Laporan Tugas Besar 1
IF2211 Strategi Algoritma 2025/2026

Disusun oleh:

Kelas K1 - Kelompok 5 - Cisa

Varel Tiara	(13523008)
Bryan Ho	(13523029)
Ivan Wirawan	(13523046)

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

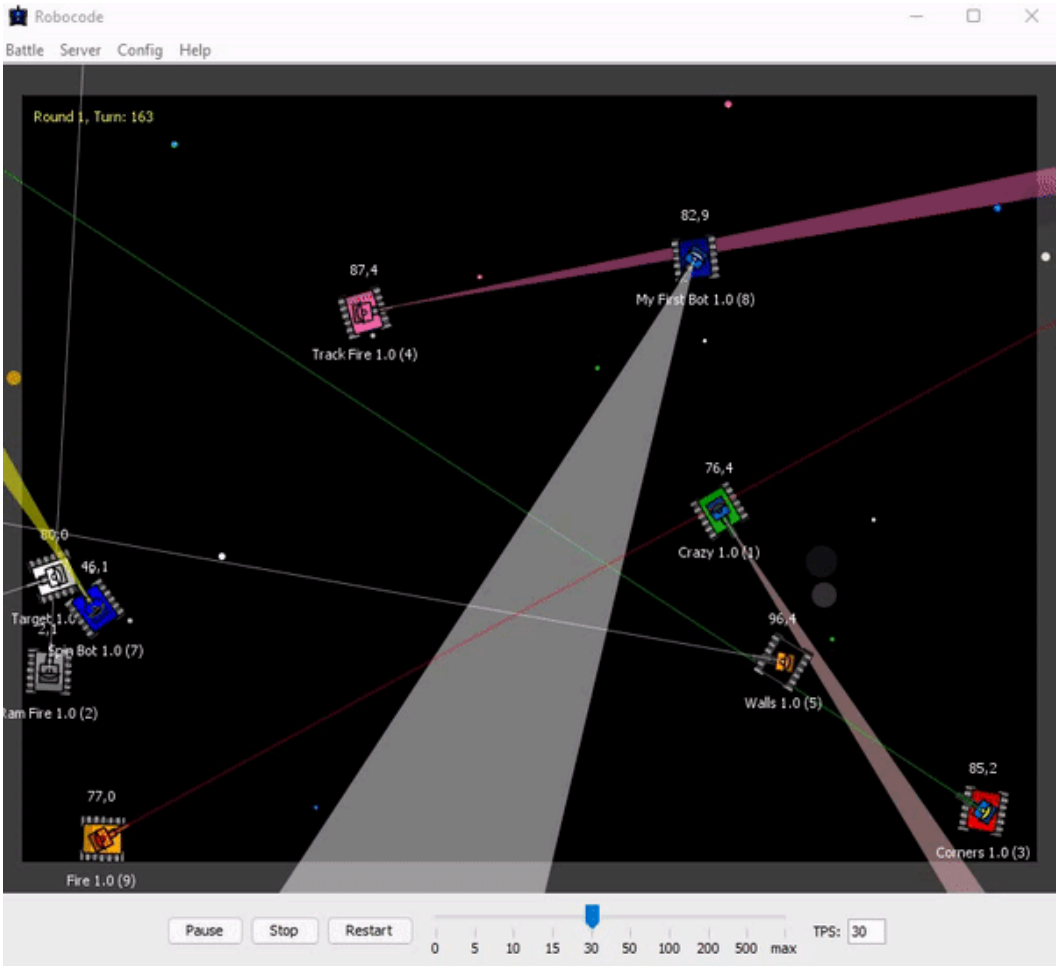
JL. GANESA 10, BANDUNG 40132

2025

DAFTAR ISI

BAB I

DESKRIPSI TUGAS



Gambar 1 Robocode Tank Royale

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, kami diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Tentunya kami sebagai mahasiswa harus menggunakan **strategi *greedy*** dalam membuat bot ini.

Komponen-komponen dari permainan ini antara lain:

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.

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 melewatkan turn tersebut. Jika bot melewatkan turn, ia tidak akan bisa menyesuaikan gerakannya atau menembakkan senjatanya karena server tidak menerima perintah tepat waktu sebelum turn berikutnya dimulai.

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.

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.

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.

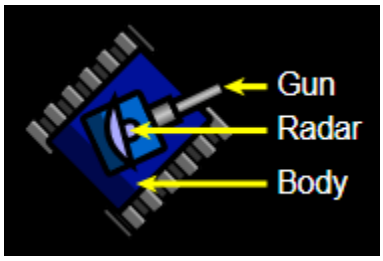
6. Tabrakan

   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.

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*.

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.

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.

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.

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

Algoritma greedy adalah algoritma yang memecahkan persoalan secara langkah per langkah (step by step) sedemikian sehingga, pada setiap langkah akan mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip "take what you can get now!") dan "berharap" bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global. Algoritma ini akan berusaha untuk mencapai solusi yang optimal dengan melakukan pemilihan yang paling rasional dan efektif pada setiap langkahnya.

2.2 Elemen Algoritma Greedy

Elemen-elemen yang terdapat pada algoritma greedy, antara lain:

1. Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap langkah.
2. Himpunan solusi, S : berisi kandidat yang sudah dipilih
3. Fungsi solusi : menentukan apakah himpunan yang dipilih sudah memberikan solusi
4. Fungsi seleksi (selection function): memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.
5. Fungsi kelayakan (feasible): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi.
6. Fungsi obyektif: fungsi untuk memaksimumkan atau meminimumkan

2.3 Game Engine Robocode Tank Royale

Game engine yang digunakan untuk menjalankan permainan dan mengimplementasikan bot ini sedikit berbeda dengan game engine default dari robocode tank royale. Diperlukan starter pack game robocode tank royale yang dapat diunduh melalui link https://github.com/Ariel-HS/tubes1-if2211-starter-pack/releases/tag/v1.0.

Pada link tersebut, terdapat game engine dan bot starter pack. Game engine berisi kode dari permainan yang berupa arena permainan dan API yang disediakan untuk berkomunikasi dengan *Graphical User Interface* (GUI) dan program bot, terdapat juga GUI permainan yang berfungsi untuk memvisualisasikan permainan. Sedangkan bot starter pack berisi sample bot yang dapat digunakan untuk memainkan permainan ini.

2.4 Alur Menjalankan Program (Memainkan Permainan)

Berikut langkah-langkah untuk menjalankan program.

1. Pastikan directory berada di file path "tubes1-if2211-starter-pack-1.0\tubes1-if2211-starter-pack-1.0" lalu jalankan GUI dengan mengetik "java -jar robocode-tankroyale-gui-0.30.0.jar" di terminal.
2. Lakukan set up config dengan cara klik tombol "Config −> Bot Root Directories". Kemudian masukkan folder yang berisi sample bots atau bot yang telah dibuat.
3. Jalankan battle dengan cara klik tombol "Battle −> Start Battle" akan muncul panel konfigurasi permainan. Boot bot yang ingin dimainkan dengan cara menekan tombol "Boot" pada bot yang ingin dimainkan dan tambahkan bot ke dalam arena dengan cara klik tombol "Add".
4. Mulai permainan dengan cara klik tombol "Start Battle".

2.5 Cara Menjalankan Bot

Berikut langkah-langkah untuk menjalankan bot.

1. Pastikan NET sudah terinstall di perangkat.
2. Ubah "Target Framework" pada file .csproj menjadi dotnet version yang sesuai dengan yang telah terpasang di perangkat.
3. Jalankan file .cmd pada folder bot dengan menggunakan command "cd <NamaBot>" dan "./<NamaBot>.cmd".
4. Boot ulang bot yang sudah dijalankan.

2.6 Aksi yang Dapat Dilakukan oleh Bot

Bot dalam Robocode Tank Royale beroperasi dalam lingkungan pertempuran berbasis event-driven, di mana setiap bot menerima informasi melalui berbagai event seperti deteksi musuh, tabrakan, atau tembakan yang mengenai target. Bot menggunakan sensor seperti radar untuk mendeteksi musuh, lalu mengambil tindakan berdasarkan informasi yang diperoleh, seperti menembak, bergerak, atau menghindar dari serangan lawan.

Terdapat beberapa komponen utama dalam bot, seperti radar untuk mendeteksi posisi musuh, gerakan untuk menghindari peluru atau mengejar musuh, dan senjata yang bisa melakukan serangan kepada musuh.

2.7 Implementasi Algoritma Greedy pada Bot

Algoritma greedy dalam bot diimplementasikan dengan membuat keputusan optimal pada setiap langkah berdasarkan kondisi saat ini tanpa mempertimbangkan langkah ke depan secara mendalam. Bot akan selalu memilih tindakan terbaik yang menghasilkan keuntungan langsung, seperti:

1. Menembak dengan Kekuatan Maksimal
    - Jika musuh berada dalam jarak dekat, bot akan menembak dengan daya maksimum untuk meningkatkan peluang mengenai target.
    - Jika musuh berada dalam jarak jauh, bot menembak dengan daya lebih kecil untuk menghemat energi.
2. Menghindari Serangan Secara Langsung
    - Saat mendeteksi penurunan energi musuh (indikasi tembakan), bot akan segera melakukan manuver menghindar tanpa mempertimbangkan pola historis tembakan lawan.
3. Mengutamakan Jarak Optimal
    - Jika bot terlalu jauh dari musuh, ia akan mendekat untuk meningkatkan akurasi tembakan.
    - Jika bot terlalu dekat, ia akan menjauh untuk menghindari risiko terkena serangan.
4. Menjauh dari Posisi Bot Musuh

- Bot akan mencari posisi yang paling aman di arena berdasarkan posisi Musuh.

5. Menabrak Bot Musuh
    - Bot akan menabrak bot lain karena skor yang dihasilkan relatif lebih tinggi dibandingkan ketika menembak.

