

LAPORAN TUGAS BESAR 1 IF2211 STRATEGI ALGORITMA

Implementasi Algoritma *Greedy* pada Pembuatan Bot Robocode Tank Royale



Dosen Pengampu : Dr. Nur Ulfa Maulidevi, S.T., M.Sc.

Asisten Pembimbing : Farhan Nafis Rayhan (13522037)

Disusun oleh:

Kelompok 23 – LastWORGHXHXX

Nathaniel Jonathan Rusli (13523013)

Maheswara Bayu Kaindra (13523015)

Peter Wongsoredjo (13523039)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

KATA PENGANTAR

Dengan penuh rasa syukur, kami menyusun dan mempersembahkan makalah ini yang membahas topik menarik dan relevan dalam dunia algoritma dan teknologi kecerdasan buatan, yaitu “Implementasi Algoritma *Greedy* pada Pembuatan Bot Robocode Tank Royale”. Makalah ini disusun sebagai bagian dari Tugas Besar 1 IF2211 Strategi Algoritma, yang bertujuan untuk menggali pemahaman dan penerapan strategi algoritma *greedy* melalui pemecahan masalah berbasis simulasi dan strategi permainan.

Makalah ini tidak hanya membahas konsep dasar dari algoritma *greedy*, tetapi juga mengeksplorasi penerapannya dalam konteks yang dinamis dan kompetitif, yaitu dalam pengembangan bot untuk permainan Robocode Tank Royale. Algoritma *greedy* merupakan algoritma yang sederhana namun efektif dalam membuat keputusan lokal yang optimal pada setiap langkah. Di dalam makalah ini, kami menjelaskan secara rinci bagaimana strategi *greedy* kami integrasikan ke dalam logika dan aksi bot berdasarkan heuristik yang dipilih, serta bagaimana keputusan taktis tersebut berdampak pada hasil pertandingan antarbot.

Melalui makalah ini, kami berharap dapat memberikan penjelasan yang komprehensif mengenai hubungan antara teori algoritma dan implementasi praktisnya, serta menunjukkan bagaimana pendekatan berbasis heuristik dapat memengaruhi kinerja dalam lingkungan yang kompetitif. Kami juga memperkenalkan pilihan bot utama yang akan digunakan sebagai lawan dalam pertandingan.

Kami mengucapkan terima kasih kepada dosen pengampu dan semua pihak yang telah memberikan arahan, dukungan, serta semangat selama proses penyusunan makalah ini. Akhir kata, kami berharap makalah ini dapat menjadi sumber inspirasi bagi para pembaca untuk terus mengeksplorasi peluang dan tantangan baru di bidang strategi algoritma dan pengembangan kecerdasan buatan.

Bandung, 23 Maret 2025

Nathaniel Jonathan Rusli (13523013)

Maheswara Bayu Kaindra (13523015)

Peter Wongsoredjo (13523039)

DAFTAR ISI

KATA PENGANTAR.....	2
DAFTAR ISI.....	3
DAFTAR TABEL.....	4
DAFTAR PEMBAGIAN TUGAS.....	5
BAB I DESKRIPSI TUGAS.....	6
1.1. Rounds dan Turns.....	6
1.2. Batas Waktu Giliran.....	7
1.3. Energi.....	7
1.4. Peluru.....	8
1.5. Panas Meriam (<i>Gun Heat</i>).....	8
1.6. Tabrakan.....	8
1.7. Bagian Tubuh Tank.....	8
1.8. Pergerakan.....	9
1.9. Berbelok.....	9
1.10. Pemindaian.....	9
1.11. Skor.....	10
BAB II LANDASAN TEORI.....	12
2.1. Algoritma Greedy.....	12
2.2. Cara Kerja Program.....	14
2.2.1 Aksi-Aksi Bot.....	14
2.2.1.1 Inisialisasi Bot.....	14
2.2.1.2 Siklus atau Loop Utama.....	15
2.2.1.3 Respon Bot Terhadap Event.....	16
2.2.2. Implementasi Algoritma Greedy pada Bot.....	17
2.2.2.1 Heuristik Menembak.....	18
2.2.2.2 Heuristik Pergerakan.....	19
2.2.2.3 Heuristik Scanning Radar.....	19
2.2.3 Menjalankan Bot.....	20
BAB III APLIKASI STRATEGI GREEDY.....	21
3.1 Bot 1: Made in China.....	21
3.1.1 Pemetaan Elemen-Elemen <i>Greedy (first scanned)</i>	22
3.1.2 Analisis Efisiensi dan Efektivitas: <i>First Enemy Scanned</i>	24
3.2 Alternatif <i>Greedy: Weakest Enemy</i> (Bot EDI).....	24
3.2.1 Pemetaan Elemen-Elemen <i>Greedy: Weakest Enemy</i>	25
3.2.2 Analisis Efisiensi dan Efektivitas: <i>Weakest Enemy</i>	27
3.3 Alternatif <i>Greedy: Concentrated Sector</i> (Bot Hari Styles).....	27
3.3.1 Pemetaan Elemen-Elemen <i>Greedy: Concentrated Sector</i>	28
3.3.2 Analisis Efisiensi dan Efektivitas: <i>Concentrated Sector</i>	29
3.4 Alternatif <i>Greedy: Survivorship</i> (Bot Steve).....	30

3.4.1 Pemetaan Elemen-Elemen <i>Greedy: Survivorship</i>	30
3.4.2 Analisis Efisiensi dan Efektivitas: <i>Survivorship</i>	32
3.5 Strategi <i>Greedy</i> yang Diimplementasikan.....	33
BAB IV IMPLEMENTASI DAN PENGUJIAN	35
4.1 Implementasi.....	35
4.2 Struktur Data Program.....	51
4.3 Analisis dan Pengujian.....	53
BAB V KESIMPULAN DAN SARAN	57
5.1 Kesimpulan.....	57
5.2 Saran.....	58
LAMPIRAN	59
DAFTAR PUSTAKA	60

DAFTAR TABEL

Tabel 1. Pembagian Tugas.....	6
Tabel 2. Kumpulan Contoh Method Aksi Bot.....	16
Tabel 3. Kumpulan Contoh Method Respon Event.....	17
Tabel 4. Overview Bot dengan Algoritma Greedy.....	22
Tabel 5. Penjelasan Struktur Data.....	52
Tabel 6. Penjelasan Fungsi dan Prosedur.....	53
Tabel 7. Implementasi dan Pengujian.....	54

DAFTAR PEMBAGIAN TUGAS

Tabel 1. Pembagian Tugas

Tugas		PIC
1.	Merancang bot EDI dan bot Steve.	Nathaniel Jonathan Rusli (13523013)
2.	Merancang bot Made In China dan bot Hari Styles.	Maheswara Bayu Kandra (13523015)
3.	Merancang bot Hari Styles dan bot EDI.	Peter Wongsoredjo (13523039)

BAB I

DESKRIPSI TUGAS

Robocode adalah permainan pemrograman yang bertujuan untuk membuat kode bot dalam bentuk tank virtual untuk berkompetisi melawan bot lain di arena. Pertempuran Robocode berlangsung hingga bot-bot bertarung hanya tersisa satu seperti permainan Battle Royale, karena itulah permainan ini dinamakan Tank Royale. Nama Robocode adalah singkatan dari "Robot code," yang berasal dari versi asli/pertama permainan ini. Robocode Tank Royale adalah evolusi/versi berikutnya dari permainan ini, di mana bot dapat berpartisipasi melalui Internet/jaringan. Dalam permainan ini, pemain berperan sebagai programmer bot dan tidak memiliki kendali langsung atas permainan. Pemain hanya bertugas untuk membuat program yang menentukan logika atau "otak" bot. Program yang dibuat akan berisi instruksi tentang cara bot bergerak, mendeteksi bot lawan, menembakkan senjatanya, serta bagaimana bot bereaksi terhadap berbagai kejadian selama pertempuran.

Pada Tugas Besar pertama Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Tentunya mahasiswa harus menggunakan strategi *greedy* dalam membuat bot ini. Komponen-komponen dari permainan ini antara lain:

1.1. Rounds dan Turns

Pertempuran dapat terdiri dari beberapa rounds. Secara default, satu pertempuran berisi 10 rounds, di mana setiap rounds akan memiliki pemenang dan yang kalah. Setiap round dibagi menjadi beberapa turns, yang merupakan unit waktu terkecil. Satu turn adalah satu ketukan waktu dan satu putaran permainan. Jumlah turn dalam satu round tergantung pada berapa lama waktu yang dibutuhkan hingga hanya tersisa bot terakhir yang bertahan. Pada setiap turn, sebuah bot dapat:

- Menggerakkan bot, memindai musuh, dan menembakkan senjata.
- Bereaksi terhadap peristiwa seperti saat bot terkena peluru atau bertabrakan dengan bot lain atau dinding.

- Perintah untuk bergerak, berputar, memindai, menembak, dan sebagainya dikirim ke server untuk setiap turn.

Perlu diperhatikan bahwa API (Application Programming Interface) bot resmi secara otomatis mengirimkan niat bot ke server di balik layar, sehingga Anda tidak perlu mengkhawatirkannya, kecuali jika Anda membuat API Bot sendiri. Pada setiap turn, bot akan secara otomatis menerima informasi terbaru tentang posisinya dan orientasinya di medan perang. Bot juga akan mendapatkan informasi tentang bot musuh ketika mereka terdeteksi oleh pemindai. Perlu diketahui bahwa game engine yang akan digunakan pada tugas besar ini tidak mengikuti aturan default mengenai komponen Round & Turns.

1.2. Batas Waktu Giliran

Penting untuk dicatat bahwa setiap bot memiliki batas waktu untuk setiap turn yang disebut turn timeout, biasanya antara 30-50 ms (dapat diatur sebagai aturan pertempuran). Ini berarti bahwa bot tidak bisa mengambil waktu sebanyak yang mereka inginkan untuk bergerak dan menyelesaikan turn saat ini.

Setiap kali turn baru dimulai, penghitung waktu ulang diatur ulang dan mulai berjalan. Jika batas waktu tercapai dan bot tidak mengirimkan pergerakannya untuk turn tersebut, maka tidak ada perintah yang dikirim ke server. Akibatnya, bot akan melewati turn tersebut. Jika bot melewati turn, ia tidak akan bisa menyesuaikan gerakannya atau menembakkan senjatanya karena server tidak menerima perintah tepat waktu sebelum turn berikutnya dimulai.

1.3. Energi

Semua bot memulai permainan dengan jumlah energi awal sebanyak 100 poin energi.

- Bot akan kehilangan energi jika ditembak atau ditabrak oleh bot musuh.
- Bot juga akan kehilangan energi jika menembakkan meriamnya.
- Bot akan mendapatkan energi jika peluru dari meriamnya mengenai musuh. Energi yang didapat akan lebih banyak 3 kali lipat dari energi yang digunakan untuk menembakkan peluru.
- Bot dengan energi nol akan dinonaktifkan dan tidak bisa bergerak. Jika bot terkena serangan dalam keadaan ini, bot akan hancur.

1.4. Peluru

Semakin banyak energi (daya tembak) yang digunakan untuk menembakkan peluru, semakin berat peluru tersebut dan semakin lambat gerakannya. Namun, peluru yang lebih berat juga menghasilkan lebih banyak kerusakan dan memungkinkan bot mendapatkan lebih banyak energi saat mengenai bot musuh. Seperti disebutkan sebelumnya, peluru yang lebih berat akan bergerak lebih lambat. Ini berarti akan membutuhkan waktu lebih lama untuk mencapai target, meningkatkan risiko peluru tidak mengenai sasaran. Sebaliknya, peluru yang lebih ringan bergerak lebih cepat, sehingga lebih mudah mengenai target, tetapi peluru ringan tidak memberikan banyak poin energi saat mengenai bot musuh.

1.5. Panas Meriam (*Gun Heat*)

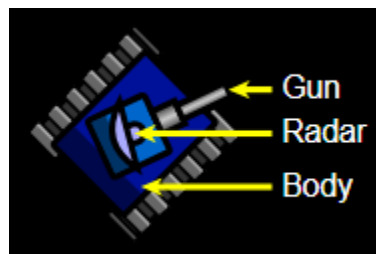
Saat menembakkan peluru, meriam akan menjadi panas. Peluru yang lebih berat menghasilkan lebih banyak panas dibandingkan peluru yang lebih ringan. Ketika meriam terlalu panas, bot tidak dapat menembak hingga suhu meriam turun ke nol. Selain itu, meriam juga sudah dalam keadaan panas di awal round dan perlu waktu untuk mendingin sebelum bisa digunakan untuk pertama kalinya.

1.6. Tabrakan

Perlu diperhatikan bahwa bot akan menerima kerusakan jika menabrak dinding (batas arena), yang disebut wall damage. Hal yang sama juga terjadi jika bot bertabrakan dengan bot lain. Jika bot menabrak bot musuh dengan bergerak maju, ini disebut ramming (menabrak dengan sengaja), yang akan memberikan sedikit skor tambahan bagi bot yang menyerang.

1.7. Bagian Tubuh Tank

Tubuh tank terdiri dari 3 bagian:



Body adalah bagian utama dari tank yang digunakan untuk menggerakkan tank. *Gun* digunakan untuk menembakkan peluru dan dapat berputar bersama *body* atau independen dari *body*. *Radar* digunakan untuk memindai posisi musuh dan dapat berputar bersama *body* atau independen dari *body*.

1.8. Pergerakan

Bot dapat bergerak maju dan mundur hingga kecepatan maksimum. Dibutuhkan beberapa giliran untuk mencapai kecepatan maksimum. Bot dapat mengalami percepatan maksimum sebesar 1 unit per giliran dan pengereman dengan perlambatan maksimum 2 unit per giliran. Percepatan dan perlambatan maksimum tidak bergantung pada kecepatan bot saat itu.

