

Tugas Besar I IF2211 Strategi Algoritma

Pemanfaatan Algoritma Greedy Dalam Pembuatan Bot Permainan Robocode Tank Royale



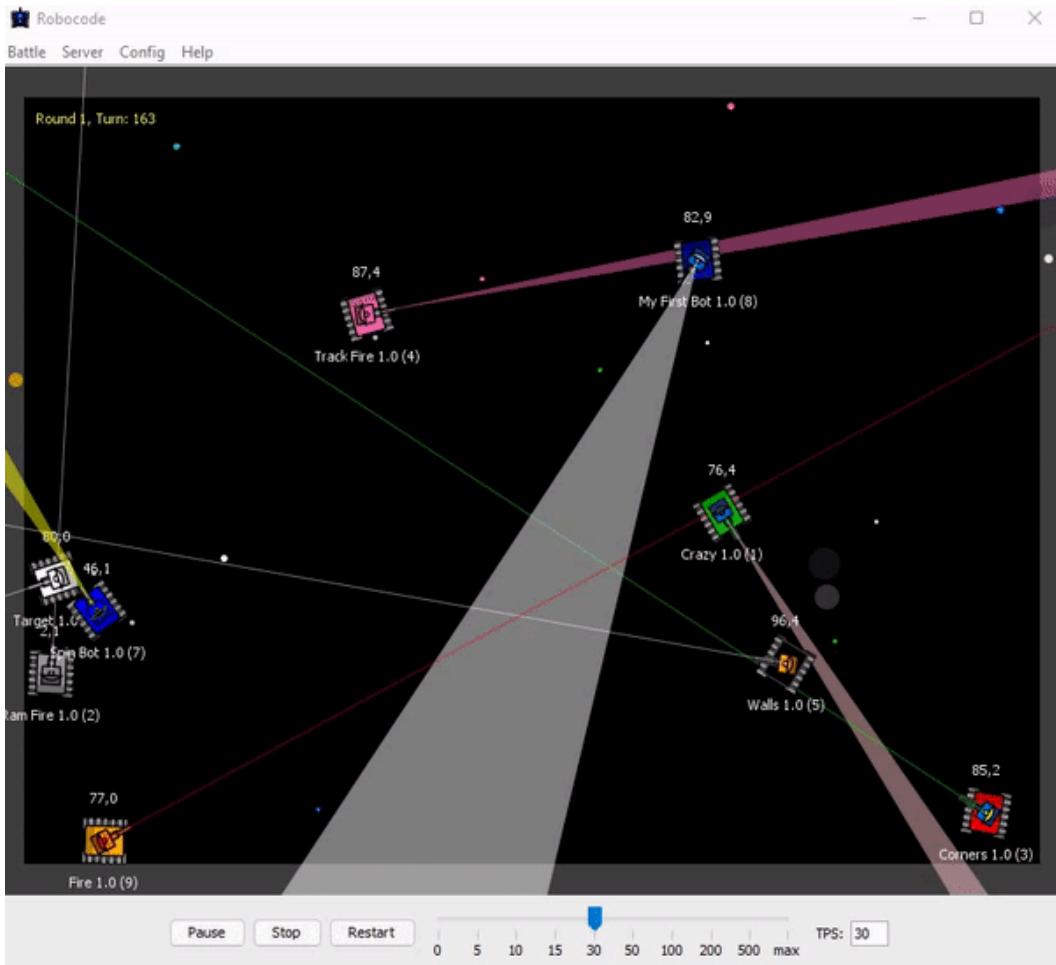
Disusun oleh:

Timothy Niels Ruslim (10123053)
Ghaisan Zaki Pratama (10122078)
William Gerald Briandelo (13222061)

**Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025**

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. 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 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.

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.

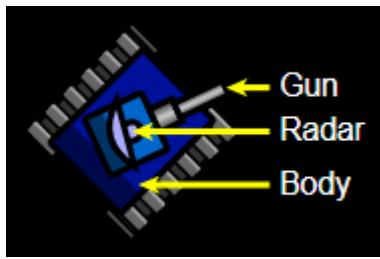
6. Tabrakan

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 penggereman 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.