Dengan demikian, implementasi algoritma greedy memastikan bahwa bot selalu mengambil tindakan terbaik secara lokal guna menghasilkan keuntungan langsung, baik dalam bentuk serangan maupun pertahanan, yang secara keseluruhan meningkatkan performa dalam pertempuran.

BAB III

APLIKASI STRATEGI *GREEDY*

3.1 Alternatif Greedy

3.1.1 Greedy by Bullet Damage - Ram Damage - Ram Damage Bonus

Algoritma ini bertujuan untuk mencari poin terbanyak melalui bullet damage, ram damage, dan ram damage bonus. Dalam peraturan permainan Robocode, bot akan mendapatkan poin sebesar kerusakan yang dilakukannya melalui tembakan yang tepat sasaran. Selain itu, apabila berhasil membunuh bot musuh, maka robot akan mendapatkan poin tambahan sebesar 20% dari kerusakan terakhir yang dibuat. Sedangkan poin yang dapat diraih oleh robot dari tabrakan adalah 2 kali nilai kerusakan yang dibuat. Robot juga akan mendapatkan bonus sebesar 30% dari kerusakan yang membunuh apabila berhasil memberikan serangan terakhir berupa tabrakan. Maka dari itu, algoritma *greedy* ini dirancang untuk mendapatkan skor terbesar melalui tembakan dan tabrakan.

3.1.1.1 Mapping Elemen Greedy

Terdapat beberapa elemen-elemen algoritma greedy, antara lain:

a. Himpunan Kandidat

Himpunan kandidat dari algoritma yang diterapkan pada sebuah program Robocode meliputi berbagai macam tindakan yang dapat diambil oleh robot setiap gilirannya. Jenis tindakannya meliputi pergerakan, penembakan, perputaran radar, dan banyak lainnya. Setiap giliran, robot dapat melakukan beberapa instruksi tindakan yang merupakan bagian dari algoritmanya.

b. Himpunan Solusi

Himpunan solusi merupakan daftar tindakan yang telah diambil oleh robot pada giliran yang telah lewat. Himpunan ini berisikan kandidat yang sudah dipilih melalui fungsi-fungsi yang telah diterapkan

untuk mendukung algoritma *greedy*. Kandidat yang dipilih adalah berdasarkan prioritas algoritma yang diterapkan. Dalam kasus ini, titik berat sumber poin utama adalah tembakan dan tabrakan.

c. Fungsi Solusi

Fungsi solusi adalah kriteria yang digunakan dalam menentukan apakah dari hasil kandidat yang dipilih mengantarkan robot untuk menang dengan poin tertinggi. Fungsi ini berkaitan dengan perhitungan hasil skor akhir permainan dimana bot dapat mendapatkan poin dengan beberapa aksi seperti menembak tepat sasaran, menabrak, dan menjadi robot yang hidup terakhir.

d. Fungsi Seleksi

Fungsi seleksi berkaitan dengan fungsi objektif dari algoritma yang diterapkan. Fungsi inilah yang akan menyeleksi kandidat-kandidat tindakan yang ada dan memilih beberapa yang dianggap sesuai dengan tujuan utama dari robot. Karena tujuan utama dari algoritma robot ini adalah mendapatkan skor melalui tembakan dan tabrakan, maka fungsi seleksinya adalah melakukan tindakan yang dapat menembak atau menabrak secara tepat.

e. Fungsi Kelayakan

Fungsi seleksi akan menentukan kelayakan dari sebuah kandidat sebelum dimasukkan ke dalam himpunan solusi. Fungsi ini akan menyeleksi agar kandidat yang dipilih tidak memberikan konsekuensi yang mengancam objektif robot sehingga akan melakukan penargetan pada musuh terdekat atau lawan yang memiliki energi rendah.

f. Fungsi Objektif

Fungsi objektif berkaitan dengan tujuan utama yang ingin dicapai dari robot. Definisi objektif akan mendasari pembuatan fungsi kelayakan, dan seleksi. Pada permainan robocode ini, tujuan utamanya

adalah mendapatkan skor paling maksimal. Dalam algoritma robot ini, pencarian skor diutamakan melalui tembakan dan tabrakan serta bonus yang bisa didapatkan.

3.1.1.2 Analisis Efisiensi Solusi

Kompleksitas algoritma ini tergolong sangat simpel. Dalam operasi pencarian, pembaruan, dan penghapusan yang dilakukan, waktu eksekusinya konstan, tidak bergantung pada ukuran dari masukan. Hal ini berarti notasi Big-O dari algoritma yang telah diimplementasikan adalah O(1). Penggunaan memori juga sangat efisien karena setiap giliran, data yang tidak digunakan akan dihapus. Implementasi strategi berupa memilih target berdasarkan energi dan jarak juga tidak membebani sistem dengan perhitungan kompleks, namun hanya dengan pembacaan serta perhitungan sederhana.

3.1.1.3 Analisis Efektifitas Solusi

Analisis efektivitas berkaitan dengan seberapa besar dampak algoritma yang diterapkan dalam mencapai objektif yang diinginkan. Dalam kasus ini, robot menggunakan analisis terhadap energi dan posisi musuh untuk menentukan target utamanya. Setelah itu, implementasi algoritma yang digunakan untuk menyerang juga memastikan bahwa tembakan akan tepat sasaran. Robot akan secara efektif mendapatkan poin besar melalui tembakan. Hal tersebut juga berlaku pada saat robot menabrak musuh. Tidak hanya itu, algoritma ini juga memungkinkan robot untuk bergerak ke sudut arena untuk menghindari kemungkinan diserang oleh banyak musuh sekaligus. Maka dari itu, kesimpulannya adalah algoritma ini sangat efektif dalam mencapai poin terbanyak melalui tembakan dan tabrakan.

3.1.2 Greedy by Bullet Damage - Bullet Damage Bonus - Survival Score

Algoritma ini bertujuan untuk mengoptimalkan strategi pertempuran bot dengan menggabungkan tiga komponen utama, yaitu bullet damage, bullet damage bonus,

dan survival score. Pendekatan ini akan memberikan potensi penembakan yang lebih akurat pada bot musuh dan memanfaatkan bonus damage yang didapat ketika membunuh bot musuh, serta menjaga posisi agar bot tetap aman dan sulit diincar oleh bot musuh dengan melakukan *dodge*.

3.1.2.1 Mapping Elemen Greedy

Terdapat beberapa elemen-elemen algoritma greedy, antara lain:

a. Himpunan Kandidat

Algoritma mempertimbangkan berbagai opsi aksi yang dapat diambil berdasarkan kondisi saat itu, seperti pilihan *firepower* (1, 2, atau 3) untuk menembak, gerakan maju atau mundur untuk menjaga jarak ideal, serta menghindar atau melakukan *dodge* ketika terjadi penurunan energi bot musuh.

b. Himpunan Solusi

Langkah-langkah yang diambil oleh bot berupa pelacakan musuh melalui radar, penyesuaian arah senjata untuk tembakan yang tepat, eksekusi tembakan dengan *firepower* optimal, dan pergerakan posisi yang mengoptimalkan keseimbangan antara menyerang dan menghindar.

c. Fungsi Solusi

Fungsi solusi menghitung skor yang ingin dioptimalkan dengan menggabungkan potensi bullet damage (berdasarkan firepower dan jarak ke musuh), bonus damage (berdasarkan energi lawan), dan survival score (keamanan posisi berdasarkan jarak dan kemungkinan terkena serangan).

d. Fungsi Seleksi

Fungsi seleksi menentukan aksi selanjutnya dengan memilih bot musuh yang terdeteksi di radar. Misalnya, jika bot musuh terlacak oleh radar, berada pada jarak dekat, dan terjadi penurunan energi, bot akan

memilih untuk menembak dengan firepower tinggi sekaligus melakukan *dodge* untuk menjaga survival score.

e. Fungsi Kelayakan

Fungsi kelayakan memastikan bahwa setiap aksi yang dipilih dapat dijalankan dengan efektif, dengan memeriksa apakah target berada dalam jangkauan efektif untuk ditembak dengan firepower tertentu dan memastikan pergerakan yang dilakukan tidak membawa bot ke posisi yang terlalu berisiko.

f. Fungsi Objektif

Fungsi objektif untuk memaksimalkan skor dengan menggabungkan komponen bullet damage, bullet damage bonus, dan survival score.

3.1.2.2 Analisis Efisiensi Solusi

Algoritma ini memiliki kompleksitas waktu $O(n)$, dimana n adalah jumlah total event yang diterima oleh bot. Setiap event OnScannedBot, OnHitWall, atau OnHitBot diproses dengan kompleksitas waktu *O(1)*. Sehingga, total kompleksitas waktu untuk memproses semua event dalam satu sesi pertempuran adalah *O(n)*.

3.1.2.3 Analisis Efektifitas Solusi

Pendekatan greedy dalam algoritma ini efektif dalam memberikan solusi optimal dengan respons cepat terhadap kondisi pertempuran yang berubah-ubah. Bot mampu menembak dengan firepower yang tepat, memanfaatkan bonus damage saat mendeteksi penurunan energi musuh, dan melakukan pergerakan untuk menghindari serangan dari bot musuh. Namun pergerakan radar pada bot ini hanya fokus pada 1 bot musuh, sehingga pergerakan menghindari musuh hanya fokus untuk 1 bot musuh.

3.1.3 Greedy by Bullet Damage - Bullet Damage Bonus - Survival Score - Last Survival Score

Algoritma ini memiliki objektif utama untuk mendapatkan skor terbanyak melalui poin yang didapatkan dari *bullet damage, bullet damage bonus, survival score dan last survival score*. Poin utama yang ingin dicapai oleh robot ini adalah menghindari kerumunan musuh agar bisa bertahan hingga akhir round. Selagi terus bergerak ke tempat yang aman, robot juga akan menembak musuh yang telah menjadi target. Algoritma ini memastikan bahwa robot akan dapat bertahan hidup sementara mendapatkan poin tambahan melalui tembakan.