1.9. Berbelok

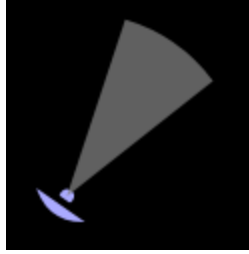
Seperti yang disebutkan sebelumnya, bagian tubuh, turret (meriam), dan radar dapat berputar secara independen satu sama lain. Jika turret atau radar tidak diputar, maka keduanya akan mengarah ke arah yang sama dengan tubuh bot. Setiap bagian tubuh memiliki kecepatan putar yang berbeda. Radar adalah bagian tercepat dan dapat berputar hingga 45 derajat per giliran, yang berarti dapat berputar 360 derajat dalam 8 giliran. Turret dan meriam dapat berputar hingga 20 derajat per giliran.

Bagian paling lambat adalah tubuh tank, yang dalam kondisi terbaik dapat berputar hingga 10 derajat per giliran. Namun, ini bergantung pada kecepatan bot saat ini. Semakin cepat bot bergerak, semakin lambat kemampuannya untuk berbelok. Perlu diperhatikan bahwa tidak ada energi yang dikonsumsi saat bot bergerak atau berbelok.

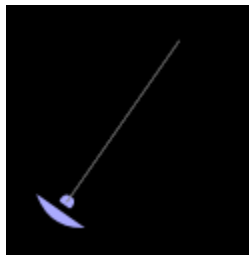
1.10. Pemindaian

Aspek penting dalam Robocode adalah memindai bot musuh menggunakan radar. Radar dapat mendeteksi bot dalam jangkauan hingga 1200 piksel. Musuh yang berada lebih dari 1200 piksel dari bot tidak dapat terdeteksi atau dipindai oleh radar.

Penting untuk diperhatikan bahwa sebuah bot hanya dapat memindai bot musuh yang berada dalam jangkauan sudut pemindaian (scan arc)-nya. Sudut pemindaian ini merupakan "sapuan radar" dari arah radar sebelumnya ke arah radar saat ini dalam satu giliran.



Jika radar tidak bergerak dalam suatu giliran, artinya radar tetap mengarah ke arah yang sama seperti pada giliran sebelumnya, maka sudut pemindaian akan menjadi nol derajat, dan bot tidak akan dapat mendeteksi musuh.



Oleh karena itu, sangat disarankan untuk selalu mengubah arah radar agar tetap dapat memindai musuh.

1.11. Skor

Pada akhir pertempuran, setiap bot akan diranking berdasarkan total skor yang diperoleh masing-masing bot selama keseluruhan pertempuran. Tentunya, tujuan utama pada tugas besar ini adalah membuat bot yang memberikan skor setinggi mungkin. Berikut adalah rincian komponen skor pada pertempuran:

- Bullet Damage: Bot mendapatkan poin sebesar *damage* yang dibuat kepada bot musuh menggunakan peluru.
- Bullet Damage Bonus: Apabila peluru berhasil membunuh bot musuh, bot mendapatkan poin sebesar 20% dari *damage* yang dibuat kepada musuh yang terbunuh.
- Survival Score: Setiap ada bot yang mati, bot lainya yang masih bertahan pada ronde tersebut mendapatkan 50 poin.
- Last Survival Bonus: Bot terakhir yang bertahan pada suatu ronde akan mendapatkan 10 poin dikali dengan banyaknya musuh.

- Ram Damage: Bot mendapatkan poin sebesar 2 kalinya *damage* yang dibuat kepada bot musuh dengan cara menabrak.
- Ram Damage Bonus: Apabila musuh terbunuh dengan cara ditabrak, bot mendapatkan poin sebesar 30% dari *damage* yang dibuat kepada musuh yang terbunuh.

Skor akhir bot adalah akumulasi dari 6 komponen diatas. Perlu diperhatikan bahwa game akan menampilkan berapa kali suatu bot meraih peringkat 1, 2, atau 3 pada setiap ronde. Namun, hal ini tidak dihitung sebagai komponen skor maupun untuk perangkingan akhir. Bot yang dianggap menang pertempuran adalah bot dengan akumulasi skor tertinggi.

BAB II

LANDASAN TEORI

2.1. Algoritma *Greedy*

Menurut Munir (2025), algoritma *greedy* merupakan algoritma yang memecahkan persoalan secara langkah per langkah sedemikian sehingga pada setiap langkah algoritma mengambil pilihan yang terbaik yang diperoleh saat itu tanpa memperhatikan konsekuensi kedepannya. Dalam kata lain, algoritma “berharap” bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimal global.

Pada algoritma *greedy*, terdapat elemen-elemen seperti:

1. Himpunan kandidat (C): merupakan himpunan berisi kandidat langkah yang akan dipilih pada setiap langkah.
2. Himpunan solusi (S): merupakan himpunan berisi kandidat yang sudah dipilih.
3. Fungsi solusi (*solution function*): merupakan fungsi yang menentukan apakah himpunan kandidat telah memberikan solusi.
4. Fungsi seleksi (*selection function*): merupakan fungsi yang memilih kandidat berdasarkan strategi *greedy* yang digunakan.
5. Fungsi kelayakan (*feasibility function*): merupakan fungsi yang memeriksa apakah kandidat yang dipilih layak atau tidak untuk dimasukkan ke dalam himpunan solusi (S).
6. Fungsi objektif: merupakan fungsi yang bertujuan untuk memaksimumkan atau meminimumkan objektif.

Menurut Munir (2025) dan Anissa (2023), algoritma *greedy* tentunya memiliki kelebihan dibandingkan dengan algoritma lainnya. Algoritma ini dikenal sebagai algoritma yang relatif mudah dipahami dan diimplementasikan, dan seringkali kinerja yang didapatkan memberikan hasil yang cepat. Algoritma ini juga cenderung menghasilkan solusi yang cukup mendekati optimal dengan waktu yang lebih singkat dibandingkan algoritma-algoritma lainnya.

Namun, algoritma *greedy* juga memiliki berbagai kelemahan. Untuk berbagai persoalan, seperti *0/1 knapsack*, *shortest path*, *minimum coin change*, dan personal lainnya, algoritma *greedy* tidak selalu memberikan hasil yang optimal. Hal ini dapat diakibatkan oleh cara kerja

alami dari algoritma *greedy* itu sendiri, yaitu algoritma mengabaikan konsekuensi masa depan atau berfokus pada optimum lokal. Tidak optimalnya hasil algoritma *greedy* juga dapat disebabkan oleh pemilihan fungsi seleksi yang tidak tepat. Algoritma *greedy* mengharuskan penulis algoritma dan program memilih kriteria yang tepat dan optimal, di mana kegagalan dalam pemilihan kriteria algoritma *greedy* dapat berakhir dengan solusi yang jauh dari solusi optimal.

Mengetahui kapabilitas algoritma *greedy*, bila solusi terbaik mutlak tidak diperlukan pada kasus tertentu, maka algoritma *greedy* dapat dimanfaatkan untuk menghasilkan *approximation* atau solusi hampiran. Metode penggunaan algoritma *greedy* seperti ini dapat menghemat waktu secara signifikan, di mana algoritma yang membutuhkan solusi optimal berpotensi membutuhkan waktu yang eksponensial.

```
function greedy( $C : \text{himpunan\_kandidat}$ )  $\rightarrow$   $\text{himpunan\_solusi}$ 
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }

Deklarasi
     $s : \text{kandidat}$ 
     $S : \text{himpunan\_solusi}$ 

Algoritma
 $S \leftarrow \{\}$  { inisialisasi  $S$  dengan kosong }
while (not SOLUSI( $S$ ) and ( $C \neq \{\}$ )) do
     $x \leftarrow \text{SELEKSI}(C)$ 
     $C \leftarrow C - \{x\}$ 
    if LAYAK( $S \cup \{x\}$ ) then
         $S \leftarrow S \cup \{x\}$ 
    endif
endwhile
{ SOLUSI( $S$ ) atau  $C = \{\}$  }
if (SOLUSI( $S$ )) then
    return  $S$ 
else
    write("Tidak ada solusi.")
```

2.2 Cara Kerja Program

2.2.1 Aksi-Aksi Bot

Setiap bot dalam Robocode Rank Royale memiliki suatu siklus eksekusi berulang sepanjang pertandingan antarbot. Siklus aksi suatu robot dapat dibedah menjadi beberapa tahap, yaitu inisialisasi bot, *loop* utama, dan respon bot terhadap suatu *event* yang akan didasari prinsip algoritma *greedy*.

2.2.1.1 Inisialisasi Bot

Tahap pertama dalam siklus aksi robot adalah inisialisasi bot, yaitu suatu tahap awal atau penyesuaian di mana bot akan mengatur beberapa parameter dasar sebelum bot tersebut mulai bertarung. Inisialisasi bot dispesifikasikan *method* Run() dengan memanggil *attribute* dan *method* aksi lainnya di dalamnya agar bot siap bertarung. Contoh aksi yang biasanya dilakukan bot dalam tahapan ini antara lain menyesuaikan kecepatan pergerakan bot, menyiapkan senjata untuk menyerang, mengatur pola pergerakan awal seperti maju setelah pertandingan dimulai, dan juga mengaktifkan radar untuk deteksi awal dengan *continuous scanning* atau *locking radar*. Berikut merupakan contoh kode pemanggilan *method* pada tahap inisialisasi bot:

```
public override void Run()
{
    // Tahap inisialisasi dapat digunakan untuk konfigurasi warna
    BodyColor      = Color.Red;
    BulletColor     = Color.Red;
    RadarColor      = Color.Red;
    ScanColor       = Color.Red;
    TurretColor     = Color.Red;

    // Tahap inisialisasi juga dapat digunakan untuk deklarasi aksi-aksi bot
    if (RadarTurnRemaining == 0)
    { // Misal radar bot terus-menerus berputar untuk scanning
        SetTurnRadarRight(Double.PositiveInfinity);
    }
}
```

2.2.1.2 Siklus atau *Loop* Utama

Selanjutnya, suatu bot akan memiliki suatu *loop* atau siklus utama di mana bot akan terus menjalankan perintah tertentu sampai terinterupsi suatu *event*, misalnya bergerak, mendeteksi musuh, dan mungkin juga menembak. Biasanya tahap *loop* utama diimplementasikan dalam *method* *Run()* dan dipasangkan dengan sebuah *loop while (IsRunning)*. Contoh sederhana untuk tahap *loop* utama adalah pengaturan pergerakan zig-zag untuk secara terus menerus untuk menghindari tembakan lawan, misal dengan sekuens *method* *Ahead()*, *TurnRight()*, *Back()*, dan *TurnLeft()*. Contoh yang lebih realistis dalam implementasi bot dalam Robocode adalah robot melakukan *continuous scanning* dengan bergerak atau berputar secara terus menerus hingga mendeteksi bot musuh dan melakukan aksi tertentu. Perancang bot dapat memilih *method* apapun beserta *attribute* pendukungnya yang digunakan dalam siklus utama, misal *attribute* dan *method* yang sering digunakan seperti yang terdapat pada **Tabel 2**.

Tabel 2. Kumpulan Contoh *Method* Aksi Bot

Kategori Aksi	Kumpulan <i>Attribute</i> dan <i>Method</i>
Pergerakan Bot	X, Y, Speed, TurnRate, MaxSpeed, Forward(distance), SetForward(distance), Back(distance), SetBack(distance), SetMaxVelocity(speed), SetTurnRate(rate), TurnLeft(angle), SetTurnLeft(angle), SetTurnRight(angle), TurnRight(angle), SetMaxSpeed(speed)
Radar Bot	RadarTurnRemaining, RadarDirection, TurnRadarLeft(angle), SetTurnRadarLeft(angle), TurnRadarRight(angle), SetTurnRadarRight(angle)
Penembakan Bot	GunHeat, GunDirection, Energy, Fire(power), SetFire(power), TurnGunLeft(angle), SetTurnGunLeft(angle), TurnGunRight(angle), SetTurnGunRight(angle)
Pertahanan Bot	Scan(), OnHitByBullet(), OnBulletHitBullet(), OnBulletHitOpponent(), OnHitWall(), OnScannedBot(), OnBotDeath(), DistanceTo(x, y), BearingTo(x, y)

Berikut merupakan contoh kode pada tahapan *loop* utama:

```
public override void Run()
{
    // Misal loop utama untuk bergerak zig-zag
    while (IsRunning) // Loop utama yang berjalan terus-menerus
    {
        Forward(100);    // Bergerak maju sejauh 100 unit
        TurnRight(45);   // Berputar ke kanan 45 derajat
        Back(100);       // Mundur sejauh 100 unit
        TurnLeft(45);    // Berputar ke kiri 45 derajat
    }
}
```

2.2.1.3 Respon Bot Terhadap *Event*

Aksi suatu bot tidak hanya didefinisikan oleh tahap inisialisasi dan siklus secara terus menerus, tapi juga dengan respon terhadap suatu *event*. Pada Robocode Tank Royale, konfigurasi respon suatu bot dapat dilakukan dengan *override method-method* yang umum digunakan seperti tertentu:

Tabel 3. Kumpulan Contoh *Method* Respon *Event*

<i>Method</i>	Definisi dan Aksi yang Biasa Dilakukan
OnScannedRobot(...)	Melakukan sekumpulan aksi terdefinisi ketika radar mendeteksi musuh, biasanya bot akan melakukan aksi penembakan dengan metode <i>locking</i> .
OnHitByBullet(...)	Melakukan sekumpulan aksi terdefinisi ketika bot terkena oleh peluru bot musuh, biasanya bot akan bergerak sesuai preferensi perancang bot untuk menghindari dari peluru bot musuh.
OnBulletHit(...)	Melakukan sekumpulan aksi terdefinisi ketika peluru bot mengenai musuh, bias digunakan untuk meningkatkan agresivitas jika serangan berhasil.
OnBulletFired(...)	Melakukan sekumpulan aksi terdefinisi ketika peluru bot internal ditembakkan, biasanya digunakan untuk bergerak setelah menembak sesuai heuristik tertentu.