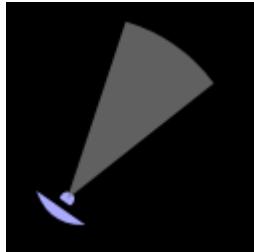
Bagian paling lambat adalah tubuh tank, yang dalam kondisi terbaik dapat berputar hingga 10 derajat per giliran. Namun, ini bergantung pada kecepatan bot saat ini. Semakin cepat bot bergerak, semakin lambat kemampuannya untuk berbelok.

Perlu diperhatikan bahwa tidak ada energi yang dikonsumsi saat bot bergerak atau berbelok.

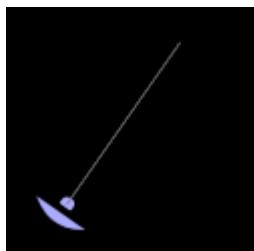
10. Pemindaian

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 lainnya 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.

Untuk informasi lebih lengkap, silahkan buka dokumentasi Tank Royale pada link [berikut](#).

Starter Pack

Untuk tugas besar ini, game engine yang akan digunakan sudah dimodifikasi oleh asisten. Berikut adalah beberapa perubahan yang dibuat oleh asisten dari game engine Tank Royale:

- **Theme GUI:** diubah menjadi light theme
- **Skor & Energi:** masing-masing bot ditampilkan di samping arena permainan agar lebih mudah diamati saat pertarungan
- **Turn Limit:** Durasi pertarungan tidak akan bergantung pada banyak ronde. Pada game engine ini, pertarungan akan berakhir apabila banyaknya turn sudah mencapai batas tertentu. Apabila batasan ini tercapai, ronde otomatis langsung berakhir dan pemenang pertarungan akan ditampilkan. Turn Limit dapat diatur pada menu “setup rules”.

Source Code untuk game engine dan template bot telah disediakan pada tautan berikut.

[tubes1-if2211-starter-pack](#)

Adapun panduan mengenai cara menjalankan game engine, membuat bot, dan melihat referensi API dapat dilihat melalui tautan berikut.