3.1.3.1 Mapping Elemen Greedy

Terdapat beberapa elemen-elemen algoritma greedy, antara lain:

a. Himpunan Kandidat

Dalam algoritma yang diterapkan pada permainan Robocode, kandidat merupakan aksi yang dapat dilakukan robot setiap ronde permainannya. Himpunan ini berisikan berbagai macam pilihan yang dapat diambil sesuai dengan objektif robot. Dari banyak macam kandidat, pengembang akan melakukan seleksi dan mengaksesnya melalui algoritma kode program. Dalam konteks robot ini, kandidat yang paling diperhatikan adalah titik pergerakan yang dituju oleh robot.

b. Himpunan Solusi

Di dalam robot ini, kandidat yang dipilih menjadi solusi merupakan tempat unik di arena yang aman dari musuh. Solusi yang telah dipilih berdampak secara positif pada tujuan akhir robot yaitu menjadi yang terakhir hidup dalam satu ronde permainan. Himpunan solusi mencakup titik-titik yang dituju oleh robot dan tindakan seperti menembak atau bergerak yang diambil.

c. Fungsi Solusi

Fungsi solusi bertujuan menyeleksi kandidat mana yang sesuai dengan tujuan utama dari Bot. Pada dasarnya, objektif utamanya adalah mendapatkan skor terbanyak. Jadi, fungsi solusi hanya menyeleksi kandidat mana yang sesuai dengan objektif utamanya. Untuk memastikan bahwa algoritma ini *greedy*, akan ada fungsi seleksi yang dibentuk.

d. Fungsi Seleksi

Tujuan dari fungsi seleksi adalah memastikan solusi yang dipilih heuristik berdasarkan algoritma *greedy* yang diterapkan. Dalam kasus ini, karena robot bertujuan utama untuk hidup terakhir dari robot lainnya, maka strateginya adalah untuk mencari tempat terjauh dari kerumunan robotnya. Tujuan dari menjauhi kerumunan adalah mengurangi kemungkinan adanya tembakan yang ditujukan kepadanya. Fungsi ini akan menentukan daerah terjauh dari kumpulan robot dan bergerak ke arahnya.

e. Fungsi Kelayakan

Fungsi ini selayaknya fungi yang melindungi fungsi-fungsi lainnya dalam pemilihan. Bentuk pemastian yang dilakukan adalah tindakan yang diambil tidak melanggar aturan dari permainan. Contohnya adalah memastikan bahwa tujuan dari robot tidak berada di luar batas arena. Karena saat menabrak tembok, robot akan menerima poin kerusakan, maka robot seharusnya tidak menabrak tembok sama sekali. Maka dari itu, fungsi kelayakan ini memeriksa bahwa tindakan yang akan diambil layak dimasukkan ke dalam himpunan solusi.

f. Fungsi Objektif

Dalam permainan robocode, fungsi objektif utama adalah memaksimalkan skor yang didapatkan oleh robot selama beberapa ronde

pertarungan. Hal ini dapat dicapai melalui memilih tindakan seperti menembak, menabrak, menghindar di waktu dan letak yang tepat.

3.1.3.2 Analisis Efisiensi Solusi

Efisiensi dari sebuah algoritma dapat dinilai melalui kompleksitas waktu dan ruangnya. Dalam penerapan algoritma ini, perhitungan lokasi hanya dilakukan sebanyak jumlah musuh yang ada di setiap rondenya. Setiap giliran, robot hanya perlu untuk memindai keseluruhan arena sekali lalu memperbaharui data posisi musuh di dalam variabel. Maka dari itu, kompleksitas waktunya hanyalah O(1). Penggunaan sumber daya juga efisien dimana hanya ada perhitungan matematika sederhana yang dipanggil ketika ada musuh terdeteksi. Dengan jumlah musuh yang tidak terlalu banyak, algoritma ini akan berjalan dengan efisien.

3.1.3.3 Analisis Efektifitas Solusi

Dalam menganalisis efektifitas, algoritma ini dapat ditelaah melalui bagaimana kinerjanya terhadap objektif. Tujuan utama dari setiap robot adalah mendapatkan skor terbanyak selama beberapa ronde. Dalam algoritma *greedy* ini, fokus utama poin ditetapkan pada pendapatan *survival score* dan *last survival bonus*. Robot akan mendapatkan 50 poin setiap ada robot lain yang hancur dan mendapatkan 10 poin dikali banyaknya musuh apabila menjadi robot terakhir. Selain fokus utama tersebut, robot juga memiliki algoritma untuk menembak dan mendapatkan poin melalui *bullet damage* dan *bullet damage bonus*. Maka dari itu, karena sudah menetapkan berbagai metode yang sesuai, algoritma ini dapat dikategorikan sebagai efektif.

3.1.4 Greedy by Bullet Damage - Bullet Damage Bonus - Ram Damage - Ram Damage Bonus - Survival Score - Last Survival Score

Dalam implementasi bot ini, strategi greedy diterapkan dengan cara terus-menerus memilih aksi terbaik pada setiap event yang diterima. Pendekatan ini terlihat dari

bagaimana bot menentukan target, memilih aksi penembakan, dan pergerakan berdasarkan kondisi terkini, seperti posisi musuh, jarak, dan perubahan energi. Keputusan diambil secara cepat dan bersifat lokal, sehingga setiap aksi merupakan pilihan optimal pada saat itu meskipun belum tentu optimal secara global.

3.1.4.1 Mapping Elemen Greedy

Terdapat beberapa elemen-elemen algoritma greedy, antara lain:

a. Himpunan Kandidat

Himpunan kandidat dalam konteks bot ini mencakup seluruh opsi aksi yang dapat dilakukan, misalnya memilih tingkat firepower (1, 2, atau 3) untuk menembak, serta menentukan arah pergerakan, apakah maju, mundur, atau melakukan dodge. Di dalam kode, opsi-opsi ini diwakili oleh kondisi dan keputusan pada metode seperti ManiacBehavior, EitsBehavior, dan LawrieBehaviour, yang mempertimbangkan faktor seperti jarak ke musuh, perubahan energi lawan, dan posisi aman di arena.

b. Himpunan Solusi

Himpunan solusi adalah rangkaian langkah yang dilakukan bot untuk mencapai tujuan strategisnya. Bot melacak musuh melalui radar, menentukan posisi target, menyesuaikan arah turret untuk menembak, dan memilih jalur pergerakan yang menghindari risiko. Misalnya, pada metode MoveToSafeLocation bot memilih titik-titik aman berdasarkan jarak dari dinding dan musuh, sedangkan pada metode ManiacBehavior, EitsBehavior, dan LawrieBehavior bot memutuskan tindakan penembakan berdasarkan posisi musuh yang terdeteksi.

c. Fungsi Solusi

Fungsi solusi berperan untuk mengevaluasi potensi keuntungan dari setiap aksi dengan menggabungkan beberapa parameter, seperti bullet damage, bullet damage bonus, survival score, last survival score, ram damage, dan ram damage bonus. Dalam kode, perhitungan seperti

menentukan tingkat firepower dan penyesuaian sudut untuk menembak mencerminkan upaya untuk mengoptimalkan serangan sekaligus mempertahankan posisi aman. Bot menghitung nilai-nilai tersebut secara dinamis untuk menentukan langkah yang memberikan nilai maksimal bagi skenario saat itu.

d. Fungsi Seleksi

Fungsi seleksi dalam algoritma ini berperan menentukan aksi atau target selanjutnya berdasarkan informasi yang diperoleh. Bot memilih target dengan cara memprioritaskan musuh yang berada pada jarak dekat, memiliki penurunan energi, atau berada dalam posisi yang relatif mudah diincar, seperti pada pengecekan di enemyTracker dan pemilihan target berdasarkan jarak serta kondisi "corner" musuh. Proses seleksi ini memastikan bahwa aksi yang diambil selalu berdasarkan kondisi terkini dan dianggap paling menguntungkan.

e. Fungsi Kelayakan

Fungsi kelayakan memastikan bahwa aksi yang dipilih dapat dijalankan secara efektif dan tidak menempatkan bot pada risiko yang tidak perlu. Misalnya, sebelum bergerak ke titik aman, bot memeriksa apakah lokasi tersebut tidak berada di zona bahaya (seperti dekat dengan musuh atau di tengah arena yang sibuk) menggunakan pengecekan pada metode IsInCenterDangerZone dan IsNearEnemy. Hal ini menjamin bahwa setiap langkah yang diambil tidak hanya agresif tetapi juga mempertimbangkan aspek pertahanan.

f. Fungsi Objektif

Fungsi objektif adalah kriteria yang ingin dimaksimalkan, yakni menggabungkan potensi bullet damage, bullet damage bonus, survival score, last survival score, ram damage, dan ram damage bonus. Dalam kode, objektif ini diwujudkan dengan memilih tingkat firepower yang

tepat dan menentukan pergerakan yang meminimalkan risiko serangan musuh. Keputusan seperti menggunakan firepower tinggi ketika kondisi "locked on" terpenuhi dan pergerakan yang menghindari musuh merupakan upaya bot untuk secara langsung memaksimalkan skor keseluruhan dalam pertempuran.

3.1.4.2 Analisis Efisiensi Solusi

Algoritma yang diterapkan memiliki kompleksitas waktu $O(n)$ karena setiap event—baik itu OnScannedBot, OnHitWall, atau OnHitBot—diproses dalam waktu konstan. Setiap event ditangani secara langsung dan independen tanpa adanya iterasi yang signifikan pada skala yang lebih besar. Dengan demikian, keseluruhan proses pengolahan event dalam satu sesi pertempuran tetap efisien, bahkan ketika jumlah event meningkat.