OnBulletMissed(...)	Melakukan sekumpulan aksi terdefinisi ketika peluru yang ditembakkan bot meleset, biasa digunakan untuk menyesuaikan strategi atau konfigurasi penembakkan bot.
OnHitWall(...)	Melakukan sekumpulan aksi terdefinisi ketika bot menabrak dinding, biasanya bot akan mundur dan berputar untuk menghindari tembok.
OnRobotDeath(...)	Melakukan sekumpulan aksi terdefinisi ketika bot musuh mati, biasanya bot akan mencari target baru sesuai metode yang dipilih.
OnCustomEvent(...)	Memungkinkan perancang aksi bot untuk membuat <i>event</i> khusus sesuai kondisi tertentu untuk mengimplementasikan strategi tambahan.
OnWin(...)	Melakukan sekumpulan aksi terdefinisi ketika bot memenangkan pertandingan, biasanya digunakan untuk melakukan selebrasi atau <i>taunting</i> pada bot-bot musuh.

Berikut merupakan contoh kode yang menerapkan *method* untuk merespon suatu *event*:

```
// Method berikut akan dipanggil dan override ketika bot menabrak tembok arena
public override void OnHitWall(HitWallEvent e)
{
    Back(50);      // Bergerak mundur sedikit
    TurnRight(90); // Berputar menjauhi tembok
}
```

2.2.2. Implementasi Algoritma *Greedy* pada Bot

Untuk mengimplementasikan algoritma *greedy*, perancang bot perlu memahami objektif dan kondisi menang permainan Robocode Tank Royale, yaitu untuk memaksimalkan skor akhir robot yang dapat diperoleh dari *bullet damage*, *bullet damage bonus* (ketika peluru berhasil membunuh musuh), *survival score*, *last survival bonus*, *ram damage*, dan *ram damage bonus* (ketika berhasil membunuh bot musuh dengan cara menabraknya). Sekumpulan heuristik yang lahir dari mekanisme dan objektif permainan Robocode Tank Royale pun dapat diadaptasikan pada pemilihan *method-method* aksi di tahap apapun. Dalam konteks ini, heuristik memegang peran penting sebagai dasar dalam pengambilan keputusan yang cepat dan efisien.

Keputusan yang diambil oleh bot didasarkan pada informasi terbatas yang tersedia saat itu juga, misalnya posisi musuh, kecepatan, arah gerak, energi, dan informasi lainnya yang memungkinkan bot untuk menilai situasi berdasarkan indikator-indikator yang tersedia. Dengan demikian, heuristik yang digunakan dalam bot dapat dianggap sebagai penilaian lokal yang memprioritaskan tindakan tertentu di atas yang lain, ditujukan untuk tujuan optimum global berupa poin yang maksimal. Guna meningkatkan efektivitas bot dalam pertandingan dan mencapai optimum lokal, berbagai aspek strategi bot dapat diterapkan seperti:

2.2.2.1 Heuristik Menembak

Heuristik penembakan bot dapat didasari berbagai faktor seperti jarak bot musuh, tingkat energi bot, dan juga estimasi pergerakan musuh:

- Jarak Bot Musuh: Jarak suatu bot terhadap bot musuh biasanya digunakan untuk menentukan prioritas target, di mana semakin dekat bot musuh, maka semakin tinggi kemungkinan untuk mengenai sasaran tersebut. Misalnya, suatu tingkatan seperti kekuatan tembakan dapat disesuaikan terhadap jarak seperti berikut:
 - Jarak < 100 : Fire(3) (kekuatan tembakan dimaksimalkan karena kemungkinan mengenai target lebih tinggi).
 - Jarak 100 - 300: Fire(2) (tembakan berkekuatan sedang guna mencari keseimbangan efisiensi daya).
 - Jarak > 300 : Fire(1) (tembakan lemah, hanya digunakan untuk menekan lawan tanpa membuang banyak energi).
- Tingkat Energi Bot: Penyesuaian kekuatan tembakan juga dapat dilakukan berdasarkan tingkat energi bot saat itu juga. Misalnya energi bot yang rendah menyebabkan lebih konservatifnya penembakkan untuk mempertahankan daya. Strategi ini biasanya digunakan untuk bot yang mengutamakan *endurance* dan efisiensi dibandingkan agresivitas dalam menyerang.
- Prediksi atau Antisipasi Pergerakan Musuh: Bot dapat mengantisipasi musuh berdasarkan kecepatan geraknya, terutama yang bergerak dengan kecepatan konstan, kemudian mengarahkan tembakan ke posisi prediksi gerakan musuh, meningkatkan kemungkinan tembakan bot mengenai musuh.

2.2.2.2 Heuristik Pergerakan

Gerakan bot dalam Robocode dapat dimanipulasi sedemikian mungkin untuk menghindari atau kabur dari tembakan musuh, dan juga menjaga posisi optimal untuk menyerang, misalnya *locking*.

- Mode Penyerangan: Bot dapat mempertahankan jarak dengan musuh dengan mengejanya dalam mode penyerangan, misalnya bila bot musuh bergerak menjauh, bot dapat mengakselerasi gerakannya agar bot musuh tetap dalam *range* serangan.
- Mode Pertahanan: Selain untuk mode menyerang, pergerakan bot juga dapat digunakan untuk mode pertahanan. Untuk menghindari tembakan musuh, bot dapat bergerak secara *random*, zig-zag. Musuh yang menyerang dengan lebih agresif dapat direspons dengan pergerakan yang lebih agresif pula untuk menghindari tembakan musuh.
- Pergerakan *Random*: Dalam mode pertahanan, lebih baik lagi bila arah dan kecepatan gerak berubah secara *random* agar pergerakan tidak dapat ditebak oleh bot musuh.
- Tingkat Energi: Faktor tingkat energi juga dapat diadaptasikan pada heuristik pergerakan, di mana ketika energi bot hampir habis, strategi pun dialokasikan lebih untuk bertahan dengan menghindar lebih banyak daripada menyerang.
- Keadaan Arena: Pergerakan bot dapat diadaptasikan untuk menghindari menabraknya batas arena dengan mengubah arah pergerakannya

2.2.2.3 Heuristik *Scanning* Radar

Selain komponen heuristik penembakan dan pergerakan, heuristik *scanning* pada radar juga tidak kalah pentingnya dan dapat meningkatkan efektivitas bot dalam pertandingan secara signifikan bila diimplementasikan dengan baik. Dengan *scanning*, strategi penyerangan yang mematikan dapat diimplementasikan seperti *lock radar* pada satu musuh, di mana radar akan terus mengunci ke arah tersebut untuk mempertahankan target setelah musuh ditemukan. *Locking* pada musuh yang bergerak cepat dapat disesuaikan dengan mengubah rotasi radar agar musuh tetap terkunci.

Sebelum melakukan *locking*, biasanya suatu bot dapat mengimplementasikan strategi rotasi 360 derajat secara berkala untuk sebelum terdapat musuh yang terdeteksi guna mencari musuh baru yang mungkin lewat dalam jangkauan radar bot. Baru setelah musuh ditemukan, bot

akan melakukan *locking*, menjadikan gabungan kedua strategi tersebut strategi yang mematikan bagi bot musuh. Strategi yang tidak kalah efektif adalah *multi-target tracking*, di mana arena masih terisi oleh lebih dari satu musuh dan bot akan menyimpan data seluruh musuh yang terdeteksi, lalu memilih bot musuh yang paling dekat atau bot dengan energi yang paling rendah.

2.2.3 Menjalankan Bot

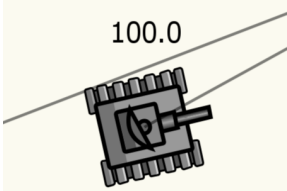
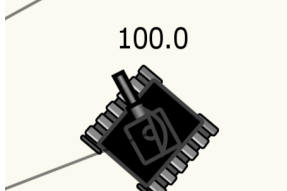
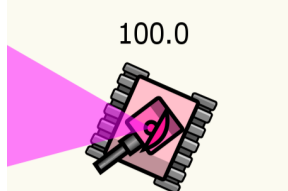

Berikut merupakan langkah-langkah menjalankan bot Robocode dengan C#:

1. Unduh dan instal .NET, gunakan .NET versi 6.0.
2. Buat proyek C# berbasis Class Library dengan menggunakan Visual Studio Code.
3. Tambahkan referensi API Robocode yang digunakan untuk mengembangkan bot.
4. Buatlah sebuah kelas Bot yang mewarisi Robot.
5. Tambahkan file-file lain yang dibutuhkan (ikuti template bot-bot template) dan simpan dalam satu direktori yang sama.
6. Kompilasi bot dengan menjalankan *dotnet build* pada direktori bot atau juga “./NamaBot.cmd” untuk Windows dan “./NamaBot.sh” untuk Linux/Mac.
7. Jalankan Robocode dan bot sudah dapat digunakan untuk bertanding.

BAB III

APLIKASI STRATEGI *GREEDY*

Tabel 4. *Overview* Bot dengan Algoritma *Greedy*

Made in China	EDI	Hari Styles	Steve
 <p>Made in China 1.0 (2)</p>	 <p>EDI 1.0 (1)</p>	 <p>Hari Styles 1.0 (2)</p>	 <p>Steve 1.0 (1)</p>
Menembak musuh yang pertama terlacak dari jarak ideal untuk memaksimalkan <i>bullet damage points</i> dan <i>survival points</i> .	Menyerang bot terlemah untuk mengoptimalkan <i>bullet damage bonus</i> dan <i>ram damage bonus</i>	Menyerang bot yang terdapat pada sektor paling padat, dan lari ke sektor sebaliknya untuk memaksimalkan <i>bullet damage points</i> dan <i>survival points</i> .	Menganalisis kondisi permainan untuk memilih gerakan melarikan diri terbaik untuk memaksimalkan <i>survival points</i> .

3.1 Bot 1: *Made in China*

Made in China didesain untuk dapat berperang di kondisi perang umum dengan kemampuan utama menembak. Bot ini dapat melakukan *aim tracking* dengan cukup akurat dan memilih untuk mengeluarkan peluru di momen yang ideal. Jika digambarkan secara iteratif, berikut merupakan algoritma greedy yang digunakan.

1. Melakukan *scanning* secara memutar.
2. Apabila terdapat bot musuh terlacak, kejar bot tersebut sampai jarak aman.
3. Ketika jarak sudah ideal, mulai tembak bot musuh, *power* beragam tergantung jarak (mulai dari 1.5 sampai 3).
4. Apabila bot musuh mendekat, jaga jarak aman (mundur).
5. Apabila tanpa sengaja menabrak bot musuh atau tembok, belok dan mundur.
6. Jika bot kehilangan *tracking* ke bot musuh atau bot musuh kehabisan energi, kembali pindai secara memutar.

Secara umum, bot akan mendekat ke arah musuh dan menembak musuh pertama yang ditarget. Namun, bot tidak pernah mau menabrak musuh sehingga akan terus menjaga jaraknya dengan musuh sesuai jarak aman. Apabila musuh yang maju mendekat, bot ini akan mundur dan berbelok sampai jarak antar bot aman.

3.1.1 Pemetaan Elemen-Elemen *Greedy (first scanned)*

- a. Himpunan Kandidat : Himpunan kandidat pada konteks ini berupa himpunan pilihan aksi yang dapat dilakukan bot setiap turn seperti menggerakkan bot (maju, mundur, kiri, dan kanan), menggerakkan radar (*scanning*), dan menembak dengan *power* tertentu.
- b. Himpunan Solusi : Himpunan solusi mencakup semua aksi yang dipilih untuk mencapai objektif. Dalam konteks ini objektifnya adalah untuk menjadi pemenang dengan poin tertinggi.
- c. Fungsi Solusi : Memeriksa apakah goal sudah dicapai. Pada implementasinya, pemeriksaan goal mencakup memeriksa status kemenangan dan memastikan bot masih hidup.

```
while isRunning do
... { iterasi }
```

- d. Fungsi Seleksi : Made in China didesain sebagai robot penembak jitu yang cenderung agresif sehingga akan menyerang bot pertama yang terlacak. Penyerangan tersebut difokuskan pada tembakan jarak dekat dan menengah, sehingga bot ini tidak akan menabrak musuh dan cenderung menjaga jarak aman. Oleh karena itu, Made in China berfokus pada poin *bullet damage* dan *survivability*.

1. Fungsi utama

```
{ Prosedur utama yang selalu dipanggil saat bot masih hidup (hanya representasi) }
procedure Main()
  if (isScanned) then
    ... { Laksanakan mekanisme locking }
  else
    SetTurnRadar(Positive.Infinity)
    ... { Memutar radar ke satu arah }
```

2. Mekanisme locking

```
{ prosedur yang mengatur perilaku bot ketika menyerang (hanya representatif) }
```

```

procedure LockingMechanism()
  isScanning ← true
  sudutPutar ← ... { Menghitung sudut putar untuk radar dan untuk bot }
  { hadapkan bot ke arah musuh dengan sedikit predictive }
  distance ← ... { jarak bot ke musuh } - saveDistance
  setForward(distance) { bisa bernilai negatif untuk mundur }
  if (... { jarak ideal }) then
    power ← getPower(distanceTo(e.x, e.y))
    Fire(power) ... { menembak sesuai power }

```

3. Fungsi untuk menentukan *power*

```

{ prosedur untuk mendapatkan power berdasarkan jarak }
procedure getPower(double distance) → float power
  if (distance > 180) then
    → 1.5 { jarak masih jauh, jangan terlalu greedy }
  else if (distance <= 180 and distance > 0) then
    → 2 { mulai perkuat peluru }
  else
    → 3 { jika bot menempel, tembak dengan kekuatan penuh }

```

- e. Fungsi Kelayakan : Fungsi kelayakan memeriksa validitas gerakan bot, sehingga cenderung mencakup batasan-batasan game. Berikut merupakan beberapa contohnya.
1. Fungsi yang mengatur *power* maksimal yang dapat digunakan untuk menembak. Pada permainan ini, *power* maksimum yang dapat digunakan adalah 3. Jika ada bot yang menembak dengan *power* lebih dari itu, *damage* yang diterima musuh akan sama dengan tembakan dengan *power* 3.
 2. Fungsi yang membatasi pergerakan bot setiap giliran. Contohnya pergerakan radar yang dibatasi 45 derajat. Oleh karena itu, apabila bot ingin memutar radar sejauh 360 derajat, diperlukan 4 kali giliran.
 3. Sistem koordinat, di mana $x = 0$ dan $y = 0$ terletak di bagian bawah kiri arena permainan, sehingga bot tidak dapat mencapai area kurang dari itu.
- f. Fungsi Objektif : Fungsi objektif merupakan fungsi-fungsi yang mendukung heuristik. Dalam konteks ini, fungsi objektif memaksimalkan poin tergantung dari cara bot menyerang. Made in China berfokus pada poin yang didapat dari menembak (*bullet damage points*) dan bertahan hidup (*survival points*).