 Get Started With Robocode

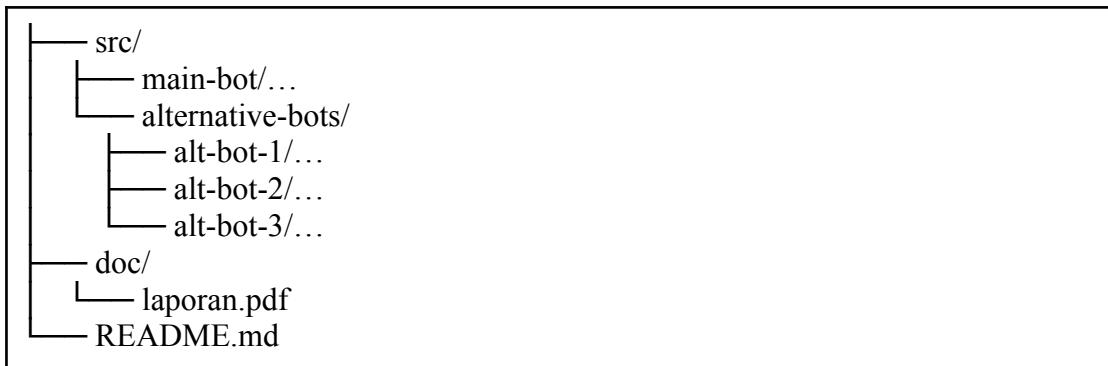
Spesifikasi Wajib

- Buatlah 4 bot (1 utama dan 3 alternatif) dalam bahasa **C# (.net)** yang mengimplementasikan **algoritma Greedy** pada *bot* permainan Robocode Tank Royale dengan tujuan memenangkan permainan.
- Tugas dikerjakan berkelompok dengan anggota **minimal 2 orang** dan **maksimal 3 orang**, boleh lintas kelas dan lintas kampus.
- Strategi *greedy* yang diimplementasikan setiap kelompok harus dikaitkan dengan fungsi objektif dari permainan ini, yaitu memperoleh skor setinggi mungkin pada akhir pertempuran. Hal ini dapat dilakukan dengan mengoptimalkan komponen skor yang telah dijelaskan diatas.
- Strategi *greedy* yang diimplementasikan **harus berbeda** untuk setiap bot yang diimplementasikan dan setiap strategi greedy harus menggunakan **heuristic** yang berbeda.
- **Bot yang dibuat TIDAK BOLEH sama dengan SAMPEL yang diberikan sebagai CONTOH.** Baik dari starter pack maupun dari repository engine asli.

- Buatlah strategi *greedy* terbaik, karena setiap **bot utama** dari masing-masing kelompok akan diadu dalam kompetisi Tubes 1.
- Strategi *greedy* yang kelompok anda buat harus **dijelaskan dan ditulis secara eksplisit** pada laporan, karena akan diperiksa saat demo apakah strategi yang dituliskan sesuai dengan yang diimplementasikan.
- Setiap kelompok dapat menggunakan kreativitas yang bermacam macam dalam menyusun strategi *greedy* untuk memenangkan permainan. Implementasi pemain **harus dapat dijalankan pada game engine** yang telah disebutkan diatas serta dapat dikompetisikan dengan bot dari kelompok lain.
- Program harus mengandung komentar yang jelas, dan untuk setiap strategi *greedy* yang disebutkan, harus dilengkapi dengan **kode sumber yang dibuat**. Artinya semua strategi harus diimplementasikan
- Mahasiswa disarankan membaca [dokumentasi](#) dari *game engine*. Perlu diperhatikan bahwa game engine yang digunakan untuk tubes ini **SUDAH DIMODIFIKASI**. Untuk Perubahan dapat dilihat pada bagian [starter pack](#)

Spesifikasi Bonus

- (maks 10) Membuat video tentang aplikasi *greedy* pada bot serta simulasinya pada game kemudian mengunggahnya di Youtube. Video dibuat harus memiliki audio dan menampilkan wajah dari setiap anggota kelompok. Untuk contoh video tubes stima tahun-tahun sebelumnya dapat dilihat di Youtube dengan kata kunci “Tubes Stima”, “strategi algoritma”, “Tugas besar stima”, dll. **Semakin menarik video, maka semakin banyak poin yang diberikan.**
- (maks 10) Beberapa kelompok pemenang lomba kompetisi akan mendapatkan nilai tambahan berdasarkan posisi yang diraih.
- Jika terdapat kesulitan selama mengerjakan tugas besar sehingga memerlukan bimbingan, maka dapat melakukan asistensi tugas besar kepada asisten (opsional). Dengan catatan asistensi hanya bersifat membimbing, bukan memberikan “jawaban”.
- Bot yang telah dibuat akan dikompetisikan dengan kelompok lain dan disaksikan oleh seluruh peserta kuliah. Terdapat hadiah menarik bagi kelompok yang memenangkan kompetisi.
- Setiap kelompok harap mengisi nama kelompok dan anggotanya pada link berikut, paling lambat **Kamis, 6 Maret pukul 22.11 WIB**.
 - [+ Pendataan Kelompok Tubes 1 Stima 2024/2025](#)
- Diwajibkan untuk memilih asisten meskipun tidak melakukan asistensi, karena asisten yang dipilih akan menjadi asisten saat asistensi (opsional) dan demo tugas besar. Pemilihan asisten dapat dilakukan pada link berikut, paling lambat **Kamis, 6 Maret 2025 pukul 22.11 WIB**.
 - [+ Pendataan Kelompok Tubes 1 Stima 2024/2025](#)
- Program disimpan dalam repository yang bernama **Tubes1_NamaKelompok** dengan nama kelompok sesuai dengan yang di sheets diatas. Berikut merupakan struktur dari isi repository tersebut:



- a. Folder src berisi semua source code
- b. Folder doc berisi laporan tugas besar dengan format **NamaKelompok.pdf** dengan ketentuan isi pada [isi laporan](#)
- c. README untuk tata cara penggunaan yang minimal berisi:
 - i. Penjelasan singkat algoritma greedy yang diimplementasikan untuk setiap bot yang dibuat
 - ii. Requirement program dan instalasi tertentu bila ada
 - iii. Command atau langkah-langkah dalam meng-compile atau build program
 - iv. Author (identitas pembuat)
- Sangat disarankan untuk menggunakan [**semantic commit**](#). Buatlah release dengan format **v1.x** dengan x adalah nomor revisi dimulai dari revisi 0. Contoh v1.0 untuk release pertama, v1.1 untuk revisi selanjutnya.
- Pastikan untuk membuat repository bersifat **Public** paling lambat **H+1 deadline (1 hari setelah deadline)**. Sebelum deadline repository harus bersifat **Private**.
- Laporan dikumpulkan hari **Senin, 24 Maret 2025** pada alamat Google Form berikut paling lambat **pukul 22.11 WIB**:
<https://bit.ly/tubes1stima25>.

PERINGATAN: Keterlambatan akan mengurangi nilai sebanyak 1 poin untuk setiap menit keterlambatan.
- Adapun pertanyaan terkait tugas besar ini bisa disampaikan melalui QnA berikut:
<https://bit.ly/QnA-Stima-25>.

BAB II

LANDASAN TEORI

2.1 Strategi *Greedy*

Strategi Greedy merupakan suatu strategi algoritma yang bertujuan untuk menyelesaikan persoalan dengan memilih solusi optimum lokal pada setiap langkahnya, dengan harapan bahwa pilihan-pilihan lokal tersebut akan menghasilkan solusi optimum global pada hasil akhirnya. Setiap menjalankan langkahnya, strategi greedy akan selalu memilih pilihan terbaik yang tersedia saat itu tanpa memikirkan kemungkinan terbaik yang mungkin timbul di masa depan. Setelah memilih pilihan tersebut, strategi greedy tidak akan kembali untuk mengubah pilihannya. Maka, tidak ada *backtracking* yang dilakukan pada algoritma greedy. Maka, untuk beberapa persoalan, strategi greedy tidak akan menghasilkan solusi optimum global.

Pada algoritma greedy, terdapat beberapa elemen yang harus didefinisikan sebagai berikut.

- a. Himpunan kandidat (C): Berisi pilihan-pilihan yang tersedia untuk dipilih pada setiap langkah.
- b. Himpunan solusi (S): Berisi kandidat yang sudah dipilih.
- c. Fungsi solusi (Solusi): Fungsi ini untuk menentukan apakah himpunan solusi yang dipilih sudah memberikan solusi pada persoalan.
- d. Fungsi seleksi (Seleksi): Fungsi ini untuk memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.
- e. Fungsi kelayakan (Layak): Fungsi ini untuk memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi.
- f. Fungsi objektif: Fungsi ini untuk meminimumkan atau memaksimumkan parameter pada suatu persoalan.

Berdasarkan istilah-istilah tersebut, dapat dirancang suatu framework greedy dari persoalan yang ingin ditangani, yaitu bahwa “Algoritma greedy melibatkan pencarian sebuah himpunan bagian, S , dari himpunan kandidat, C ; yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu S menyatakan suatu solusi dan S di-optimisasi oleh fungsi objektif” (Munir).