3.1.4.3 Analisis Efektifitas Solusi

Pendekatan greedy yang digunakan terbukti efektif dalam konteks pertempuran dinamis. Bot mampu melakukan penyesuaian cepat berdasarkan kondisi terkini, seperti memilih target dengan tepat dan menentukan aksi yang menggabungkan serangan dan pertahanan secara optimal. Meskipun fokus pergerakan radar hanya pada satu bot musuh pada satu waktu, strategi ini memungkinkan respons cepat dan adaptif terhadap perubahan kondisi, sehingga meningkatkan peluang bertahan dan memberikan serangan yang tepat sasaran.

3.2 Strategi Greedy yang Dipilih

Strategi greedy yang kami pilih adalah strategi ketiga, yaitu "Greedy by Bullet Damage - Bullet Damage Bonus - Survival Score - Last Survival Score". Berikut adalah alasan dan pertimbangan pemilihan strategi tersebut:

1. **Optimalisasi Poin dengan Kombinasi Serangan dan Pertahanan:**

   Strategi ini tidak hanya mengandalkan damage tembakan (bullet damage) dan

bonus saat membunuh lawan (bullet damage bonus), tetapi juga mengintegrasikan faktor survival. Dengan memasukkan survival score dan last survival score, bot akan lebih fokus untuk mencari lokasi yang aman, sehingga mengurangi risiko terkena tembakan lawan dan meningkatkan peluang untuk bertahan hingga akhir ronde.

2. **Efisiensi dalam Pengolahan Event:**

   Pendekatan algoritma ini memiliki kompleksitas waktu O(n) untuk memproses setiap event seperti OnScannedBot, OnHitWall, dan OnHitBot. Hal ini memastikan bahwa setiap keputusan dapat diambil secara cepat dan responsif terhadap perubahan kondisi di arena.

3. **Adaptabilitas terhadap Kondisi Pertempuran:**

   Dengan memantau kondisi musuh, seperti penurunan energi dan posisi yang relatif mudah diincar, bot dapat menentukan prioritas aksi secara dinamis. Kombinasi antara tembakan dan pergerakan ke titik aman memungkinkan bot untuk menyeimbangkan antara agresif menyerang dan menghindari risiko, sehingga meningkatkan efektivitas secara keseluruhan.

4. **Fokus pada Last Survival:**

   Last survival score memberikan nilai tambah bagi bot yang berhasil bertahan sampai akhir. Pendekatan ini mendorong bot untuk tidak terjebak dalam kerumunan musuh dan memilih lokasi yang aman, sehingga peluang untuk menjadi satu-satunya yang bertahan semakin besar.

5. **Implementasi yang Relatif Sederhana namun Efektif:**

   Algoritma ini menggabungkan perhitungan sederhana—seperti evaluasi jarak, posisi aman, dan level firepower—untuk menentukan aksi terbaik. Meskipun pendekatannya lokal dan responsif, secara kolektif strategi ini menghasilkan performa yang optimal dalam pertempuran.

Dengan pertimbangan-pertimbangan di atas, strategi ketiga menjadi pilihan terbaik karena mampu memaksimalkan skor tidak hanya melalui serangan tembakan yang efektif, tetapi juga melalui penekanan pada pertahanan dan kelangsungan hidup di arena pertempuran.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Penjelasan Robot Greedy by Bullet Damage - Ram Damage - Ram Damage Bonus (Maniac)

4.1.1 Implementasi Pseudocode Algoritma

```
procedure Main()
   bot ← new Maniac()
   bot.Start()

class Maniac extends Bot
   enemyTracker ← new Dictionary<string, EnemyData>()
   rnd ← new Random()
   currentTargetId ← null
   inCorner ← false
   CornerThreshold ← 50

   constructor Maniac()
      BotInfo ← LoadConfiguration("Maniac.json")
      AdjustRadarForBodyTurn ← true
      AdjustGunForBodyTurn ← true
      AdjustRadarForGunTurn ← true

   procedure Run()
      BodyColor ← Red
      GunColor ← Black
      RadarColor ← Lime
      BulletColor ← Red
      ScanColor ← Purple

      if not inCorner then
         MoveToNearestCorner()
         inCorner ← true
      end if

      while IsRunning do
         TurnRadarLeft(Infinite)

      Rescan()

   procedure MoveToNearestCorner()
      arenaWidth ← ArenaWidth
      arenaHeight ← ArenaHeight
```

```
        corners ← list of
            (CornerThreshold, CornerThreshold),
            (arenaWidth - CornerThreshold, CornerThreshold),
            (CornerThreshold, arenaHeight - CornerThreshold),
            (arenaWidth - CornerThreshold, arenaHeight - CornerThreshold)

        // Pilih sudut terdekat berdasarkan jarak Euclidean dari posisi (X, Y) saat ini
        (targetX, targetY) ← corner in corners with minimal sqrt((corner.X - X)² + (corner.Y -
Y)²)

        angleToCorner ← atan2(targetY - Y, targetX - X) * (180/π)
        turnAngle ← NormalizeRelativeAngle(angleToCorner - Direction)

        if turnAngle ≥ 0 then
            SetTurnRight(turnAngle)
        else
            SetTurnLeft(-turnAngle)
        end if

        distance ← sqrt((targetX - X)² + (targetY - Y)²)
        SetForward(distance)

        Go()

    procedure IsEnemyInCorner(enemy: EnemyData) → boolean
        arenaWidth ← ArenaWidth
        arenaHeight ← ArenaHeight
        return ((enemy.X ≤ CornerThreshold or enemy.X ≥ arenaWidth - CornerThreshold)
and
                (enemy.Y ≤ CornerThreshold or enemy.Y ≥ arenaHeight - CornerThreshold))

    // Event: Ketika bot mendeteksi musuh
    procedure OnScannedBot(event e)
        currentTurn ← TurnNumber

        // Hapus data musuh yang sudah usang
        for each key in enemyTracker.keys() do
            if (currentTurn - enemyTracker[key].LastTurnScanned) > 1 then
                enemyTracker.Remove(key)
                if currentTargetId = key then
                    currentTargetId ← null
                end if
            end if
        end for

        enemyIdStr ← ConvertToString(e.ScannedBotId)
        lockedOn ← false
```

```
    previousBearing ← 0

    // Perbarui tracking data musuh
    if enemyTracker contains key enemyIdStr then
        data ← enemyTracker[enemyIdStr]
        previousBearing ← atan2(data.X - Y, data.Y - X) * (180/π)
        data.PrevX ← data.X
        data.PrevY ← data.Y
    else
        enemyTracker[enemyIdStr] ← new EnemyData()
    end if

    enemy ← enemyTracker[enemyIdStr]
    newBearing ← atan2(e.Y - Y, e.X - X) * (180/π)
    enemy.X ← e.X
    enemy.Y ← e.Y
    enemy.Energy ← e.Energy
    enemy.LastTurnScanned ← currentTurn

    if enemy.PrevX ≠ 0 or enemy.PrevY ≠ 0 then
        bearingDifference ← Abs(NormalizeRelativeAngle(newBearing -
previousBearing))
        if bearingDifference < 5 then
            lockedOn ← true
        end if
    end if

    // Prioritaskan musuh yang berada di sudut. Jika tidak ada, pilih musuh terdekat.
    if currentTargetId is null or enemyTracker does not contain key currentTargetId then
        cornerEnemies ← filter enemyTracker where IsEnemyInCorner(enemy) is true
        if cornerEnemies is not empty then
            (key, enemyData) ← entry in cornerEnemies with minimal DistanceTo(enemy.X,
enemy.Y)
            currentTargetId ← key
        else
            (key, enemyData) ← entry in enemyTracker with minimal DistanceTo(enemy.X,
enemy.Y)
            if enemyData is not null then
                currentTargetId ← key
            end if
        end if
    end if

    if currentTargetId is null then
        return
    end if

    // Target yang telah dipilih
```

```
        target ← enemyTracker[currentTargetId]
        angleToEnemy ← NormalizeAbsoluteAngle(Direction + BearingTo(target.X,
target.Y))
        distance ← DistanceTo(target.X, target.Y)

        // --- Radar Lock-On ---
        radarTurn ← NormalizeRelativeAngle(angleToEnemy - RadarDirection)
        extraTurn ← Min(Atan(36.0 / distance), MaxTurnRate)
        if radarTurn < 0 then
            radarTurn ← radarTurn - extraTurn
        else
            radarTurn ← radarTurn + extraTurn
        end if
        SetTurnRadarLeft(radarTurn)

        // --- Gun Tracking ---
        gunTurn ← NormalizeRelativeAngle(angleToEnemy - GunDirection)
        SetTurnGunLeft(gunTurn)
        if Abs(gunTurn) < 10 then
            firePower ← (lockedOn ? 3 : 2)
            Fire(firePower)
        end if

        // --- Pergerakan Menuju Target ---
        moveAngle ← NormalizeRelativeAngle(angleToEnemy - Direction)
        if moveAngle ≥ 0 then
            SetTurnLeft(moveAngle)
        else
            SetTurnRight(-moveAngle)
        end if

        if distance > 100 then
            SetForward(100)
        else
            SetForward(50)
        end if

    procedure TurnToFaceTarget(targetX, targetY)
        desiredAngle ← atan2(targetY - Y, targetX - X) * (180/π)
        turnAngle ← NormalizeRelativeAngle(desiredAngle - Direction)
        if turnAngle ≥ 0 then
            SetTurnRight(turnAngle)
        else
            SetTurnLeft(-turnAngle)
        end if
        Go()

    // Event: Ketika bot ditabrak musuh
```

```
procedure OnHitBot(event e)
   TurnToFaceTarget(e.X, e.Y)

   if e.Energy > 10 then
      Fire(3)
   else if e.Energy > 4 then
      Fire(1)
   else if e.Energy > 2 then
      Fire(0.5)
   else if e.Energy > 0.4 then
      Fire(0.1)
   end if

   Forward(40)

// Event: Ketika bot menabrak dinding
procedure OnHitWall(event e)
   bearing ← BearingTo(X, Y)
   if bearing ≥ 0 then
      SetTurnLeft(bearing)
   else
      SetTurnRight(bearing)
   end if
   SetForward(50)
```

4.1.2 Penjelasan Struktur Data

- EnemyData (Class)

  Kelas ini menyimpan informasi penting mengenai musuh, seperti posisi (X, Y), energi, dan informasi pemindaian sebelumnya (PrevX, PrevY serta giliran terakhir terdeteksi). Data ini digunakan untuk:

  - ➢ Melacak pergerakan musuh (dengan membandingkan posisi saat ini dan sebelumnya).
  - ➢ Menentukan apakah musuh sedang *locked-on* (perubahan sudut kecil menunjukkan pergerakan linear).