3.1.2 Analisis Efisiensi dan Efektivitas: *First Enemy Scanned*

Made In China merupakan bot yang sangat efektif untuk permainan umum (permainan dengan anggota banyak bot). Bot ini dengan efektif membersihkan musuh satu persatu (terutama musuh yang tidak lincah). Untuk musuh lincah sekalipun, bot ini memiliki sistem prediksi bidik untuk memperkirakan pergerakan musuh (bukan *machine learning* karena sifatnya *fixed*). Dengan sistem yang cenderung menembak, Made In China seringkali menjadi *third party* dalam pertempuran, sehingga secara tidak langsung memanfaatkan orang lain yang sedang saling berperang. Selain itu, heuristik *first scanned bot* membuat Made In China bergerak secara efisien karena cenderung memilih bot terdekat, sehingga membutuhkan sedikit pergerakan. Dengan ini, kemungkinan bot ini tertembak dalam perjalanan semakin kecil.

Di samping kelebihan tersebut, Made In China bukanlah sebuah robot yang ideal untuk berperang satu lawan satu dengan robot yang sifatnya ramming. Mekanismenya untuk mundur membuatnya rentan terjebak di tembok dan membuatnya kehabisan energi akibat tabrakan. Oleh karena itu, dibutuhkan juga keberuntungan berupa *spawning point* yang strategis. Selain itu, bot ini juga kurang baik dalam menembak musuh yang gerakannya bolak balik (*predictive aiming* membuatnya menembak terlalu samping, padahal musuh bergerak ke arah yang berlainan setiap waktu). Meskipun begitu, bot ini memiliki performa yang stabil dalam pertempuran dan dapat diandalkan dalam pertandingan secara keseluruhan.

3.2 Alternatif *Greedy: Weakest Enemy* (Bot EDI)

EDI (Energy-Domination Instinct) merupakan sebuah bot yang didesain dengan heuristik penyerangan utama terhadap bot-bot terlemah. Pada Robocode Tank Royale, bot tidak hanya diberikan poin untuk komponen penyerangan *bullet damage* untuk penembakan bot musuh dan *ram damage* untuk penabrakan bot musuh, tapi juga *bullet damage bonus* dengan poin tambahan sebesar 20% dari *damage* untuk musuh yang terbunuh penembakan dan juga *ram damage bonus* dengan poin tambahan sebesar 30% dari *damage* untuk musuh yang terbunuh dengan *ramming*. Dengan memprioritaskan penyerangan pada bot musuh yang terlemah, probabilitas mendapatkan *bullet damage bonus* dan *ram damage bonus* meningkat secara signifikan.

EDI akan melakukan *scanning* 360 derajat untuk mendeteksi seluruh musuh yang terdapat di sekitarnya. Pada proses *scanning*, EDI juga akan menyimpan informasi posisi, jarak,

kecepatan, arah, serta energi masing-masing bot musuh dan memilih bot dengan tingkat energi terendah. Setelah EDI mendeteksi bot dengan energi terendah, EDI akan menyerang bot musuh dengan mekanisme pendukung seperti *locking*, *predictive firing*, dan *aggressive ramming* untuk mengamankan *bonus damage* pada bot tujuan.

3.2.1 Pemetaan Elemen-Elemen *Greedy: Weakest Enemy*

Berikut merupakan pemetaan seluruh elemen *greedy* bot EDI dengan alternatif *greedy weakest enemy*:

- a. Himpunan Kandidat: Himpunan kandidat implementasi algoritma *greedy* pada bot EDI meliputi seluruh kemungkinan musuh yang terdeteksi selama proses *radar scanning*.
- b. Himpunan Solusi: Himpunan solusi implementasi algoritma *greedy* pada bot EDI merupakan *subset* dari himpunan kandidat, yaitu seluruh musuh dengan energi terendah yang diserang EDI saat itu.
- c. Fungsi Solusi: Fungsi solusi implementasi algoritma *greedy* pada bot EDI merupakan kumpulan aksi terhadap elemen hasil seleksi berupa:

1. Mekanisme *locking*:

```
{ Prosedur untuk melakukan locking dan menembakkan }  
  
private void LockAndFire()  
... {Kumpulan aksi untuk melakukan locking dan menembak musuh dengan  
    lokasi berdasarkan prediksi dan kekuatan berdasarkan perkiraan }
```

2. Mekanisme dukungan *locking* dengan menembak musuh berdasarkan prediksi:

```
{ Fungsi yang mengembalikan koordinat posisi X dan Y penembakkan sebagai  
hasil prediksi  
  
private (double x, double y) GetPredictiveFirePosition(double enemyX, double  
enemyY, double speed, double directionDeg, double bulletSpeed) →  
(predictedX, predictedY)  
... { Kalkulasi predictedX dan predictedY berdasarkan rumus fisika }
```

3. Mekanisme dukungan *locking* dengan memilih kekuatan tembakan berdasarkan jarak dan tingkat energi musuh:

```
{ Fungsi yang mengembalikan tingkatan kekuatan tembakan berdasarkan jarak  
dan energi musuh }
```

```
private double GetBulletPower(double distance, double energy) → bulletPower
... { Kekuatan tembakan meningkat seiring semakin dekatnya dan rendahnya
energi
      musuh }
```

- d. Fungsi Seleksi: Fungsi seleksi implementasi algoritma *greedy* pada bot EDI memiliki kriteria seleksi energi musuh terendah.

```
{ Bot EDI akan selalu menjadikan musuh dengan energi yang lebih rendah
menjadi target }

if (e.Energy < lowestEnergy) then
... { Menyimpan informasi bot dengan energi yang lebih rendah }
```

- e. Fungsi Kelayakan: Fungsi kelayakan implementasi algoritma *greedy* pada bot EDI memastikan dapat tidaknya melakukan aksi penyerangan pada bot dengan energi terendah yang dengan memeriksa:

1. Tingkat *gun heat* dan energi internal bot EDI:

```
{ Memastikan tingkat gun heat dan energi berada pada tingkat yang sesuai
sebelum menembak }

if (GunHeat = 0 and Energy > 1) then
Fire(power)
```

2. Lokasi relatif bot EDI terhadap tembok (*wall*):

```
{ Bot EDI akan selalu menghindari tembok sebelum melakukan aksi lanjutan }

private void AvoidWall()
... {kumpulan aksi untuk mengecek lokasi relatif bot EDI terhadap tembok
dan menjauhi tembok sebelum melanjutkan aksi lanjutan}
```

3. Status tertembak oleh peluru bot musuh:

```
{ Bot EDI akan melakukan reaksi berupa gerakan random bila tertembak oleh
peluru musuh sebelum melanjutkan aksi lanjutan }

private override void OnHitByBullet(HitByBulletEvent e)
... { Kumpulan instruksi untuk kabur menjauhi peluru bot musuh }
```

- f. Fungsi Objektif: Fungsi objektif implementasi algoritma *greedy* pada bot EDI adalah memaksimalkan poin *bullet damage bonus* dan *ram damage bonus* untuk memaksimalkan poin secara keseluruhan dan meningkatkan probabilitas kemenangan.

3.2.2 Analisis Efisiensi dan Efektivitas: *Weakest Enemy*

Strategi *Weakest Enemy* pada bot EDI menunjukkan efisiensi dan efektivitas tinggi dalam mengeliminasi lawan secara selektif. Bot EDI yang berfokus menyerang musuh dengan energi paling rendah meningkatkan peluang kill untuk memperoleh poin tambahan. Mekanisme predictive firing juga memungkinkan bot untuk menembak dengan akurasi tinggi. Dengan *bullet power* yang juga dihitung berdasarkan jarak terhadap musuh, bot EDI unggul dalam efisiensi pengelolaan energi karena kompleksitas seleksi yang rendah, akurasi yang tinggi, serta *bullet power* yang disesuaikan. Bot EDI akan efektif ketika dihadapkan dengan kondisi musuh tidak padat, energi yang variatif, dengan pergerakan yang mudah diprediksi.

Meskipun demikian, strategi memiliki beberapa keterbatasan, seperti ketika mendapat tekanan dari musuh, bot kurang mampu beradaptasi karena hanya berfokus pada target utama. Bot juga berisiko terkena serangan dari musuh lain saat melakukan duel dengan target, khususnya ketika dikepung atau berada di sekitar tembok. Pola pergerakan yang minim variasi membuat bot mudah diprediksi, sehingga rawan terhadap kondisi bot yang terlalu padat ataupun dekat tembok, menjadi titik kelemahan strategi yang krusial.

3.3 Alternatif *Greedy: Concentrated Sector* (Bot Hari Styles)

Bot Hari Styles berfokus menyerang sebanyak banyaknya bot yang terdapat pada salah satu dari dua belas sektor, dan mencari area yang kosong jika terkena serangan. Bot melakukan pemindaian radar sejauh 360 derajat, dan membagi medan menjadi 12 sektor yang mencakup 30 derajat. Setelah menemukan sektor dengan bot paling banyak, Hari Styles akan:

1. Mengunci radar ke sektor tersebut,
2. Menggerakkan (gun) secara berosilasi agar dapat menjangkau seluruh area dalam sektor,
3. Menembak secara terus-menerus menggunakan tembakan paling kuat ($\text{Min}(3, \text{Energy})$),
4. Tidak berhenti menembak hingga seluruh bot dalam kluster habis atau tidak terdeteksi.

Bot akan menembak dan tidak bergerak selama menyerang, namun jika terkena tembakan atau bertabrakan, bot akan menjauh dengan bergerak ke tengah sektor yang sebelumnya terdeteksi memiliki jumlah musuh paling sedikit untuk mencari keamanan.

Strategi Greedy yang diimplementasikan oleh Bot Hari Styles memungkinkan bot mencari poin terbanyak melalui komponen utama berupa *bullet damage* dan juga *survival score*. Dengan bot yang melakukan penguncian terhadap suatu sektor dengan bot terbanyak, apabila kita memenuhi sektor tersebut dengan sebanyak banyaknya tembakan, dengan maksimal *bullet power*, kita dapat memiliki kesempatan besar untuk meraup poin dari *bullet damage*. Melakukan *locking* terhadap sektor tersebut, hingga seluruh bot telah mati ataupun tidak terdeteksi, kita mencoba untuk mendapatkan *Bullet Damage Bonus* dengan membunuh musuh di dalam kluster. Bot juga berlari ke sektor dengan minimal bot *count*, mencari keamanan sehingga mendapatkan *survival score* sebanyak banyaknya.

3.3.1 Pemetaan Elemen-Elemen Greedy: Concentrated Sector

Berikut merupakan pemetaan seluruh elemen *greedy* bot EDI dengan alternatif *greedy weakest enemy*:

- a. Himpunan Kandidat: Himpunan kandidat adalah himpunan pilihan aksi yang dapat dilakukan bot setiap turn seperti menggerakkan bot (maju, mundur, kiri, dan kanan), menggerakkan radar (scanning), dan menembak dengan power tertentu.
- b. Himpunan Solusi: Himpunan solusi berupa semua aksi yang dipilih untuk mencapai objektif, yaitu menjadi pemenang dengan poin tertinggi.
- c. Fungsi Solusi: Fungsi solusi Memeriksa apakah goal sudah dicapai, yaitu memeriksa status kemenangan dan memastikan bot masih hidup.

```
while isRunning do
... { iterasi }
```

- d. Fungsi Seleksi: Fungsi seleksi Hari Styles terbagi menjadi dua yaitu Fungsi Seleksi *bullet damage*, dan *survival point*, yaitu:

1. *Bullet Damage*

```
{ Bot Hari Styles mencari sektor 30 derajat dengan jumlah musuh terbanyak }

if (sector.count > maxCount) then
    Lock sector dan tembak
```

2. *Survival Point*

```

{ Bot Hari Styles mencari sektor 30 derajat dengan jumlah musuh terbanyak }

if (sector.count < minCount) then
    Lari ke sektor dengan kepadatan rendah

```

- e. Fungsi Kelayakan: Fungsi kelayakan implementasi algoritma *greedy* pada bot Hari Styles memastikan aksi yang dilakukan tidak melewati batasan game berupa:

1. *Bullet power* berdasarkan energi bot:

```

{ Memastikan energi yang digunakan untuk menembak tidak melebihi batasan
game, max bullet power = 3 }

if(Energy > 3){
    SetFire(3);
} else {
    Fire(Energy);
}

```

- f. Fungsi Objektif: Fungsi objektif strategi algoritma *greedy* bot Hari Styles memaksimalkan poin melalui *bullet damage* dan juga *survival score* untuk mencapai poin tertinggi.

3.3.2 Analisis Efisiensi dan Efektivitas: *Concentrated Sector*

Strategi *Concentrated Sector* memaksimalkan poin melalui bullet damage dengan menargetkan sektor terpadat berdasarkan hasil pemindaian. Efektivitas bot terletak pada kemampuannya melakukan penguncian sektor dengan kepadatan tertinggi, lalu menyapu sektor dengan tembakan berantai menggunakan kekuatan maksimal. Dari segi efisiensi, bot memiliki keunggulan karena minim pergerakan memaksimalkan penggunaan turn, dengan hanya bergerak ketika bot menerima serangan. Selain itu, bot tidak menggunakan tracking kompleks ataupun *predictive aiming*, hanya mengandalkan kemampuan radar, menunjukkan efisiensi strategi komputasional. Strategi *greedy* ini akan efektif ketika musuh berada dalam jarak dekat, serta pada suatu formasi yang rapat, sehingga peluang peluru mengenai musuh meningkat secara signifikan.

