Ini menandai pergeseran paradigma dari masalah prediksi fisika menjadi masalah analisis perilaku, yang terbukti jauh lebih efektif dalam lingkungan adversarial di mana kesuksesan bergantung pada kemampuan untuk "membaca" niat lawan, bukan hanya lintasannya.



Gambar 2. Visualisasi prediksi lintasan pergerakan musuh yang didapatkan menggunakan pencocokan pola Boyer-Moore.

2. Fleksibilitas Multi-Panjang sebagai Kunci Adaptabilitas Dinamis: Implementasi strategi pencocokan pola multi-panjang memberikan keuntungan adaptif yang signifikan, memungkinkan sistem untuk secara cerdas menyeimbangkan antara spesifisitas dan generalisasi. Di awal pertempuran, ketika data masih langka, bot mungkin hanya menemukan kecocokan pada pola-pola pendek (5-7 *tick*) yang merepresentasikan manuver menghindar dasar dan reaktif. Namun, seiring berjalannya waktu dan data terkumpul, bot mampu mengidentifikasi dan memanfaatkan pola-pola yang lebih panjang dan spesifik (20-30 *tick*), yang mungkin merefleksikan seluruh sekuens strategi lawan, seperti "menembak, bergerak ke kiri, memutar radar, lalu kembali ke kanan". Kemampuan untuk secara dinamis "memilih" panjang pola terbaik yang tersedia memastikan sistem dapat memberikan prediksi yang relevan dalam berbagai kondisi. Ini membuat bot menjadi tangguh, baik saat menghadapi lawan baru dengan perilaku tak dikenal, maupun saat menghadapi lawan lama dengan strategi kompleks yang sudah terekam. Kemampuan ini bisa dianalogikan dengan proses belajar manusia, dari mengenali kata-kata tunggal hingga memahami kalimat-kalimat panjang yang sarat konteks.
3. Memori Persisten sebagai Fondasi Pembelajaran Strategis Jangka Panjang: Modul memori persisten yang menyimpan dan memuat data pergerakan antar pertandingan berhasil mengubah bot dari agen reaktif menjadi agen yang mampu belajar secara strategis. Keunggulan kompetitif terbesar dari arsitektur ini adalah kemampuannya untuk melakukan *meta-learning*. Ini berarti bot tidak hanya belajar tentang taktik sebuah bot dalam satu ronde, tetapi juga berpotensi belajar tentang *gaya pemrograman* dari pembuat bot tersebut. Sebagai contoh, jika seorang pemrogram cenderung menggunakan algoritma penghindar dinding sumber terbuka yang sama di beberapa bot ciptaannya, "Meong" akan mampu mengenali dan mengeksploitasi kelemahan fundamental dari algoritma tersebut secara lintas-pertandingan. Saat bertemu dengan bot baru dari pemrogram yang sama, "Meong" sudah memiliki "pemahaman awal" tentang bagaimana bot tersebut akan berperilaku di dekat dinding, memberikannya keuntungan strategis bahkan sebelum tembakan pertama dilepaskan.
4. Sinergi Holistik antara Subsistem Ofensif dan Defensif: Keberhasilan bot tidak dapat diatribusikan pada satu modul saja, melainkan pada sinergi holistik antara modul penargetan prediktif dan modul pergerakan Anti-Gravitasi. Interaksi ini menciptakan sebuah *positive feedback loop* yang kuat. Pergerakan defensif yang efektif tidak hanya memperpanjang kelangsungan hidup bot, tetapi juga secara aktif membentuk medan pertempuran. Dengan terus bergerak dan memaksa lawan untuk terus-menerus bereaksi dan menyesuaikan bidikannya, modul pergerakan dapat secara tidak langsung "memancing" lawan untuk masuk ke dalam pola-pola menghindar yang lebih dapat diprediksi. Setiap kali lawan terpaksa melakukan manuver defensif, ia menghasilkan data baru yang bersih. Data ini kemudian diumpankan ke modul penargetan, yang pada gilirannya meningkatkan kualitas prediksi tembakan. Tembakan yang lebih akurat memberikan tekanan lebih besar pada lawan, memaksanya melakukan lebih banyak gerakan defensif, yang kembali menghasilkan lebih banyak data.



Gambar 3. Visualisasi daerah yang memiliki gravitasi tinggi (merah).

Secara keseluruhan, optimasi penargetan menggunakan pencocokan pola Boyer-Moore dengan memori persisten menunjukkan bahwa performa agen otonom dalam lingkungan yang kompleks dapat ditingkatkan secara signifikan dengan

beralih dari model berbasis asumsi ke model berbasis data historis. "Meong" berfungsi sebagai bukti konsep yang kuat bahwa agen yang mampu mengingat, mengenali pola, dan beradaptasi memiliki keunggulan inheren atas agen yang hanya bereaksi terhadap kondisi sesaat.

UCAPAN TERIMA KASIH

Pertama-tama, penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa, karena atas berkat dan rahmat-Nya penulis dapat menyelesaikan makalah ini dengan baik dan tepat waktu. Penulis juga mengucapkan terima kasih kepada orang tua karena selalu memberikan semangat dan dukungan, termasuk dalam proses pembuatan makalah ini. Tidak lupa, penulis juga mengucapkan terima kasih kepada Bapak Dr. Rinaldi Munir, Strategi Algoritma kelas K02, Semester II Tahun 2024/2025, karena telah memberikan bimbingan dan ilmu pengetahuan yang berharga dalam penyelesaian makalah ini. Penulis juga mengucapkan terima kasih kepada teman-teman yang telah membantu dalam penyelesaian makalah ini

REFERENSI

[1] Boyer, R. S., & Moore, J. S. 1977. *A Fast String Searching Algoritm*. Communications of the ACM, 20(20). https://dl.acm.org/doi/pdf/10.1145/359842.359859 (Diakses 15 Juni 2025).

[2] Munir, Rinaldi. 2025. Pencocokan String (String/Pattern Matching). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf (Diakses 15 Juni 2025).

[3] Nelson, M. (n.d.). *RoboWiki*. https://robowiki.net/ (Diakses 15 Juni 2025).

PRANALA

[1] Repositori bot “meong” https://github.com/aibrahim185/meong.git (Diakses 15 Juni 2025).

[2] Repositori arsip bot robocode C#/dotnet https://github.com/aibrahim185/meong.git (Diakses 15 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, 15 Juni 2025

Ahmad Ibrahim, 13523089