- enemyTracker (Dictionary)

  Struktur data dictionary dengan key berupa ID musuh (sebagai string) dan value berupa objek EnemyData. Dictionary ini digunakan untuk menyimpan dan

meng-update data musuh secara real time sehingga bot dapat memilih target yang optimal (misalnya, musuh yang berada di pojok atau yang paling dekat).

4.1.3 Penjelasan Fungsi dan Prosedur

- Run()

  Fungsi utama bot yang melakukan inisialisasi dengan memindahkan bot ke pojok arena (strategi awal untuk mengamankan posisi) menggunakan prosedur MoveToNearestCorner() dan memasuki loop utama untuk melakukan scan musuh secara terus-menerus dengan perputaran radar tanpa henti.

- MoveToNearestCorner()

  Prosedur untuk menggerakkan bot ke pojok terdekat berdasarkan perhitungan jarak Euclidean. Strategi ini merupakan bagian dari solusi greedy, karena bot langsung memilih pojok terdekat tanpa perhitungan strategi jangka panjang.

- IsEnemyInCorner()

  Fungsi helper yang menentukan apakah musuh berada dalam area pojok dengan membandingkan posisi musuh terhadap threshold (CornerThreshold).

- OnScannedBot(e):

  Fungsi event handler yang dipanggil setiap kali radar menemukan musuh.

  - ➢ Memperbarui data musuh di enemyTracker dan menghapus data yang sudah usang (lebih dari 1 giliran).
  - ➢ Menghitung perubahan arah (bearing difference) untuk menentukan apakah musuh locked-on.
  - ➢ Melakukan pemilihan target dengan prioritas pada musuh yang berada di pojok atau paling dekat.
  - ➢ Mengatur pergerakan radar, penargetan gun, dan pergerakan bot untuk mendekati target.
  - ➢ Menentukan kekuatan tembakan berdasarkan kondisi locked-on.

- TurnToFaceTarget()

Prosedur bantu yang mengatur bot untuk menghadap ke target tertentu, digunakan terutama ketika terjadi tabrakan dengan bot lain.

- OnHitBot(e)

  Mengatur pergerakan dan tembakan ketika terjadi tabrakan dengan bot lain, dengan penyesuaian firepower berdasarkan energi musuh.

- OnHitWall(e)

  Menyesuaikan pergerakan bot untuk menghindari dinding, sehingga bot tidak terjebak atau kehilangan posisi strategis.

4.2 Penjelasan Robot Greedy by Bullet Damage - Bullet Damage Bonus - Survival Score (Eits)

4.2.1 Implementasi Pseudocode Algoritma

```
procedure Main()
    bot ← new Eits()
    bot.Start()

class Eits extends Bot
    enemyEnergy ← new Dictionary()

    constructor Eits()
        BotInfo ← LoadConfiguration("Eits.json")
        AdjustRadarForBodyTurn ← true
        AdjustGunForBodyTurn ← true
        AdjustRadarForGunTurn ← true

    procedure Run()
        BodyColor = Color.White;
        GunColor = Color.White;
        TurretColor = Color.White;
        RadarColor = Color.White;
        ScanColor = Color.White;
        BulletColor = Color.White;

        while IsRunning do
            TurnRadarLeft(Infinite)

    // Event: Ketika bot mendeteksi musuh
```

```
procedure OnScannedBot(event e)
    // Hitung sudut relatif ke musuh
    bearing ← ComputeBearingTo(e.X, e.Y)
    angleToEnemy ← CurrentDirection + bearing
    distance ← ComputeDistanceTo(e.X, e.Y)

    // --- Radar Tracking ---
    radarTurn ← NormalizeAngle(angleToEnemy - CurrentRadarDirection)
    extraTurn ← Min(Atan(36.0 / distance), MaxTurnRate)

    if radarTurn < 0 then
        radarTurn ← radarTurn - extraTurn
    else
        radarTurn ← radarTurn + extraTurn
    SetTurnRadarLeft(radarTurn)
    end if

    // --- Gun Tracking dan Penembakan ---
    gunTurn ← NormalizeAngle(angleToEnemy - CurrentGunDirection)
    SetTurnGunLeft(gunTurn)

    if distance < 200 then
        firepower ← 3
    else
        firepower ← 2
    SetFire(firepower)
    end if

    // --- Pergerakan untuk Menjaga Jarak ---
    desiredDistance ← 250
    distanceError ← distance - desiredDistance

    if distanceError > 50 then
        SetTurnLeft(NormalizeAngle(angleToEnemy - CurrentDirection))
        SetForward(100)
    else if distanceError < -50 then
        SetTurnRight(NormalizeAngle(angleToEnemy + CurrentDirection))
        SetBack(100)
    end if

    // --- Deteksi Penembakan Bullet oleh Musuh ---
    if enemyEnergy contains key e.ScannedBotId then
        energyDrop ← enemyEnergy[e.ScannedBotId] - e.Energy
        // Menghindar jika musuh kemungkinan menembak
        if energyDrop between 0.1 and 3.0 then
            PerformDodge()
        end if
    end if
```

```
        enemyEnergy[e.ScannedBotId] ← e.Energy

    // Metode untuk menghindari bullet
    procedure PerformDodge()
        if RandomChoice(0, 1) equals 0 then
            dodgeAngle ← 90
        else
            dodgeAngle ← -90
        end if

        SetTurnLeft(NormalizeAngle(CurrentDirection + dodgeAngle - CurrentDirection) +
90)
        SetForward(100)

    // Event: Ketika bot menabrak dinding
    procedure OnHitWall(event e)
        bearing ← ComputeBearingTo(CurrentX, CurrentY)
        if bearing ≥ 0 then
            SetTurnLeft(bearing)
        else
            SetTurnRight(bearing)
        end if
        SetForward(50)
```

4.2.2 Penjelasan Struktur Data

- enemyEnergy (Dictionary)

  Sebuah struktur data *dictionary* digunakan untuk menyimpan nilai energi masing-masing musuh dengan key sebagai ID musuh. Ini berguna untuk mendeteksi penurunan energi musuh, yang dapat menandakan bahwa musuh baru saja menembak, sehingga bot bisa mengambil tindakan *dodge*.

4.2.3 Penjelasan Fungsi dan Prosedur

- Run()

  Fungsi utama yang dipanggil saat bot mulai dijalankan. Di sini bot mengatur warnanya dan memulai loop utama dengan melakukan perputaran radar tanpa henti. Pendekatan greedy digunakan di sini dengan langsung berfokus pada deteksi musuh melalui radar.

- OnScannedBot(ScannedBotEvent e)

  Fungsi event handler yang dipanggil setiap kali radar menemukan musuh.

  - ➢ Radar Tracking: Menggunakan perhitungan sudut dan penyesuaian ekstra untuk memastikan radar tetap terkunci pada musuh.
  - ➢ Gun Tracking & Firing: Menentukan besaran firepower berdasarkan jarak dari musuh.
  - ➢ Movement: Bot bergerak maju atau mundur untuk menjaga jarak optimal dengan musuh (target distance: 250 unit).
  - ➢ Bullet Dodging: Dengan membandingkan energi musuh yang sebelumnya dan saat ini, bot mendeteksi kemungkinan tembakan musuh dan kemudian menjalankan prosedur penghindaran (PerformDodge).

- PerformDodge()

  Prosedur yang menangani gerakan menghindar saat terdeteksi bahwa musuh menembak. Gerakan di sini berupa perputaran secara acak ke kiri atau ke kanan (90 derajat) kemudian maju sejauh 100 unit. Ini merupakan solusi greedy karena bot segera bereaksi terhadap indikasi tembakan dengan aksi menghindar yang sederhana.

- OnHitWall(HitWallEvent e)

  Event handler untuk menangani tabrakan dengan dinding. Bot menghitung sudut relatif terhadap dinding, kemudian menyesuaikan arah dan bergerak maju untuk menghindari masalah posisi yang terlalu dekat dengan tembok.