Namun, dengan algoritma bot yang memaksa penguncian sektor hingga tidak terdapat bot satupun didalamnya. Efektivitas dan efisiensi bot akan menurun drastis ketika hanya tersisa satu musuh dengan jarak yang jauh, ataupun jika musuh memiliki pergerakan acak. Dalam kondisi

seperti itu, mekanisme *locking* akan menjadi kurang optimal karena tembakan bot yang bertujuan untuk memenuhi satu sektor tidak akan mengenai musuh. Hal ini menyebabkan adanya pemborosan energi peluru, sehingga peluang poin yang dihasilkan cenderung kecil. Selain itu, dengan kondisi *locking*, bot juga mungkin kehilangan peluang untuk mendapatkan sektor yang terpadat selanjutnya karena banyak bot yang sudah tumbang.

3.4 Alternatif *Greedy: Survivorship* (Bot Steve)

Bot Steve dirancang dengan heuristik pertahanan, di mana bot Steve akan menerapkan aksi-aksi untuk bertahan selama mungkin pada arena pertandingan antarbot. Heuristik ini diterapkan untuk memaksimalkan komponen poin *survival score* yang memberikan 50 poin tambahan setiap terdapat bot musuh yang mati dan *last survival bonus* yang memberikan sebanyak 10 poin dikalikan banyaknya musuh yang berguguran. Pertahanan pada bot Steve tidak hanya diimplementasikan dengan mekanisme kabur, tapi juga penyerangan berupa penembakan pada kondisi-kondisi tertentu yang dianggap aman.

Bot Steve akan melakukan *radar scanning* untuk mendeteksi musuh-musuh di sekitarnya. Ketika musuh terdeteksi, bot Steve akan menjauh dari musuh yang dekat dan mendekat bila terdapat jauh yang musuh. Bila tidak terkena tembakan, bot Steve akan mendapatkan kesempatan untuk menembak musuh jika dan hanya jika musuh di sekitarnya memiliki tingkat energi yang rendah, jauh dari bot Steve, atau bergerak dengan lambat. Bot Steve juga dilengkapi dengan mekanisme seperti pergerakan yang *random* ketika terkena peluru bot musuh agar tidak terprediksi dan bot Steve akan selalu menghindari tembok agar tidak rawan tersudutkan oleh bot lawan sehingga arah kabut bot Steve tidak hanya mempertimbangkan posisi musuh, tapi juga lokasi relatif bot Steve dengan tembok.

3.4.1 Pemetaan Elemen-Elemen *Greedy: Survivorship*

Berikut merupakan pemetaan seluruh elemen *greedy* bot Steve dengan alternatif *greedy weakest enemy*:

- a. Himpunan Kandidat: Himpunan kandidat implementasi algoritma *greedy* pada bot Steve meliputi keputusan menghindari atau menyerang musuh berdasarkan informasi yang diperoleh proses *radar scanning*.

- b. Himpunan Solusi: Himpunan solusi implementasi algoritma *greedy* pada bot Steve merupakan *subset* dari himpunan kandidat, yaitu musuh yang layak diserang.
- c. Fungsi Solusi: Fungsi solusi implementasi algoritma *greedy* pada bot Steve merupakan kumpulan aksi terhadap elemen hasil seleksi berupa menembakkan peluru berdasarkan perhitungan prediksi posisi musuh untuk meningkatkan tingkat keakuratan dan pengaturan kekuatan tembakan berdasarkan jarak dan tingkat energi musuh.

```

if (isKillable || !isClose || isSlow) then
    { Mengatur kekuatan tembakan }
    double power = GetBulletPower(distance, energy)
    double bulletSpeed = 20 - 3 * power
    { Mendapatkan koordinat tembakan berdasarkan prediksi }
    var (predictedX, predictedY) = GetPredictiveFirePosition(e.X, e.Y,
speed, enemyDirection, bulletSpeed)
    double turretAngle = NormalizeRelativeAngle(Direction +
BearingTo(predictedX, predictedY) - GunDirection)
    { Menyesuaikan arah tembakan dan tembak musuh }
    SetTurnGunLeft(turretAngle)
    Fire(power)

```

- d. Fungsi Seleksi: Fungsi seleksi implementasi algoritma *greedy* pada bot Steve memutuskan apakah musuh layak untuk dihindari atau diserang.

```

{ Bot Steve akan selalu menyesuaikan posisi sebagai respon terhadap
pergerakan musuh yang di-scan }

if (distance > 500) then
    ... { Mendekati musuh untuk menghindari flank dari bot lainnya }
else if (distance < 300) then
    ... { Menjauhi musuh untuk menjaga jarak dan mengurangi kemungkinan
        diserang }

{ Bot Steve tidak akan menyerang bila baru saja tertembak }

if (hasBeenHit) then
    ...
    return

{ Bot Steve hanya akan menembak pada kondisi tertentu }

if (isKillable || !isClose || isSlow) then
    ... { Mekanisme penembakan }

```


- e. Fungsi Kelayakan: Fungsi kelayakan implementasi algoritma *greedy* pada bot Steve memastikan untuk melakukan aksi penyerangan hanya ketika aman dengan memeriksa:

1. Apakah bot Steve baru saja ditembak:

```
if (hasBeenHit) then  
... { tidak melakukan penyerangan dan me-reset status hasBeenHit }
```

2. Lokasi relatif bot Steve terhadap tembok (*wall*):

```
{ Bot Steve akan selalu menghindari tembok sebelum melakukan aksi lanjutan }  
  
private void AvoidWall()  
... { kumpulan aksi untuk mengecek lokasi relatif bot Steve terhadap  
tembok  
dan menjauhi tembok agar tidak rawan disudutkan musuh }
```

3. Lokasi relatif musuh terhadap musuh sekitar:

```
{ Bot Steve akan melakukan reaksi berupa gerakan random bila tertembak oleh  
peluru musuh sebelum melanjutkan aksi lanjutan }  
  
private override void OnHitByBullet(HitByBulletEvent e)  
... { Kumpulan instruksi untuk bergerak secara random menghindari peluru  
musuh }
```

- f. Fungsi Objektif: Fungsi objektif implementasi algoritma *greedy* pada bot Steve adalah memaksimalkan poin *survival* dengan bertahan hidup selama mungkin dengan menghindari posisi berbahaya dan hanya menembak saat aman.

3.4.2 Analisis Efisiensi dan Efektivitas: *Survivorship*

Bot Steve yang dirancang berbasis *survivorship*, demi memaksimalkan poin survival score sangat efektif dalam menghadapi musuh secara defensif. Bot mampu secara aktif menghindari tembakan dan menjaga jarak dari musuh, serta secara adaptif berpindah arah ketika disudutkan. Respons bot secara cepat beradaptasi terhadap ancaman dengan menahan tembakan dan melakukan gerakan untuk menghindari potensi tembakan setelahnya. Dari segi efisiensi, Steve menunjukkan utilisasi peluru yang bijak dengan hanya menembak dalam kondisi tertentu, memberikan kontribusi besar terhadap penghematan energi. Gerakan dinamis menyebabkan bot juga menjadi susah diprediksi, mencegah tembakan akurat dari lawan. Steve akan efektif pada

kondisi pertandingan dengan banyak musuh agresif, dimana pendekatan defensif bot cocok untuk mempertahankan hidup dan meraih poin secara konsisten.

Dalam memaksimalkan poin survival, Bot Steve memiliki kekurangan inisiatif menyerang, karena hanya akan menembak jika kondisi sangat menguntungkan, meminimalisir poin *bullet damage*. Dengan juga tidak melakukan penguncian terhadap satu target, Steve juga bersifat lebih situasional dibanding ofensif. Efisiensi bot juga terkadang terganggu oleh siklus radar dan penembakan yang kurang optimal, tanpa adanya prioritas terhadap kondisi musuh. Steve yang kesulitan mengalahkan bot *locking*, akan kurang efektif apabila berada pada kondisi penuh dengan bot yang dapat mengunci pergerakan Steve secara efektif.

3.5 Strategi *Greedy* yang Diimplementasikan

Alternatif *Greedy* pada keempat bot yang telah dirancang memiliki kelebihan dan kekurangan masing - masing. Setelah melalui proses pertimbangan dan analisis kasus, strategi *greedy* yang dipilih sebagai strategi utama adalah strategi *First Scanned*. Berdasarkan tes pertandingan yang dilakukan, strategi *First Scanned*, dengan kemampuannya untuk berfokus pada aspek ofensif, sekaligus menjaga jarak aman dengan variasi bot musuh, konsisten memperoleh skor tertinggi berdasarkan *bullet damage* dan *survival point*.

Strategi *Weakest Enemy* tidak dipilih meskipun fokus penyerangan terhadap bot dengan energi paling rendah mampu meningkatkan peluang *bullet damage bonus*, ataupun *ram damage bonus*, karena pendekatan ini berisiko tinggi. Pada skenario dengan banyak bot di arena, bot EDI cenderung memasuki area yang padat dan berbahaya demi mengejar bot lemah. Selain itu, strategi *ramming* yang dilakukan pada strategi ini juga meningkatkan eksposur bot terhadap *bullet* yang ditembakkan musuh, serta menyulitkan proses escape ketika bot sudah diserang dari beberapa arah. Strategi *Weakest Enemy* yang berfokus cukup berat pada ofensif, tanpa banyaknya *spatial awareness* menyebabkan bot mungkin kehilangan nyawa ketika permainan berlangsung karena diserang dari berbagai arah.

Strategi *Concentrated Sector* tidak digunakan dengan pertimbangan bahwa bot memiliki kecenderungan untuk menghabiskan energi secara berlebihan. Bot akan melakukan spray tembakan ke arah kluster tanpa adanya pertimbangan efisiensi energi secara terus-menerus, menyebabkan bot menjadi lebih rentan terhadap musuh dengan pola pergerakan acak seperti

SpinBot ataupun *Crazy*. Pergerakan musuh yang tidak selalu berada dalam arah tembakan menyebabkan akurasi menurun, dan konsumsi energi menjadi tidak efektif.

Strategi *Survivorship* yang difokuskan untuk bertahan selama mungkin dan memaksimalkan survival point juga tidak dipilih, karena strategi ini kurang agresif dalam menghasilkan *bullet damage*. Bot hanya menembak dalam kondisi yang dianggap aman, seperti ketika energi musuh rendah ataupun pergerakan yang lambat, yang efektif untuk menghindari posisi berbahaya dan bertahan hingga akhir. Namun dalam praktiknya, bot ini akan kesulitan melawan bot dengan mekanisme *locking*, dan mampu aktif menyerang musuh dengan presisi. Sifat pasif bot untuk mendapatkan *bullet damage*, maupun mencari kemenangan pada ronde, menyebabkan peluang meraih poin tertinggi menjadi lebih kecil dibanding strategi lainnya.

Strategi *First Scanned* menggunakan pendekatan tracking dan tembak, dimana bot akan memindai medan, dan bot pertama yang ditemukan akan dikejar hingga berada pada jarak ideal. Bot kemudian akan menembak dengan kekuatan peluru yang disesuaikan dengan jarak musuh (1.5 - 3), selagi menjaga jarak aman. Strategi ini mampu mempertahankan keseimbangan antara menyerang secara agresif dan bertahan secara efisien. Meskipun terdapat beberapa kasus uji dimana strategi *Weakest Enemy* dapat mengungguli strategi *First Scanned* ketika jumlah bot sedikit, kelompok memperkirakan pertandingan sebenarnya akan melibatkan lebih banyak bot. Kemampuan strategi *First Scanned* untuk menjaga jarak, dan tetap menyerang secara maksimal menjadikan bot lebih fleksibel dan adaptif terhadap berbagai kondisi.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi

4.1.1 Alternatif 1: *Shooting the First Scanned Bot*

Secara umum, implementasi strategi *shooting the first scanned bot* terdiri atas program utama dengan fungsi-fungsi yang melakukan *overriding* seiring berjalannya program utama. Berikut merupakan implementasinya.