Adapun bahwa skema umum penyelesaian suatu persoalan adalah seperti pada *pseudocode* berikut.

```

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

Deklarasi
 $x : \text{kandidat}$ 
 $S : \text{himpunan}_\text{solusi}$ 

Algoritma:
 $S \leftarrow \{\}$  {inisialisasi  $S$  dengan kosong}
while (not SOLUSI( $S$ )) and ( $C \neq \{\}$ ) do
     $x \leftarrow \text{SELEKSI}(C)$  {pilih sebuah kandidat dari  $C$ }
     $C \leftarrow C - \{x\}$  {buang  $x$  dari  $C$  karena sudah dipilih}
    if LAYAK( $S \cup \{x\}$ ) then { $x$  memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi}
         $S \leftarrow S \cup \{x\}$  {masukkan  $x$  ke dalam himpunan solusi}
    endif
endwhile
{ $\text{SOLUSI}(S)$  or  $C = \{\}$ }

if SOLUSI( $S$ ) then {solusi sudah lengkap}
    return  $S$ 
else
    write('tidak ada solusi')
endif

```

Perhatikan bahwa program menggunakan `while` untuk mengambil kandidat-kandidat solusi dan mengecek apakah yang dipilih adalah suatu solusi sah. Lalu, pengambilan juga berdasarkan suatu heuristik yang sama terus pada setiap pengambilan.

2.2 Robocode Tank Royale

Robocode Tank Royale adalah sebuah permainan pemrograman yang mengharuskan pemain menulis kode untuk mengendalikan bot berbentuk tank yang bertarung di arena virtual. Pada permainan Robocode yang diujikan, digunakan bahasa pemrograman C# dalam pembuatan bot. Permainan ini menggabungkan elemen strategi dan kecerdasan buatan, sehingga setiap bot harus dibuat dengan logika untuk bergerak, mendeteksi lawan, dan menembak dengan tepat, semua tanpa interaksi langsung dari pemain selama pertempuran. Selain itu, versi Tank Royale mendukung pertarungan secara daring melalui jaringan, menjadikannya evolusi modern dari permainan Robocode klasik.

Cara kerja permainan ini mengikuti aturan yang telah ditetapkan: setiap tank memiliki sistem persenjataan (gun) dengan mekanisme pendinginan (gun heat) yang membatasi frekuensi penembakan, serta radar yang berfungsi untuk mendeteksi keberadaan lawan dengan rentang dan kecepatan putar tertentu. Pertempuran berlangsung dalam turn, di mana setiap bot mengirim perintah seperti bergerak, memutar badan, mengarahkan senjata, dan menembak. Perhitungan poin umumnya didasarkan pada kerusakan yang diberikan, kelangsungan hidup, dan bonus kemenangan.

Pada permainan Tank Royale terdapat berbagai Class yang disediakan dalam API Tank Royale, seperti.

| Nama | Deskripsi |
|---------------------------------|--|
| OnScannedBot(ScannedBotEvent) | Suatu penanganan kejadian ketika terdapat bot musuh yang terdeteksi radar. |
| Fire(double) | Menembakkan energi sesuai dengan arah tembakan dari tank (bot). |
| SetTurnLeft(double) | Mengatur arah gerak bot ke arah kiri. |
| SetTurnRight(double) | Mengatur arah gerak bot ke arah kanan. |
| SetTurnGunLeft(double) | Mengatur arah gerak tembakan/radar ke arah kiri. |
| SetTurnGunRight(double) | Mengatur arah gerak tembakan/radar ke arah kanan. |
| SetForward(double) | Mengatur gerak maju bot. |
| OnHitBot(HitBotEvent) | Suatu penanganan kejadian ketika bot menabrak bot musuh. |
| OnHitByBullet(HitByBulletEvent) | Suatu penanganan kejadian ketika bot terkena tembakan bot musuh. |
| OnHitWall(HitWallEvent) | Suatu penanganan kejadian ketika bot menabrak batas arena. |
| BearingTo(double, double) | Mengarahkan bot pada suatu posisi spesifik. |

2.3 Game Engine Robocode Tank Royale

Game engine Robocode Tank Royale bertanggung jawab untuk melakukan simulasi pertempuran bot secara *real-time* dengan pendekatan berbasis *turn*. Engine ini digunakan oleh pemain untuk memasukkan bot yang telah dirancang. Engine ini mengelola logika permainan, mulai dari pembaruan status setiap turn, penanganan event melalui event queue, hingga penentuan hasil perhitungan poin dan status bot. Selain itu, game engine mengimplementasikan aturan-aturan dalam permainan seperti gerakan, rotasi, kecepatan peluru, dan pendinginan senjata. Dengan dukungan komunikasi melalui WebSocket, engine ini memungkinkan pertempuran *online* oleh beberapa bot, serta menyediakan *interface* grafis yang menampilkan aksi bot secara *real-time*.