4.3 Penjelasan Robot Greedy by Bullet Damage - Bullet Damage Bonus - Survival Score - Last Survival Score (Lawrie)

4.3.1 Implementasi Pseudocode Algoritma

```
procedure Main()
    bot ← new Lawrie()
    bot.Start()

class Lawrie extends Bot
    enemyLocations ← new Dictionary()
    minDistanceFromWall ← 50
    dangerZoneMargin ← 50
    enemyDangerRadius ← 200
    random ← new Random()

    constructor Lawrie()
        BotInfo ← LoadConfiguration("Lawrie.json")

    procedure Run()
        BodyColor ← Green
        TurretColor ← Red
        RadarColor ← White
        while IsRunning do
            Print("Posisi Saya: (" + X + ", " + Y + ") | Arah: " + Direction)
            TurnRadarLeft(Infinite)

    procedure OnScannedBot(event e)
        enemyLocations[e.ScannedBotId] ← (e.X, e.Y)
        angleToEnemy ← NormalizeAbsoluteAngle(Direction + BearingTo(e.X, e.Y))
        distance ← DistanceTo(e.X, e.Y)
        // --- Radar Tracking ---
        radarTurn ← NormalizeRelativeAngle(angleToEnemy - RadarDirection)
        extraTurn ← Min(Atan(36.0 / distance), MaxTurnRate)
        if radarTurn < 0 then
            radarTurn ← radarTurn - extraTurn
        else
            radarTurn ← radarTurn + extraTurn
    end if
        SetTurnRadarLeft(radarTurn)
        // --- Gun Tracking dan Penembakan ---
        gunTurn ← NormalizeRelativeAngle(angleToEnemy - GunDirection)
        SetTurnGunLeft(gunTurn)
        Fire(2)
        MoveToSafeLocation()
```

```
    procedure MoveToSafeLocation()
        potentialSpots ← List of spots:
            (ArenaWidth - minDistanceFromWall, ArenaHeight -
minDistanceFromWall)
            (minDistanceFromWall, ArenaHeight - minDistanceFromWall)
            (ArenaWidth - minDistanceFromWall, minDistanceFromWall)
            (minDistanceFromWall, minDistanceFromWall)

            validSpots ← Filter potentialSpots where:
            (not IsInCenterDangerZone(spot.X, spot.Y)) and (not
IsNearEnemy(spot.X, spot.Y))
        if validSpots is not empty then
            // Pilih spot dengan jarak terjauh dari musuh
            safestSpot ← Spot di validSpots dengan nilai minimum jarak ke musuh
tertinggi
            targetX ← safestSpot.X
            targetY ← safestSpot.Y
            Print("Safe spot dipilih: (" + targetX + ", " + targetY + ")")
        else
            alternativeSpot ← Spot di potentialSpots dengan jarak maksimum ke
(ArenaWidth/2, ArenaHeight/2)
            targetX ← alternativeSpot.X
            targetY ← alternativeSpot.Y
        targetX ← Max(minDistanceFromWall, Min(ArenaWidth -
minDistanceFromWall, target X))
        targetY ← Max(minDistanceFromWall, Min(ArenaHeight -
minDistanceFromWall, target Y))
        GoToPosition(targetX, targetY)



    function IsInCenterDangerZone(x, y)
        centerX ← ArenaWidth / 2
        centerY ← ArenaHeight / 2
        return (Abs(x - centerX) < dangerZoneMargin) and (Abs(y - centerY) <
dangerZoneMargin)

    function IsNearEnemy(x, y)
        for each enemy in enemyLocations.Values do
            distance ← EuclideanDistance(enemy.X, enemy.Y, x, y)
            if distance < enemyDangerRadius then
                return true
        return false

    procedure GoToPosition(targetX, targetY)
        bearing ← BearingTo(targetX, targetY)
        angleToTarget ← NormalizeRelativeAngle(bearing + Direction)
        distance ← DistanceTo(targetX, targetY)
```

```
        if angleToTarget ≥ 0 then
            SetTurnLeft(angleToTarget)
        else
            SetTurnRight(-angleToTarget)
    SetForward(distance)
```

4.3.2 Penjelasan Struktur Data

- enemyLocations (Dictionary)

  Menyimpan data dari hasil pemindaian oleh robot. Data yang disimpan adalah ID musuh beserta koordinat letaknya saat terpindai oleh radar. Struktur data ini digunakan dalam menentukan titik terjauh dari musuh-musuh yang ada.

- potentialSpots (List)

  Menyimpan 4 data titik yang mungkin dituju oleh robot. Data ini menyimpan koordinat bawah-kiri, bawah-kanan, atas-kiri, dan atas-kanan dari arena. Struktur data ini digunakan sebagai titik potensial tujuan robot yang merupakan titik terjauh dari musuh-musuh ada.

4.3.3 Penjelasan Fungsi dan Prosedur

- Run()

  Fungsi utama yang dipanggil saat bot mulai dijalankan. Di sini bot mengatur warnanya dan memulai loop utama dengan melakukan perputaran radar tanpa henti. Pendekatan greedy digunakan di sini dengan langsung berfokus pada deteksi musuh melalui radar.

- OnScannedBot(event e)

  Dalam robot ini, fungsi OnScannedBot akan dipanggil ketika ada robot yang terpindai oleh radar. Kemudian, data mengenai posisi musuh akan ditambahkan ke dalam enemyLocations. Selain itu, sudut dan jarak ke musuh juga akan dihitung dan arah dari senjata disesuaikan ke arah target. Terakhir, akan ada pemanggilan prosedur MoveToSafeLocation()

- MoveToSafeLocation()

  Prosedur ini merupakan algoritma utama yang diterapkan pada robot ini. Pada awalnya, didefinisikan empat titik kandidat tujuan bagi robot. Setelah itu, titik-titik tersebut akan diseleksi dengan fungsi IsInCenterDangerZone() dan IsNearEnemy(). Setelah seleksi pertama, maka akan ditentukan satu spot yang paling jauh dari musuh. Selain itu, akan ada fungsi Max() dan Min() yang dikombinasikan agar koordinat target tidak keluar batas arena. Teerakhir, setelah menentukan titik yang terbaik, akan dipanggil prosedur GoToPosition() agar robot bergerak ke titik tersebut.

- Max()

  Fungsi ini akan mengembalikan nilai yang paling besar dari perbandingan 2 variabel.

- Min()

  Fungsi ini akan mengembalikan nilai yang paling kecil dari perbandingan 2 variabel.

- IsInCenterDangerZone(x,y)

  Fungsi ini akan melakukan pengecekan apabila suatu titik berada pada pusat arena. Awalnya, didefinisikan area tengah yang dianggap bahaya yaitu radius 50 piksel dari pusat arena. Jadi, titik yang berada di dalam radius itu dikategorikan sebagai berbahaya.

- IsNearEnemy(x,y)

  Fungsi ini akan melakukan pengecekan apabila suatu titik berada dalam radius bahaya dari musuh yang datanya tersimpan dalam enemyLocations. Daerah radiusnya adalah 200 piksel dari pusat robot. Sehingga, untuk titik yang berada di dalam radius tersebut, dikategorikan sebagai berbahaya karena dekat dengan musuh.

- GoToPosition()

Prosedur GoToPosition() berfungsi untuk mengarahkan dan menggerakan robot ke titik tujuannya. Prosedur ini menghitung sudut yang perlu diputar oleh robot serta jarak pergerakan dari robot tersebut untuk sampai ke titik tujuan.

4.4 Penjelasan Robot Greedy by Bullet Damage - Bullet Damage Bonus - Ram Damage - Ram Damage Bonus - Survival Score - Last Survival Score (Union)