1. Program utama: berisi pengulangan yang berlangsung selama bot masih hidup dan permainan belum berakhir. Program utama memiliki parameter berupa boolean `isScanned` untuk sebagai penanda kondisi radar untuk menentukan aksi bagi bot.

```
public override void Run()
    while isRunning do
        if (isRammed == 1) then
            SetBack(100)
            SetTurnRight(90)
            isRammed = 0
        else
            if (isTracking == 0) then
                TurnRadarRight(double.PositiveInfinity)
            AvoidWalls() { Mencegah kasus bot menabrak tembok }
```

2. Kasus bot mendeteksi musuh.

```
public override void OnScannedBot(input ScannedBotEvent e, input/output
isTracking)
    isTracking ← 1 { menghentikan scan berputar }
    if (e.Energy < 0) then return
    { mencegah kasus energi musuh tepat nol, memerintahkan bot untuk menembak
    sekali lagi ketika energi musuh nol }

    angleToEnemy ← Direction + BearingTo(e.X, e.Y)
    radarTurnAngle ← NormalizeRelativeAngle(angleToEnemy - RadarDirection)
    { jauhnya radar harus berputar }
    toleranceAngle = Min(Atan(36.0 / DistanceTo(e.X, e.Y)), 45)
    setForward(DistanceTo(e.X, e.Y) - saveDistance)

    firePower ← GetFirePowerForDistance(DistanceTo(e.X, e.Y))
    if (DistanceTo(e.X, e.Y) ≤ idealDistance and e.Energy ≥ 0) then
        if (e.Speed < 2) then
            Fire(3)
        else
            Fire(firePower)
    lastEnemyHeading ← e.Direction
```

```
lastEnemyID = e.ScannedBotId
```

3. Kasus bot menabrak bot lain.

```
public override void OnHitBot(input/output isRammed, input HITBotEvent e)
    isRammed ← 1

{ prosedur kabur sendiri terdapat pada program utama, yaitu }
if (isRammed == 1) then
    SetBack(100)
    SetTurnRight(90)
    isRammed = 0
```

4. Kasus bot menabrak tembok.

```
public override void OnHitWall(HitWallEvent e)
    SetBack(100)
    SetTurnRight(90)
```

5. Implementasi fungsi-fungsi tambahan untuk membantu fungsi utama.

```
{ Prosedur untuk membelokkan bot menghadap koordinat (x,y) }
private void TurnToFaceTarget(input double x, input double y)
    var bearing = BearingTo(x, y)
    turnDirection = (bearing ≥ 0) ? 1 : -1
    SetTurnLeft(bearing)

{ Fungsi untuk mendapatkan kekuatan peluru berdasarkan jarak }
private double GetFirePowerForDistance(input double distance)
    if (distance > 180) then
        → 1.5
    else if (distance ≤ 180 and distance > 0) then
        → 2
    else
        → 3

{ Fungsi untuk memberikan (x,y) baru hasil prediksi berdasarkan gerakan
musuh dan kekuatan peluru }
private double PredictEnemyPositionCircular(input ScannedBotEvent e, double
bulletPower, output double x, double y)
    double bulletSpeed ← 20 - 3*bulletPower { rumus kecepatan peluru dalam
game }
    double enemyX ← e.X
    double enemyY ← e.Y
    double enemyHeadingRad ← e.Direction * (PI/180) { mengubah ke bentuk
radian }

    double headingChange ← 0
    if (lastEnemyId = e.ScannedBotId) then
        headingChange ← (e.Direction - lastEnemyHeading) * (PI/180)
```

```

    double predictedX ← enemyX
    double predictedY ← enemyY
    double predictedHeading ← enemyHeadingRad
    double time ← 0

    while (true) do
        double distance ← Sqrt(Pow(predictedX - X, 2) + Pow(predictedY - Y,
2))
        double travelTime ← distance / bulletSpeed
        if (time ≥ travelTime) then
            time ← time + 1
        → (predictedX, predictedY)

{ prosedur untuk menghindari tembok }
private void AvoidWalls()
    bool nearLeft ← X < wallMargin
    bool nearRight ← X > (ArenaWidth - wallMargin)
    bool nearTop ← Y > (ArenaHeight - wallMargin)
    bool nearBottom ← Y < wallMargin

    if (nearLeft) then
        SetTurnRight(NormalizeRelativeAngle(45 - Direction))
        SetForward(100)
    else if (nearRight) then
        SetTurnRight(NormalizeRelativeAngle(135 - Direction))
        SetForward(100)
    else if (nearTop) then
        SetTurnRight(NormalizeRelativeAngle(225 - Direction))
        SetForward(100)
    else if (nearBottom) then
        SetTurnRight(NormalizeRelativeAngle(315 - Direction))
        SetForward(100)
{ Intinya membuat ia sejajar dengan garis x atau y, lalu maju }

```

4.1.1 Alternatif 2: *Weakest Enemy*

Alternatif *greedy weakest enemy* diimplementasikan dengan *looping* pada program utama yang dilengkapi dengan berbagai *event driven method* yang akan melakukan *override* sesuai dengan situasi yang didefinisikan. Alternatif ini juga memanfaatkan berbagai *method* tambahan untuk mempermudah implementasi strategi alternatif ini.

1. Program utama: inialisasi bot berupa konfigurasi warna fisik bot dan deklarasi *looping* bot berupa *scanning* secara kontinu untuk mencari bot tujuan dengan energi terendah:

```
public override void Run():
```

```

{ Konfigurasi warna fisik bot EDI }
BodyColor      ← Color.Black
BulletColor     ← Color.Black
RadarColor      ← Color.Black
ScanColor       ← Color.Black
TurretColor     ← Color.Black

{ Bot melakukan continuous scanning untuk mencari musuh dengan
energi
    terendah }
while (IsRunning) do
    hasTarget ← false
    lowestEnergy ← double.MaxValue

    { Bot melakukan rotasi 360 derajat secara terus menerus }
    if (RadarTurnRemaining = 0) then
        SetTurnRadarRight(double.PositiveInfinity)

    { Memastikan bot menghindari tembok }
    AvoidWall()
    Go()

    { Bot mengeksekusi mekanisme lock and fire setelah mendeteksi
musuh }
    if (hasTarget) then
        LockAndFire()

```

2. Kasus bot mendeteksi musuh:

```

public override void OnScannedBot(input ScannedBotEvent e):
    { Bot menyimpan informasi bot musuh baru ketika menemukan yang
    energinya lebih rendah }
    if (e.Energy < lowestEnergy) then
        lowestEnergy ← e.Energy
        targetEnergy ← e.Energy
        targetX ← e.X
        targetY ← e.Y
        targetDistance ← DistanceTo(e.X, e.Y)
        targetSpeed ← e.Speed

```

```
targetDirection ← e.Direction  
hasTarget ← true
```

3. Implementasi *method* tambahan LockAndFire():

```
{ Bot melakukan penembakan secara terarah (locking dan predictive) }  
private void LockAndFire():  
    { Memprediksi arah tembakan }  
    double power ← GetBulletPower(targetDistance, targetEnergy)  
    double bulletSpeed ← 20 - 3 * power  
    var (predictedX, predictedY) ← GetPredictiveFirePosition(targetX,  
targetY, targetSpeed, targetDirection, bulletSpeed)  
  
    { Mekanisme locking }  
    double angleToEnemy ← Direction + BearingTo(predictedX, predictedY)  
    double radarTurn ← NormalizeRelativeAngle(angleToEnemy -  
RadarDirection)  
    double extraTurn ← Math.Min(Math.Atan(36.0 / targetDistance), 45)  
    radarTurn ← radarTurn + (radarTurn < 0 ? -extraTurn : extraTurn)  
    SetTurnRadarLeft(radarTurn)  
  
    { Memutarkan senjata terhadap bot musuh }  
    double gun ← NormalizeRelativeAngle(angleToEnemy - GunDirection)  
    SetTurnGunLeft(gun)  
  
    { Bot mendekati bot musuh }  
    TurnTowardsTarget(predictedX, predictedY)  
    SetForward(DistanceTo(predictedX, predictedY) + 2)  
  
    { Menembak bila kondisi memenuhi }  
    if (GunHeat = 0 && Energy > 1) then  
        Fire(power)
```

4. Kasus bot menabrak bot musuh:

```
{ Bot akan meneruskan serangan secara agresif setelah menembak }  
public override void OnHitBot(input HitBotEvent e):  
    Fire(3)  
    SetForward(40)
```


5. Kasus bot menabrak tembok:

```
{ Bot cukup mundur dan menjauhi tembok dan kembali melakukan scanning dan  
locking pada bot musuh di sekitarnya }  
public override void OnHitWall(input HitWallEvent e)  
    SetBack(50)
```

6. Kasus bot tertembak peluru musuh

```
{ Bot melakukan pergerakan random untuk menghindari musuh yang melakukan  
locking atau predictive firing }  
public override void OnHitByBullet(input HitByBulletEvent e):  
    int turnAngle ← random.Next(-90, 91)  
    int distance ← random.Next(50, 151)  
    SetTurnRight(turnAngle)  
    if (random.Next(0, 2) = 0) then  
        SetForward(distance)  
    else  
        SetBack(distance)
```

7. Implementasi *method* tambahan GetPredictiveFirePosition():

```
{ Mengembalikan koordinat X dan Y tembakan yang telah diprediksi menggunakan  
perhitungan fisika sederhana }  
private GetPredictiveFirePosition(double enemyX, double enemyY, double speed,  
double directionDeg, double bulletSpeed) → (double, double):  
    double directionRad ← directionDeg * (Math.PI / 180)  
    double predictedX ← enemyX  
    double predictedY ← enemyY  
    double distance ← DistanceTo(enemyX, enemyY)  
    double time ← distance / bulletSpeed  
  
    predictedX ← predictedX (speed * time * Math.Cos(directionRad))  
    predictedY ← predictedY + (speed * time * Math.Sin(directionRad))  
  
    → (predictedX, predictedY)
```

8. Implementasi *method* tambahan GetBulletPower():

```

{ Mengembalikan tingkatan kekuatan peluru berdasarkan jarak dan tingkat energi bot musuh }
private double GetBulletPower(double distance, double energy) → int:
    if ((energy < 20 and distance < 300) or (distance < 200)) then
        → 3
    else if (distance < 500) then
        → 2
    else
        → 1

```

9. Implementasi *method* tambahan TurnTowardsTarget():

```

{ Memutar bot dengan arah menuju bot musuh }
private void TurnTowardsTarget(double x, double y):
    var bearing ← BearingTo(x, y)
    SetTurnLeft(bearing)

```

10. Implementasi *method* tambahan AvoidWall():

```

{ Menjauhkan bot dari tembok bila sudah berada mendekatnya }
private void AvoidWall():
    const double margin ← 10
    const double fieldWidth ← 800
    const double fieldHeight ← 600
    double x ← X
    double y ← Y

    if (x < margin or x > fieldWidth - margin or y < margin or y >
fieldHeight - margin) then
        SetTurnRight(180)
        SetBack(50)

```

4.1.1 Alternatif 3: *Concentrated Sector*

Alternatif *greedy Concentrated Sector* diimplementasikan dengan *looping* pada program utama yang dilengkapi dengan berbagai *event driven method* yang akan melakukan *override*

sesuai dengan situasi yang didefinisikan. Alternatif ini juga memanfaatkan berbagai *method* tambahan untuk mempermudah implementasi strategi alternatif ini.

1. Program utama: inisialisasi bot berupa konfigurasi warna fisik bot dan deklarasi *looping* bot berupa *scanning* secara kontinu untuk mencari sektor tujuan dengan tingkat kepadatan bot tertinggi:

```
public override void Run()
    {Konfigurasi Warna Bot Hari Styles}
    BodyColor ← Color.Pink
    TurretColor ← Color.HotPink
    RadarColor ← Color.DeepPink
    ScanColor ← Color.Fuchsia

    {Melepas pergerakan bersama masing masing komponen}
    AdjustRadarForGunTurn ← true
    AdjustRadarForBodyTurn ← true
    AdjustGunForBodyTurn ← true

    {Sweep Radar 360 Derajat Initial}
    SetTurnRadarRight(double.PositiveInfinity)

    while (IsRunning) do
        {Jika bot belum melakukan locking}
        if (not hasCluster) then
            {Melakukan Sweep ulang 360 derajat}
            totalRadarSweep ← totalRadarSweep +
Abs(NormalizeAngle(RadarDirection - lastRadarDirection))
            lastRadarDirection ← RadarDirection

            {Jika sudah menyelesaikan sweep}
            if (totalRadarSweep ≥ 360) then
                LockOn()
        else
            SprayAndPray()
            ticksSinceLastScan ← ticksSinceLastScan + 1
            if (ticksSinceLastScan > 3) then
                ResetScans()

    Go()
```

2. Kasus bot mendeteksi musuh:

```
public override void OnScannedBot(input ScannedBotEvent e)
    double absAngle ← NormalizeAngle(Direction + BearingTo(e.X, e.Y))
    int sector ← (int)((absAngle + 180) / SectorAngle) % SectorCount

    if (not hasCluster) then
        {Melakukan Listing Sektor untuk Count Bot}
        if (!sectors.ContainsKey(sector)) then
            sectors[sector] ← new List<(double x, double y)>()

        sectors[sector].Add((e.X, e.Y))
    else
        {Locking hingga tidak ada bot lagi yang terdeteksi radar}
        if (DistanceTo(e.X, e.Y) < 1200) then
            ticksSinceLastScan ← 0
```

3. Implementasi *method* tambahan LockOn():

```
private void LockOn():
    {Mencari sektor dengan count bot terbanyak}
    int maxCount ← 0
    iterate [var kvp in sectors]:
        if (kvp.Value.Count > maxCount) then
            maxCount ← kvp.Value.Count
            targetSector ← kvp.Key

    {Mencari angle ke tengah sektor}
    targetAngle ← -180 + SectorAngle * targetSector + SectorAngle / 2
    hasCluster ← true
```

4. Implementasi *method* tambahan SprayAndPray():

```
private void SprayAndPray():
    {Memindahkan radar ke tengah sektor}
    double radarTarget ← targetAngle + (sweepRight ? SectorAngle / 2 :
    -SectorAngle / 2)
    SetTurnRadarLeft(NormalizeAngle(radarTarget - RadarDirection))
    sweepRight ← not sweepRight
```

```

        {Melakukan tembakan berantai dengan osilasi dan jitter tembakan}
        gunJitterOffset ← gunJitterOffset + gunJitterDirection *
GunJitterStep
        if (Abs(gunJitterOffset) > GunJitterMax) then
            gunJitterDirection ← gunJitterDirection * -1

        double gunAngle ← targetAngle + gunJitterOffset
        double gunTurn ← NormalizeAngle(gunAngle - GunDirection)
        SetTurnGunLeft(gunTurn)

        {Menembak sekuat tenaga antara 3 atau energy ketika sudah < 3}
        if (Energy > 3) then
            SetFire(3)
        else
            Fire(Energy)

```

5. Implementasi *method* tambahan ResetScans():

```

{Reset semua atribut untuk melakukan scan ulang}
private void ResetScans():
    hasCluster ← false
    sectors.Clear()
    totalRadarSweep ← 0
    ticksSinceLastScan ← 0
    SetTurnRadarRight(double.PositiveInfinity)

```

6. Kasus bot tertembak peluru musuh:

```

{Jika Bot tertembak peluru musuh, berpindah ke sektor dengan tingkat
kepadatan terendah}
public override void OnHitByBullet(HitByBulletEvent e):
    MoveToLeastPopulatedSector()

```

7. Kasus bot menembak peluru dan mengenai bot musuh:

```

{Jika Bot tertabrak musuh, berpindah ke sektor dengan tingkat kepadatan
terendah}
public override void OnHitBot(HitBotEvent e):

```

```
MoveToLeastPopulatedSector()
```

8. Implementasi *method* tambahan MoveToLeastPopulatedSector():

```
private void MoveToLeastPopulatedSector():  
    {Mencari sektor dengan count minimal}  
    int minCount ← 9999  
    int leastPopulatedSector ← -1  
    iterate [var kvp in sectors]:  
        if (kvp.Value.Count < minCount) then  
            minCount ← kvp.Value.Count  
            leastPopulatedSector ← kvp.Key  
  
    {Menggerakkan bot ke sektor dengan tingkat kepadatan rendah}  
    if (leastPopulatedSector ≠ -1) then  
        double targetAngle ← -180 + SectorAngle * leastPopulatedSector +  
SectorAngle / 2  
        double moveAngle ← NormalizeAngle(targetAngle - Direction)  
        SetTurnRight(moveAngle)  
        {Berjalan sejauh 80 pixel}  
        SetForward(80)
```

9. Implementasi *method* tambahan NormalizeAngle():

```
{Normalisasi sudut untuk turn yang sesuai}  
private NormalizeAngle(double angle) → double:  
    while (angle > 180) do  
        angle ← angle - 360  
    while (angle < -180) do  
        angle ← angle + 360  
    → angle
```

4.1.1 Alternatif 4: *Survivorship*

Alternatif *greedy survivorship* diimplementasikan dengan *looping* pada program utama yang dilengkapi dengan berbagai *event driven method* yang akan melakukan *override* sesuai

dengan situasi yang didefinisikan. Alternatif ini juga memanfaatkan berbagai *method* tambahan untuk mempermudah implementasi strategi alternatif ini.

1. Program utama: inialisasi bot berupa konfigurasi warna fisik bot dan deklarasi *looping* bot berupa *scanning* secara kontinu untuk mencari bot tujuan dengan energi terendah:

```
public override void Run()
{
    Konfigurasi warna fisik bot Steve }
    BodyColor      ← Color.Red
    BulletColor     ← Color.Red
    RadarColor      ← Color.Red
    ScanColor       ← Color.Red
    TurretColor     ← Color.Red

    { Bot melakukan scanning sebesar 360 derajat untuk mendeteksi musuh
      di sekitarnya }
    if (RadarTurnRemaining = 0) then
        SetTurnRadarRight(Double.PositiveInfinity)
    AvoidWall()
    Go()
```

2. Kasus bot mendeteksi musuh:

```
public override void OnScannedBot(input ScannedBotEvent e):
{
    Kondisi internal bot }
    double distance ← DistanceTo(e.X, e.Y)
    double energy   ← e.Energy
    double speed    ← e.Speed
    double enemyDirection ← e.Direction
    double bearing  ← BearingTo(e.X, e.Y)

    [ Kondisi eksternal bot musuh ]
    bool isKillable ← energy < 20
    bool isClose    ← distance < 400
    bool isSlow     ← Math.Abs(speed) < 2

    { Bot mendekati bot musuh yang jauh untuk menghindari bot lainnya
      yang berada dekat dan agar tidak terkena flank }
    if (distance > 500) then
```

```

        SetTurnLeft(bearing)
        SetForward(50)

    { Bot menjauhi bot musuh yang mendekatinya atau dekat dengannya }
    else if (distance < 300) then
        double escapeAngle ← GetEscapeAngle(e.X, e.Y)
        SetTurnLeft(NormalizeRelativeAngle(escapeAngle - Direction))
        SetForward(100)

    { Bot tidak akan menembak bila baru saja tertembak }
    if (hasBeenHit) then
        hasBeenHit ← false
        → nothing

    if (isKillable or not isClose or isSlow) then
        double power ← GetBulletPower(distance, energy)
        double bulletSpeed ← 20 - 3 * power
        var (predictedX, predictedY) ← GetPredictiveFirePosition(e.X,
e.Y, speed, enemyDirection, bulletSpeed)
        double turretAngle ← NormalizeRelativeAngle(Direction +
BearingTo(predictedX, predictedY) - GunDirection)
        SetTurnGunLeft(turretAngle)
        Fire(power)

```

3. Kasus bot menabrak tembok:

```

{ Bot akan menjauhi tembok bila menabraknya }
public override void OnHitWall(input HitWallEvent e):
    bool nearLeft ← X < 100
    bool nearRight ← X > 700
    bool nearTop ← Y > 500
    bool nearBottom ← Y < 100

    double escapeAngle ← 0

    { Bot akan bergerak ke arah berlawanan dari sisi tembok tertabrak }
    if (nearLeft) then
        escapeAngle ← 0
    else if (nearRight) then

```



```

        escapeAngle ← 180
    else if (nearBottom) then
        escapeAngle ← 90
    else if (nearTop) then
        escapeAngle ← 270

    SetTurnRight(escapeAngle)
    SetForward(100)

```

4. Kasus bot tertembak peluru musuh

```

{ Bot akan melarikan diri ketika tertembak peluru musuh ke arah yang
berlawanan dengan tembok dengan sudut yang random agar tidak terpojoki oleh
musuh }
public override void OnHitByBullet(input HitByBulletEvent e):
    hasBeenHit ← true
    int turnAngle

    const int safeDistance ← 50
    bool nearLeft ← X < safeDistance
    bool nearRight ← X > 800 - safeDistance
    bool nearTop ← Y > 600 - safeDistance
    bool nearBottom ← Y < safeDistance

    if (nearLeft) then
        turnAngle ← random.Next(0, 90)
    else if (nearRight) then
        turnAngle ← random.Next(90, 180)
    else if (nearTop) then
        turnAngle ← random.Next(180, 270)
    else if (nearBottom) then
        turnAngle ← random.Next(270, 360)
    else
        turnAngle ← random.Next(-90, 91)

    SetTurnRight(turnAngle)

    { Kabur lebih jauh seiring semakin agresifnya musuh }
    double distance ← 30 * e.Damage

```

```

{ Arah pergerakan yang random }
if (random.Next(0, 2) = 0) then
    SetForward(distance)
else
    SetBack(distance)

```

5. Implementasi *method* tambahan GetPredictiveFirePosition():

```

{ Mengembalikan koordinat X dan Y tembakan yang telah diprediksi menggunakan
  perhitungan fisika sederhana }
private GetPredictiveFirePosition(double enemyX, double enemyY, double speed,
double directionDeg, double bulletSpeed) → (double, double):
    double directionRad ← directionDeg * (Math.PI / 180)
    double predictedX ← enemyX
    double predictedY ← enemyY
    double distance ← DistanceTo(enemyX, enemyY)
    double time ← distance / bulletSpeed

    predictedX ← predictedX (speed * time * Math.Cos(directionRad))
    predictedY ← predictedY + (speed * time * Math.Sin(directionRad))

    → (predictedX, predictedY)

```

6. Implementasi *method* tambahan GetBulletPower():

```

private GetBulletPower(double distance, double enemyEnergy) → double:
    if (enemyEnergy < 20) then
        → 3
    else if (distance < 200) then
        → 2
    else if (distance < 500) then
        → 1.5
    else
        → 1

```

7. Implementasi *method* tambahan GetEscapeAngle():

```

{ Mengembalikan sudut kabur berdasarkan arah pergerakan musuh dan letak tembok

```

```

agar tidak terpojoki musuh }
private GetEscapeAngle(double enemyX, double enemyY) → double:
    double angleToEnemy ← BearingTo(enemyX, enemyY)
    double escapeAngle ← NormalizeRelativeAngle(Direction + angleToEnemy
+ 180)

    bool nearLeft ← X < 100
    bool nearRight ← X > 700
    bool nearTop ← Y > 500
    bool nearBottom ← Y < 100

    { Bot selalu berusaha bergerak menjauhi musuh sekaligus tembok }
    if (nearLeft and (escapeAngle >= 90 and escapeAngle <= 270)) then
        escapeAngle ← 0
    else if (nearRight and (escapeAngle <= 90 or escapeAngle >= 270)) then
        escapeAngle ← 180
    else if (nearBottom and (escapeAngle >= 180 and escapeAngle <= 360))
then
        escapeAngle ← 90
    else if (nearTop and (escapeAngle <= 180)) then
        escapeAngle ← 270

    → escapeAngle

```

8. Implementasi *method* tambahan AvoidWall():

```

{ Menjauhkan bot dari tembok bila sudah berada mendekatinya }
private void AvoidWall():
    const double margin ← 10
    const double fieldWidth ← 800
    const double fieldHeight ← 600
    double x ← X
    double y ← Y

    if (x < margin or x > fieldWidth - margin or y < margin or y >
fieldHeight - margin) then
        SetTurnRight(180)
        SetBack(50)

```

4.2 Struktur Data Program

Dari keempat alternatif solusi *greedy*, dipilih satu alternatif yang paling stabil dan mampu bersaing di segala kondisi pertempuran, yaitu strategi *Shooting the First Scanned Bot* (Made In China). Berikut merupakan struktur data, fungsi, dan prosedur yang mendukung.

Tabel 5. Penjelasan Struktur Data

No.	Struktur Data	Penjelasan
1.	<u>double</u> safeDistance	Jarak aman yang bernilai 100, digunakan sebagai patokan bagi bot untuk maju dan mundur sesuai jarak aman dengan musuh. Bot mendekati musuh, namun tidak menabraknya (menyisakan jarak sejauh safeDistance).
2.	<u>double</u> idealDistance	Jarak ideal yang bernilai 200 yang menjadi jarak maksimum bagi bot untuk menembak musuh. Dengan ini, bot tidak akan membuang-buang energi untuk tembakan yang terlalu jauh.
3.	<u>int</u> isTracking	Diinterpretasikan sebagai boolean. Digunakan sebagai penanda bahwa bot telah berhasil memindai musuh, sehingga mematikan putaran radar pada program utama dan memberikan tanggung jawab kontrol ke prosedur OnScannedBot(ScannedBotEvent e).
4.	<u>int</u> turnDirection	Dapat bernilai 1 dan -1 sebagai penentu apakah bot akan berbelok ke kiri atau kanan.
5.	<u>double</u> angleToEnemy	Diisi dengan penjumlahan <i>direction</i> bot saat ini dengan selisih sudut antara bot dengan musuh. Nantinya akan digunakan untuk menentukan sudut perputaran radar (melakukan <i>aim tracking</i>).
6.	<u>double</u> toleranceAngle	Diisi dengan sudut tambahan yang perlu ditambahkan ke radarTurnAngle.
7.	<u>double</u> radarTurnAngle	Variabel yang menjadi patokan sudut perputaran radar pada OnScannedBot.
8.	<u>double</u> lastEnemyHeading	Menyimpan arah pergerakan terakhir musuh yang nantinya digunakan dalam <i>Predictive Aim</i> .
9.	<u>double</u> lastEnemyId	Menyimpan ID musuh yang terakhir dipindai untuk memastikan bahwa penargetan cenderung diarahkan pada musuh yang sama.
10.	<u>double</u> wallMargin	Batas aman kedekatan bot dengan tembok yang bernilai

		100. Digunakan untuk memastikan bot tidak mati karena menabrak tembok.
--	--	--

Tabel 6. Penjelasan Fungsi dan Prosedur

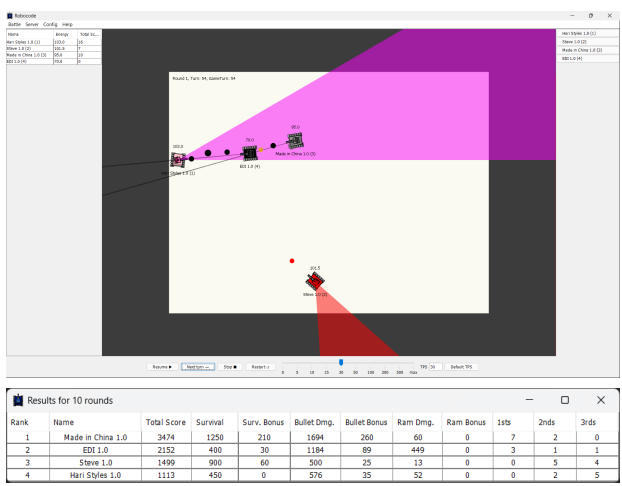
No.	Struktur Data	Penjelasan
1.	<code>public override void Run()</code>	Menjadi fungsi utama yang berjalan selama bot masih aktif. Terdapat <i>looping</i> utama yaitu <code>while isRunning</code> yang mencakup aksi tergantung dari kondisi bot. <ol style="list-style-type: none"> 1. Ketika bot ditabrak, mundur dan belok kanan. 2. Ketika bot tidak memindai musuh, lakukan pemindaian memutar. 3. Ketika bot memindai musuh (tunggu <i>override</i> dari <code>OnScannedBot</code>). 4. Hindari tembok setiap waktu.
2.	<code>public override void OnScannedBot(ScannedBotEvent e)</code>	Fungsi yang mengaktifkan mode tracking (<code>isTracking = 1</code>), mengarahkan radar ke musuh dengan sedikit angle tambahan, memanggil fungsi untuk prediksi <i>aiming</i> dan meminta bot menghadap arah yang sudah diprediksi, meminta bot untuk mendekat ke arah musuh jika jarak masih jauh (namun tidak terlalu dekat), dan menembak bot sesuai jarak dan kecepatan musuh saat ini. Setelah semua proses selesai, prosedur menyimpan arah terakhir pergerakan musuh beserta ID musuh tersebut.
3.	<code>public override void OnHitBot(HitBotEvent e)</code>	Mengaktifkan mode <code>isRammed</code> (mengubah sifat program utama selama bot masih ditabrak).
4.	<code>public override void OnHitWall(HitWallEvent e)</code>	Ketika menabrak tembok (jika masih menabrak), mundur dan belok kanan.
5.	<code>private void TurnToFaceTarget(double x, double y)</code>	Menghitung selisih sudut antara bot dengan musuh, menentukan arah perputaran (kanan atau kiri, mana yang lebih dekat), dan meminta bot berbelok ke arah tersebut.
6.	<code>private double GetFirePowerForDistance(double distance)</code>	Mengembalikan 1.5 jika jarak bot musuh lebih dari 180, 2 jika jarak bot antara 0 sampai 180, dan 3 jika bot menempel dengan

		bot lain.
7.	<code>private (double x, double y) PredictEnemyPosition (ScannedBotEvent e, double bulletPower)</code>	Berfungsi untuk memperkirakan koordinat musuh saat bergerak sirkular. Ditandai dengan sebuah loop while true yang memperbarui posisi musuh setiap tick. True di sini berarti waktu = waktu tempuh peluru, dengan waktu yang menyamai waktu tempuh peluru, dapat dipastikan peluru mengenai musuh (jika musuh tidak tiba-tiba berganti arah).
8.	<code>private void AvoidWalls()</code>	Memeriksa apakah bot dekat dengan tembok kiri, kanan, atas, atau bawah. Jika mendekati salah satunya sampai kurang dari jarak aman, kebur dengan sudut 45 derajat dari tembok tersebut.