4.4.1 Implementasi Pseudocode Algoritma

```
procedure Main()
    bot ← new Union()
    bot.Start()

class Union extends Bot
    HighEnergyThreshold ← 60
    MiddleEnergyThreshold ← 30

    enemyLocations ← new Dictionary<int, (double X, double Y)>
    enemyEnergy ← new Dictionary<int, double>
    enemyTracker ← new Dictionary<string, EnemyData>()

    // --- Konstanta lain ---
    minDistanceFromWall ← 50
    dangerZoneMargin ← 50
    enemyDangerRadius ← 200
    CornerThreshold ← 50
    inCorner ← false
    currentTargetId ← null
    random ← new Random()

    constructor Union()
        BotInfo ← LoadConfiguration("Union.json")

    procedure Run()
        BodyColor ← Green
        TurretColor ← White
        RadarColor ← White

        // Jika energi tinggi (Maniac) dan belum berada di sudut, pindah ke sudut terdekat
        if Energy ≥ HighEnergyThreshold and not inCorner then
            MoveToNearestCorner()
            inCorner ← true
        end if
```

```
    while IsRunning do
        TurnRadarLeft(Infinite)
    end while

// Event: Ketika bot mendeteksi musuh
procedure OnScannedBot(event e)
    // Pilih perilaku berdasarkan level energi bot
    if Energy ≥ HighEnergyThreshold then
        ManiacBehavior(e)
    else if Energy ≥ MiddleEnergyThreshold then
        EitsBehavior(e)
    else
        LawrieBehavior(e)
    end if

#region Lawrie Behavior (Low Energy)
procedure LawrieBehavior(event e)
    enemyLocations[e.ScannedBotId] ← (e.X, e.Y)
    angleToEnemy ← NormalizeAbsoluteAngle(Direction + BearingTo(e.X, e.Y))
    distance ← DistanceTo(e.X, e.Y)

    // Radar tracking
    radarTurn ← NormalizeRelativeAngle(angleToEnemy - RadarDirection)
    extraTurn ← Min(Atan(36.0 / distance), MaxTurnRate)
    if radarTurn < 0 then
        radarTurn ← radarTurn - extraTurn
    else
        radarTurn ← radarTurn + extraTurn
    end if
    SetTurnRadarLeft(radarTurn)

    gunTurn ← NormalizeRelativeAngle(angleToEnemy - GunDirection)
    SetTurnGunLeft(gunTurn)
    Fire(1)

    MoveToSafeLocation()
end procedure

procedure MoveToSafeLocation()
    potentialSpots ← list of
        (ArenaWidth - minDistanceFromWall, ArenaHeight - minDistanceFromWall),
        (minDistanceFromWall, ArenaHeight - minDistanceFromWall),
        (ArenaWidth - minDistanceFromWall, minDistanceFromWall),
        (minDistanceFromWall, minDistanceFromWall)

    validSpots ← filter potentialSpots where:
        not IsInCenterDangerZone(spot.X, spot.Y) and not IsNearEnemy(spot.X, spot.Y)
```

```
    if validSpots is not empty then
        // Pilih spot dengan jarak maksimum dari musuh terdekat
        safestSpot ← spot in validSpots with maximum (minimum distance from each
enemy in enemyLocations)
        (targetX, targetY) ← (safestSpot.X, safestSpot.Y)
    else
        // Jika tidak ada spot aman, pilih spot yang paling jauh dari pusat arena
        alternativeSpot ← spot in potentialSpots with maximum (Distance from center)
        (targetX, targetY) ← (alternativeSpot.X, alternativeSpot.Y)

    end if

    // Pastikan target berada dalam batas arena
    targetX ← Max(minDistanceFromWall, Min(ArenaWidth - minDistanceFromWall,
targetX))
    targetY ← Max(minDistanceFromWall, Min(ArenaHeight - minDistanceFromWall,
targetY))

    GoToPosition(targetX, targetY)
  end procedure

  procedure IsInCenterDangerZone(x, y) → boolean
    centerX ← ArenaWidth / 2
    centerY ← ArenaHeight / 2
    return (Abs(x - centerX) < dangerZoneMargin) and (Abs(y - centerY) <
dangerZoneMargin)
  end procedure

  procedure IsNearEnemy(x, y) → boolean
    for each enemy in enemyLocations.Values do
      d ← Sqrt((enemy.X - x)^2 + (enemy.Y - y)^2)
      if d < enemyDangerRadius then
        return true
      end if
    end for
    return false
  end procedure

  procedure GoToPosition(targetX, targetY)
    bearing ← BearingTo(targetX, targetY)
    angleToTarget ← NormalizeRelativeAngle(bearing + Direction)
    distance ← DistanceTo(targetX, targetY)
    if angleToTarget ≥ 0 then
      SetTurnLeft(angleToTarget)
    else
      SetTurnRight(-angleToTarget)
    end if
```

```
    SetForward(distance)
  end procedure
  #endregion

  #region Maniac Behavior (High Energy)
  procedure ManiacBehavior(event e)
    currentTurn ← TurnNumber

    // Bersihkan data musuh usang dari enemyTracker
    for each key in enemyTracker.keys() do
      if (currentTurn - enemyTracker[key].LastTurnScanned) > 1 then
        enemyTracker.Remove(key)
        if currentTargetId = key then
          currentTargetId ← null
        end if
      end if
    end for

    enemyIdStr ← ConvertToString(e.ScannedBotId)
    lockedOn ← false
    previousBearing ← 0

    if enemyTracker contains key enemyIdStr then
      data ← enemyTracker[enemyIdStr]
      previousBearing ← Atan2(data.X - Y, data.Y - X) * (180/π)
      data.PrevX ← data.X
      data.PrevY ← data.Y
    else
      enemyTracker[enemyIdStr] ← new EnemyData()
    end if

    enemy ← enemyTracker[enemyIdStr]
    newBearing ← Atan2(e.Y - Y, e.X - X) * (180/π)
    enemy.X ← e.X
    enemy.Y ← e.Y
    enemy.Energy ← e.Energy
    enemy.LastTurnScanned ← currentTurn

    if enemy.PrevX ≠ 0 or enemy.PrevY ≠ 0 then
      bearingDifference ← Abs(NormalizeRelativeAngle(newBearing -
previousBearing))
      if bearingDifference < 5 then
        lockedOn ← true
      end if
    end if

    // Seleksi target: Utamakan musuh yang berada di sudut arena
    if currentTargetId is null or enemyTracker does not contain currentTargetId then
```

```
        cornerEnemies ← filter enemyTracker where IsEnemyInCorner(enemy) is true
        if cornerEnemies is not empty then
            targetEntry ← entry in cornerEnemies with minimal DistanceTo(enemy.X,
enemy.Y)
            currentTargetId ← targetEntry.Key
        else
            targetEntry ← entry in enemyTracker with minimal DistanceTo(enemy.X,
enemy.Y)
            if targetEntry.Value exists then
                currentTargetId ← targetEntry.Key
            end if
        end if
    end if

    if currentTargetId is null then
        return
    end if

    target ← enemyTracker[currentTargetId]
    angleToEnemy ← NormalizeAbsoluteAngle(Direction + BearingTo(target.X,
target.Y))
    distance ← DistanceTo(target.X, target.Y)

    // Radar lock-on
    radarTurn ← NormalizeRelativeAngle(angleToEnemy - RadarDirection)
    extraTurn ← Min(Atan(36.0 / distance), MaxTurnRate)
    if radarTurn < 0 then
        radarTurn ← radarTurn - extraTurn
    else
        radarTurn ← radarTurn + extraTurn
    end if
    SetTurnRadarLeft(radarTurn)

    // Gun tracking dan tembakan
    gunTurn ← NormalizeRelativeAngle(angleToEnemy - GunDirection)
    SetTurnGunLeft(gunTurn)
    if Abs(gunTurn) < 10 then
        firePower ← (lockedOn ? 3 : 2)
        Fire(firePower)
    end if

    // Gerakan menuju target
    moveAngle ← NormalizeRelativeAngle(angleToEnemy - Direction)
    if moveAngle ≥ 0 then
        SetTurnLeft(moveAngle)
    else
        SetTurnRight(-moveAngle)
    end if
```

```
    if distance > 100 then
        SetForward(100)
    else
        SetForward(50)
    end if
  end procedure

  procedure IsEnemyInCorner(enemy: EnemyData) → boolean
    return (enemy.X ≤ CornerThreshold or enemy.X ≥ ArenaWidth - CornerThreshold)
and
          (enemy.Y ≤ CornerThreshold or enemy.Y ≥ ArenaHeight - CornerThreshold)
  end procedure

  procedure MoveToNearestCorner()
    corners ← list of
      (CornerThreshold, CornerThreshold),
      (ArenaWidth - CornerThreshold, CornerThreshold),
      (CornerThreshold, ArenaHeight - CornerThreshold),
      (ArenaWidth - CornerThreshold, ArenaHeight - CornerThreshold)

    (targetX, targetY) ← corner in corners with minimal Sqrt((corner.X - X)^2 +
(corner.Y - Y)^2)

    angleToCorner ← Atan2(targetY - Y, targetX - X) * (180/π)
    turnAngle ← NormalizeRelativeAngle(angleToCorner - Direction)
    if turnAngle ≥ 0 then
      SetTurnRight(turnAngle)
    else
      SetTurnLeft(-turnAngle)
    end if

    distance ← Sqrt((targetX - X)^2 + (targetY - Y)^2)
    SetForward(distance)
    Go()
  end procedure
  #endregion

  #region Eits Behavior (Medium Energy)
  procedure EitsBehavior(event e)
    bearing ← BearingTo(e.X, e.Y)
    angleToEnemy ← Direction + bearing
    distance ← DistanceTo(e.X, e.Y)

    // Radar tracking
    radarTurn ← NormalizeRelativeAngle(angleToEnemy - RadarDirection)
    extraTurn ← Min(Atan(36.0 / distance), MaxTurnRate)
    if radarTurn < 0 then
```

```
        radarTurn ← radarTurn - extraTurn
    else
        radarTurn ← radarTurn + extraTurn
    end if
    SetTurnRadarLeft(radarTurn)

    // Gun tracking dan tembakan
    gunTurn ← NormalizeRelativeAngle(angleToEnemy - GunDirection)
    SetTurnGunLeft(gunTurn)
    firepower ← (distance < 200 ? 3 : 2)
    SetFire(firepower)

    // Menjaga jarak ideal (desiredDistance = 250)
    desiredDistance ← 250
    distanceError ← distance - desiredDistance
    if distanceError > 50 then
        SetTurnLeft(NormalizeRelativeAngle(angleToEnemy - Direction))
        SetForward(100)
    else if distanceError < -50 then
        SetTurnRight(NormalizeRelativeAngle(angleToEnemy + Direction))
        SetBack(100)
    end if

    // Bullet dodging
    if enemyEnergy contains key e.ScannedBotId then
        energyDrop ← enemyEnergy[e.ScannedBotId] - e.Energy
        if energyDrop ≥ 0.1 and energyDrop ≤ 3.0 then
            PerformDodge()
        end if
    end if
    enemyEnergy[e.ScannedBotId] ← e.Energy
end procedure

procedure PerformDodge()
    if random.Next(2) equals 0 then
        moveAngle ← Direction + 90
    else
        moveAngle ← Direction - 90
    end if
    SetTurnLeft(NormalizeRelativeAngle(moveAngle - Direction))
    SetForward(100)
end procedure
#endregion

#region Common Event Handlers
procedure OnHitBot(event e)
    TurnToFaceTarget(e.X, e.Y)
    // Tembak dengan power berdasarkan energi musuh
```

```
    if e.Energy > 10 then
        Fire(3)
    else if e.Energy > 4 then
        Fire(1)
    else if e.Energy > 2 then
        Fire(0.5)
    else if e.Energy > 0.4 then
        Fire(0.1)
    end if
    Forward(40)
end procedure

procedure OnHitWall(event e)
    bearing ← BearingTo(X, Y)
    if bearing ≥ 0 then
        SetTurnLeft(bearing)
    else
        SetTurnRight(bearing)
    end if
    SetForward(50)
end procedure

procedure TurnToFaceTarget(targetX, targetY)
    desiredAngle ← Atan2(targetY - Y, targetX - X) * (180/π)
    turnAngle ← NormalizeRelativeAngle(desiredAngle - Direction)
    if turnAngle ≥ 0 then
        SetTurnRight(turnAngle)
    else
        SetTurnLeft(-turnAngle)
    end if
    Go()
end procedure
#endregion
```

4.4.2 Penjelasan Struktur Data

- enemyTracker (Dictionary)

  Dictionary untuk menyimpan informasi musuh yang mencakup posisi, energi, dan waktu terakhir terdeteksi.

- enemyLocations (Dictionary)

  Dictionary untuk menyimpan posisi terakhir musuh yang terdeteksi.

- enemyEnergy (Dictionary)

  Dictionary yang digunakan untuk melacak perubahan energi musuh untuk mendeteksi terjadinya tembakan.

4.4.3 Penjelasan Fungsi dan Prosedur

- Run()

  Fungsi utama yang menginisialisasi bot dan menjalankan perintah untuk memindai arena.

- OnScannedBot(e)

  Handler untuk event pemindaian musuh, memilih strategi berdasarkan tingkat energi.

- ManiacBehavior(e)

  Strategi agresif yang mengunci target dan menyerang dengan daya tembak tinggi.

- EitsBehavior(e)

  Strategi seimbang yang mempertahankan jarak optimal dan menghindari tembakan musuh.

- LawrieBehavior(e)

  Strategi defensif dengan prioritas bertahan hidup.

4.5 Pengujian dan Analisis

4.5.1 Pengujian Bot

Hasil Pengujian 1

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Lawrie 1.0	2776	1350	270	910	134	112	0	9	0	0
2	Union 1.0	1882	550	0	820	57	385	69	0	5	3
3	Eits 1.0	1648	750	30	722	43	102	0	1	2	5
4	Maniac 1.0	1245	350	0	536	15	332	11	0	3	2

Hasil Pengujian 2

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Maniac 1.0	2484	750	60	910	75	563	125	3	3	3
2	Lawrie 1.0	2302	1050	180	810	83	154	25	4	3	1
3	Eits 1.0	1682	700	60	698	49	174	0	2	2	3
4	Union 1.0	1597	500	0	740	38	300	18	1	2	3

Hasil Pengujian 3

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Lawrie 1.0	2648	1200	240	960	135	113	0	6	1	1
2	Maniac 1.0	2128	700	30	844	103	401	50	4	2	2
3	Eits 1.0	1469	650	30	578	21	186	4	0	4	4
4	Union 1.0	1397	450	0	552	47	336	11	0	3	3

Hasil Pengujian 4

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Lawrie 1.0	2575	1100	180	970	103	199	22	6	1	2
2	Union 1.0	2032	900	60	710	43	277	42	2	2	5
3	Maniac 1.0	1976	600	0	742	60	486	87	0	6	2
4	Eits 1.0	1511	400	60	726	22	274	30	2	1	1

Hasil Pengujian 5

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Lawrie 1.0	2428	1200	180	910	92	46	0	5	3	1
2	Union 1.0	2236	900	60	854	93	305	24	2	6	1
3	Maniac 1.0	1601	450	0	690	62	367	32	1	1	5
4	Eits 1.0	1338	450	60	600	29	193	5	2	0	3

4.5.2 Hasil Analisis

Berdasarkan hasil pengujian sebanyak lima kali, didapatkan data bahwa Robot Lawrie memiliki frekuensi peringkat pertama paling banyak diantara bot lainnya. Keunggulan ini terutama didukung oleh perpaduan strategi *bullet damage, bullet damage bonus, survival score,* dan *last survival bonus* yang dijalankannya. Robot Lawrie mampu menjaga jarak dari kerumunan musuh, memanfaatkan area yang relatif aman di tepi arena, dan tetap menyerang dari jarak menengah hingga jauh untuk mengumpulkan poin tanpa terlalu banyak menerima serangan balik.

Akan tetapi, 4 robot lainnya tidak berarti memiliki algoritma *greedy* yang tidak baik. Dalam beberapa kesempatan, mereka dapat meraih skor yang mendekati atau bahkan menyaingi bot Lawrie, terutama ketika posisi awal yang acak menempatkan mereka pada situasi yang menguntungkan.

Pada Robot Maniac, algoritma yang diimplementasikan berbasis *bullet damage, ram damage,* dan *ram damage bonus*. Fokus utama dari robot ini adalah mendapatkan skor dari serangan ofensif. Konsep dari robot ini adalah *high risk high return* yang berarti bahwa robot ini memiliki resiko yang tinggi karena terus menyerang dan kemungkinan mendapatkan skor sangat banyak apabila berhasil. Apabila robot berhasil mendapatkan target yang tepat, maka skor yang didapatkan sangatlah besar karena bersifat mengunci target. Akan tetapi, dalam penargetan ke bot lain memungkinkan robot ini mendapatkan banyak serangan dari bot lainnya karena bot Maniac hanya fokus terhadap satu bot hingga bot tersebut dinyatakan "mati".

Pada Robot Eits, basis sumber skor utama algoritma adalah *bullet damage, bullet damage bonus,* dan *survival score*. Jadi, robot ini mengutamakan menembak dari jarak aman agar tidak terkena serangan lawan. Selain itu, robot juga memiliki algoritma yang mendeteksi penurunan energi musuh yang mengindikasi musuh baru saja melakukan tembakan. Dalam menanggapi hal tersebut, bot akan melakukan manuver menghindar agar tidak terkena oleh tembakan. Pendekatan ini efektif menekan kerugian yang diterima, tetapi bot Eits bisa kesulitan jika robot agresif lain memaksanya mundur ke area dinding. Dalam kondisi tersebut, Eits bisa terjebak dan menerima serangan beruntun.

Pada Robot Lawrie, tujuan utama yang ingin dicapai adalah skor dari *bullet damage, survival score,* dan *last survival bonus.* Robot ini akan memprioritaskan untuk mencari tempat yang paling aman. Definisi dari paling aman adalah tidak berada di tengah-tengah arena dan merupakan titik terjauh dari kumpulan bot musuh. Dengan ini, robot ini berusaha agar tidak terperangkap dalam serangan-serangan lawan. Selain itu, robot ini juga mengimplementasikan algoritma menembak untuk mendapatkan skor di samping algoritma *survivalnya*. Namun, jika posisi acak

menempatkan Lawrie di tengah kerumunan lawan pada awal pertandingan, ia mungkin akan kesulitan menemukan “tempat aman” dengan cepat, sehingga bisa menerima serangan bertubi-tubi.

Pada Robot Union, robot ini akan mengutamakan hampir seluruh aspek yang ada. Dimulai dari skor *bullet damage, bullet damage bonus, ram damage, dan ram damage bonus, survival score,* hingga *last survival score*. Definisi robot ini adalah mengambil keuntungan sesaat dengan berbasis segala kemungkinan pendapatan skor bagi bot. Namun, karena cakupan strateginya luas, prioritas serangan atau pertahanan dapat menjadi kurang terfokus apabila tidak ditangani dengan baik.

Faktor peletakan acak setiap robot pada awal pertandingan juga memengaruhi hasil akhir. Penempatan awal di dekat lawan yang lemah dapat memberikan kesempatan bagi Robot Maniac untuk segera meraih skor besar, sementara penempatan Eits di dekat dinding dan musuh agresif justru menurunkan peluangnya untuk bertahan. Interaksi antarrobot pun turut membentuk dinamika yang tidak bisa sepenuhnya diprediksi, seperti ketika dua robot agresif saling beradu, membuka peluang bagi robot lain yang lebih defensif untuk menembak dari kejauhan dan bertahan lebih lama.

Secara keseluruhan, semua robot telah mengimplementasikan algoritma greedy sesuai dengan pendekatan masing-masing. Robot Lawrie berhasil memaksimalkan poin melalui keseimbangan antara penyerangan dan pertahanan, sehingga kerap kali unggul dalam lima kali pengujian. Meski begitu, setiap robot tetap memiliki potensi meraih skor tinggi jika faktor posisi awal dan interaksi di arena mendukung strategi yang dijalankan. Dengan demikian, strategi greedy yang diterapkan memang cukup efektif, namun tidak menjamin kemenangan absolut karena hasil akhir juga ditentukan oleh dinamika pertempuran yang kompleks dan bersifat acak.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Melalui pengerjaan tugas besar ini, kami telah berhasil mengimplementasikan algoritma greedy dalam pembuatan bot. Dalam tugas ini, kami menerapkan berbagai algoritma greedy seperti, mengincar musuh dengan cara mengejar, menembak, dan menabrak hingga bot musuh mati, menjaga jarak dari musuh dan memastikan dapat menghindar dari bullet musuh, serta memastikan bot berada dalam posisi yang aman sehingga tidak dapat dijangkau oleh bot musuh dengan mudah.

Bot yang dibuat kemudian diuji dengan pertandingan 1 vs 1 vs 1 vs 1. Hasil pengujian menunjukkan bahwa pendekatan greedy dapat menghasilkan strategi yang efektif dalam beberapa situasi, namun masih terdapat ruang untuk perbaikan terutama ketika menghadapi musuh yang menggunakan strategi canggih dan adaptif, yang memerlukan peningkatan dalam hal deteksi, penargetan, dan respons real-time terhadap perubahan kondisi pertempuran.

5.2 Saran

Dalam pengerjaan tugas besar ini, terdapat beberapa saran yang akan kami perhatikan untuk dikembangkan kedepannya, antara lain:

1. Mengeksplorasi lebih banyak dan mempelajari secara detail terkait algoritma greedy sebelum mulai mengimplementasikan sehingga pemilihan strategi dan parameter yang digunakan dapat lebih optimal.
2. Memastikan penamaan yang sama terkait nama variabel maupun nama fungsi yang mempunyai tugas yang sama sehingga memudahkan pembaca untuk memahami isi kode.
3. Mengembangkan kemampuan menyerang atau menghindari beberapa musuh sekaligus, terutama dalam situasi pertempuran dengan banyak bot.

BAB VI

LAMPIRAN

6.1 Daftar Pustaka

Munir, Rinaldi. 2025. "Algoritma Greedy (Bagian 1)" (https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/04-Algoritma-Greedy-(2025)-Bag1.pdf,, diakses 21 Maret 2025)

Munir, Rinaldi. 2025. "Algoritma Greedy (Bagian 2)" (https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/05-Algoritma-Greedy-(2025)-Bag2.pdf, diakses 21 Maret 2025)

Munir, Rinaldi. 2025. "Algoritma Greedy (Bagian 3)" (https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/06-Algoritma-Greedy-(2025)-Bag3.pdf, diakses 21 Maret 2025)

6.2 Tautan Repository

Link repository dari Tugas Besar 1 IF2211 Strategi Algoritma kelompok Cisa adalah sebagai berikut:

https://github.com/varel183/Tubes1_Cisa

6.3 Tautan Video

Link repository dari Tugas Besar 1 IF2211 Strategi Algoritma kelompok Cisa adalah sebagai berikut:

https://youtu.be/AJbqW1k4PN4