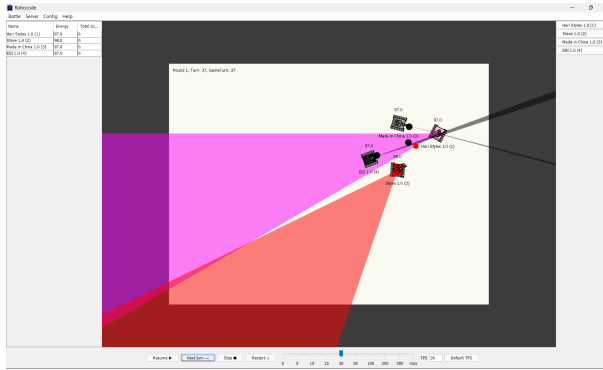
4.3 Analisis dan Pengujian

Pengujian efektivitas algoritma *greedy* dilakukan sebanyak 3 kali yang berupa pertandingan antara 4 bot yang telah dibuat berdasarkan algoritma *First Scanned*, *Weakest Enemy*, *Concentrated Sector*, *Survivorship*. Pengujian dilakukan dengan konfigurasi ukuran medan perang 800x600, dengan total turn berjumlah 5000. Pengujian yang dilakukan memberikan hasil sebagai berikut.

Tabel 7. Implementasi dan Pengujian

No.	Implementasi dan Pengujian	Penjelasan
1.	<div></div>	Hasil pengujian pertama menunjukkan Made In China memimpin poin dengan cukup jauh, diikuti oleh EDI, kemudian Steve dan juga Hari Styles. Tampak bahwa hanya Made In China dan EDI yang memenangkan pertandingan.

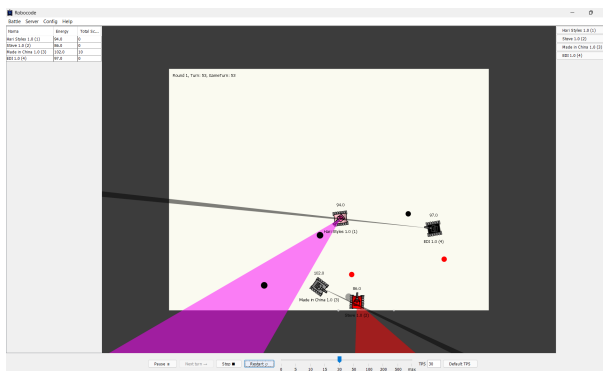
2.



Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Made In China 1.0	3142	1250	210	1480	200	1	0	7	2	0
2	EDI 1.0	3100	900	60	1322	125	566	126	3	7	0
3	Steve 1.0	818	500	30	242	12	34	0	0	1	5
4	Hari Styles 1.0	683	350	0	288	3	42	0	0	0	5

Hasil pengujian kedua menunjukkan skor yang cukup ketat antara Made In China dan juga EDI, dimana poin yang didapatkan EDI melalui poin damage nyaris mengalahkan Made In China. Posisi ketiga dan keempat kembali lagi dipegang oleh Steve dan Hari Styles.

3.



Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	EDI 1.0	3064	850	120	1318	116	481	179	5	3	0
2	Made In China 1.0	2751	1100	150	1269	153	79	0	4	5	0
3	Steve 1.0	931	650	0	262	1	18	0	0	2	6
4	Hari Styles 1.0	906	400	30	416	25	35	0	1	0	4

Pengujian ketiga menunjukkan bahwa EDI kini berada di posisi pertama dengan poin damage yang mampu mengalahkan total poin Made In China dengan poin survival terbesar. Posisi ketiga kembali diambil oleh Steve dengan poin survival 650, sedangkan posisi keempat diambil oleh Hari Styles dengan bullet damage 416.

Strategi *First Scanned* (Bot Made In China) mampu menunjukkan performa yang sangat optimal dalam perolehan skor secara keseluruhan. Strategi greedy berupa *locking* pada musuh pertama mampu menghasilkan poin *bullet damage* yang terbesar pada setiap pertandingan, tanpa adanya *ram damage* dari bot tersebut. Selain itu, strategi untuk menjaga jarak aman juga menunjukkan performa yang optimal dengan bot selalu memperoleh *survival* score yang paling tinggi di setiap pertandingannya. Dengan jumlah musuh yang relatif sedikit, bot mampu melakukan duel 1 lawan 1 dengan bot lain dan memenangkan pertandingan. Namun, pada game ketiga poin yang didapat oleh bot, terkalahkan secara minim oleh bot EDI, yang juga memiliki heuristik *locking* yang menyebabkan bot pada akhir pertandingan akan melakukan duel, dan pemenang ditentukan oleh energi bot yang lebih besar. Pada kebanyakan uji kasus, strategi *First Scanned*, mampu menunjukkan keunggulan terhadap *Weakest First* karena mampu mengalahkan

bot EDI ketika bot EDI mengincar musuh dengan energi yang lebih sedikit. Strategi *First Scanned* yang dilanjutkan dengan *locking* terhadap musuh terbukti mampu memberikan perolehan poin yang optimal dengan memaksimalkan poin *damage* dan juga *survival*.

Strategi *Weakest First* (Bot EDI) mampu menunjukkan performa yang juga optimal dalam memperoleh skor maksimal melalui komponen *damage*, baik itu *bullet damage* maupun *ram damage*. *Ram Damage* yang didapatkan oleh bot merupakan yang terbesar diantara strategi lainnya, bahkan mampu mendorong bot untuk memenangkan pertandingan, meskipun dengan *survival score* yang lebih rendah. Dengan total Bot yang relatif sedikit, bot seperti EDI yang mampu melakukan *locking* terhadap segala jenis musuh, mampu memenangkan pertandingan pada beberapa kasus melalui duel 1 lawan 1. Permasalahan terbesar yang terdapat pada strategi yang dimiliki bot EDI adalah kurangnya *survival score* yang disebabkan oleh bot melakukan ramming, serta serangan oleh bot lain ketika bot EDI sedang berfokus kepada musuh terlemah. Agresivitas yang dimiliki oleh bot EDI dengan strategi *Weakest First*, mampu menghasilkan keunggulan poin yang diraih melalui *bullet damage* dan *ram damage*, meskipun tanpa adanya skor *survival* yang tinggi, yang mungkin menghambat total perolehan poin oleh strategi ini.

Strategi *Concentrated Sector* (Bot Hari Styles) tampak tidak mencapai performa optimal strategi. Bot Hari Styles yang meraih penempatan posisi terakhir pada setiap match, gagal meraih poin maksimal karena strategi spray ke cluster hanya efektif jika terdapat banyak musuh dalam suatu sektor. Sedangkan dalam praktik pengujian, bot tidak selalu memiliki target yang cukup padat, menyebabkan efisiensi energi peluru menjadi buruk dengan adanya banyak peluru yang tidak mengenai target. Bot Hari Styles, pada pengujian ini mampu meraih poin *bullet damage* lebih besar daripada bot Steve, yang menunjukkan kompetensi bot untuk meraih poin *bullet damage* tinggi yang pada akhirnya membantu bot untuk mendekati perolehan poin strategi *Survivorship*. Strategi *Concentrated Sector* membutuhkan musuh yang terdapat relatif banyak, pada area yang sempit sehingga tembakan bot dapat lebih sering mengenai musuh.

Strategi *Survivorship* (Bot Steve) juga tidak mampu memperlihatkan performa optimal. Hal ini disebabkan oleh jenis bot yang menjadi musuhnya di pengujian ini, yaitu jenis bot dengan agresivitas tinggi, dengan mekanisme *locking* yang mampu mengikuti pergerakan bot secara aktif. Meskipun demikian, strategi ini mampu menunjukkan perolehan poin yang cukup kompeten, berada di rank 3 untuk setiap match, dengan perolehan poin *survival* yang tinggi,

mengalahkan beberapa bot lain pada pengujian. Permasalahan utama strategi *Survivorship* adalah kurangnya agresivitas bot untuk mendapatkan poin yang lebih tinggi pada beberapa komponen lainnya, menyebabkan bot bergantung pada bot lain untuk mengalahkan musuh. Selain itu, dengan adanya dua bot *locking* ini menyebabkan bot Steve lebih berpotensi untuk diserang oleh bot tersebut. Strategi *Survivorship* membutuhkan medan yang luas dengan musuh yang agresif satu dengan yang lain terlebih dahulu, dan tidak melakukan tracking terhadap Steve sehingga pergerakan Steve dapat memaksimalkan poin *survival* yang diperoleh, yang akan berkontribusi besar terhadap poin keseluruhan bot.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Tugas Besar Strategi Algoritma ini berhasil mengevaluasi dan membandingkan efektivitas dari empat strategi algoritma greedy yang diimplementasikan pada bot Robocode Tank Royale. Keempat strategi yang dibentuk berupa strategi *First Scanned*, *Weakest First*, *Concentrated Sector*, dan *Survivorship*. Setiap strategi *Greedy* yang dirancang telah berhasil diimplementasikan dengan baik pada bot yang terbentuk, memaksimalkan poin pada masing-masing komponen sesuai dengan strategi yang dipilih. Namun, setelah melewati proses pengujian, kelompok akhirnya memilih strategi *First Scanned* yang mampu memperoleh perolehan poin paling tinggi secara agregat.

Pengujian dilakukan dalam arena berukuran 800x600 selama 5000 turn untuk setiap match. Strategi *First Scanned* terbukti mampu mendominasi dua dari tiga pertandingan, karena mampu menjaga keseimbangan dengan mempertahankan posisi aman untuk memperoleh *survival point*, selagi tetap aktif menyerang secara efisien melalui sistem *tracking*. Strategi *Weakest First* unggul pada pertandingan ketiga dengan mengandalkan strategi *locking* serta *ramming*, namun cenderung sering kali lebih berisiko dalam kondisi ramai karena cenderung fokus pada target yang lemah, menyebabkan bot terkepung. Strategi *Concentrated Sector* unggul dalam *bullet damage* jika dihadapkan terhadap kelompok musuh yang padat, tetapi tidak efisien ketika jumlah musuh relatif sedikit dan bergerak secara acak, menyebabkan potensi pemborosan energi. Strategi *Survivorship* tampak stabil dalam mengumpulkan poin *survival*, namun kurangnya agresivitas bot, khususnya dalam kondisi 1 lawan 1 menyebabkan bot kurang mampu mengoptimalkan poin *damage*.

Berdasarkan keseluruhan pengujian dan analisis, strategi *Greedy Scanned First* terbukti sebagai strategi paling efektif dalam konteks kompetisi Robocode Tank Royale. Bot seringkali menunjukkan performa unggul dalam menjaga keseimbangan antara ofensif dan defensif, dengan adaptasi responsif terhadap kondisi pertempuran. Dengan demikian, strategi *Greedy Scanned First* adalah strategi yang paling optimal mengakumulasi poin secara efektif dan efisien.

5.2 Saran

Penulis menyarankan agar pada penyelenggaraan tugas besar selanjutnya, panitia atau penyusun tugas dapat lebih memperhatikan kompatibilitas sistem operasi, khususnya bagi pengguna macOS atau sistem non-Windows lainnya. Pada tugas besar ini, penulis menemui sejumlah kendala teknis saat menjalankan framework dan tools yang disediakan, terutama karena tidak semua dependensi dan perintah terminal langsung kompatibel dengan mac OS. Meskipun akhirnya ditemukan solusi alternatif, akan sangat membantu jika sejak awal disediakan panduan lintas platform agar seluruh peserta dapat memulai dengan kondisi teknis yang setara.

LAMPIRAN

a. Tabel :

No	Poin	Ya	Tidak
1	Bot dapat dijalankan pada Engine yang sudah dimodifikasi asisten.	✓	
2	Membuat 4 solusi greedy dengan heuristic yang berbeda.	✓	
3	Membuat laporan sesuai dengan spesifikasi.	✓	
4	Membuat video bonus dan diunggah pada Youtube.	✓	

b. Repository : https://github.com/MaheswaraKaindra/Tubes1_LastWORGHXHXX

c. Video : <https://youtu.be/93lQ7rZDfmk>

DAFTAR PUSTAKA

- Annisa. (2023). Algoritma Greedy: Pengertian, Jenis dan Contoh Program. [Online] Tersedia: <https://fikti.umsu.ac.id/algoritma-greedy-pengertian-jenis-dan-contoh-program/> [Diakses 19 Maret 2025].
- Munir, R. (2025). Algoritma *Greedy* (Bagian 1). [Online] Tersedia: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/04-Algoritma-Greedy-\(2025\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/04-Algoritma-Greedy-(2025)-Bag1.pdf) [Diakses 19 Maret 2025].
- Munir, R. (2025). Algoritma *Greedy* (Bagian 2). [Online] Tersedia: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/05-Algoritma-Greedy-\(2025\)-Bag2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/05-Algoritma-Greedy-(2025)-Bag2.pdf) [Diakses 19 Maret 2025].
- Munir, R. (2025). Algoritma *Greedy* (Bagian 3). [Online] Tersedia: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/06-Algoritma-Greedy-\(2025\)-Bag3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/06-Algoritma-Greedy-(2025)-Bag3.pdf) [Diakses 19 Maret 2025].
- Robocode Home. Robocode 1.9.5.0 API. [Online] Tersedia: <https://robocode.sourceforge.io/docs/robocode/> [Diakses 19 Maret 2025].