2.4 Penerapan Strategi Greedy pada Robocode Tank Royale

Permainan Robocode Tank Royale dapat menerapkan banyak strategi. Karena permainan melibatkan waktu berupa turns yang terus berjalan, strategi naive seperti *bruteforce* tidak mungkin dilakukan, karena *feedback* keoptimalan solusi tidak dapat langsung diketahui. Untuk banyak bots dalam game, terutama yang turn based, sering juga digunakan model berdasarkan *tree traversal* suatu *decision tree*, yang bahkan dapat dikembangkan menjadi suatu *finite state-machine* (Mount & Eastman). Dalam Robocode Tank Royale, ini berarti aksi yang dilakukan pada suatu turn dapat ditentukan oleh kondisional-kondisional yang kompleks berdasarkan informasi-informasi pada turn sebelumnya. Model pembelajaran mesin lainnya seperti reinforcement learning dan neural networks dapat dilakukan juga untuk melakukan prediksi untuk setiap turn. Tetapi, kerumitan solusi-solusi tersebut harus dipertanggung jawabkan.

Tetapi, solusi termudah untuk permasalahan memenangkan permainan Robocode Tank Royale adalah dengan strategi greedy. Secara praktis, ini melibatkan pemilihan gerakan radar, gun, dan body yang paling baik berdasarkan suatu heuristik pada suatu turn hanya berdasarkan informasi pada turn sebelumnya, tanpa memikirkan apakah gerakan tersebut akan memenangkan permainan di akhir. Maka, dengan menerapkan strategi greedy dalam permainan Robocode Tank Royale, bot memilih suatu solusi optimum *lokal* dalam setiap turn. Algoritma greedy bersifat lumayan sederhana namun dapat berupa sangat efektif dalam suatu permainan turn-based yang tidak dapat diterapkan *exhaustive search*. Oleh karena itu, strategi tersebut menjadi fokus utama dari tugas besar ini.

BAB III

APLIKASI STRATEGI *GREEDY*

3.1 Pemetaan *Greedy* di Bot

Tujuan utama dari permainan Robocode Tank Royale adalah mendapatkan poin terbanyak agar menang. Maka, secara abstrak, kita ingin memaksimalkan skor akhir bot menggunakan gerakan-gerakan yang dapat kita lakukan. Dalam bahasa strategi *greedy*, ini menghasilkan framework *meta-greedy* seperti pada tabel berikut.

| Komponen <i>Greedy</i> | Penerapan di Robocode Tank Royale |
|------------------------|--|
| Himpunan Kandidat | Seluruh gerakan dan rotasi yang dapat dilakukan oleh radar, gun, dan body bot. |
| Himpunan Solusi | Gerakan pada radar, gun, dan body bot yang dipilih pada suatu turn berdasarkan fungsi seleksi. |
| Fungsi Solusi | Pemeriksaan jumlah poin (atau peringkat) yang diraih pada akhir round akibat seluruh gerakan yang dipilih. |
| Fungsi Seleksi | Pemilihan gerakan yang mungkin berdasarkan informasi bot sendiri, bot lain, dan arena, juga suatu heuristik yang akan memaksimalkan poin. |
| Fungsi Kelayakan | Pemeriksaan apakah suatu gerakan dapat dilakukan pada suatu turn. Ini terjadi secara implisit, yaitu berdasarkan apakah program meng- <i>compile</i> bot dengan benar. |
| Fungsi Objektif | Jumlah poin di akhir permainan yaitu, $\text{BulletDamage} + \text{SurvivalScore} + \text{RamDamage}$, beserta bonus mereka ingin dimaksimalkan. |

Akan tetapi, biasanya heuristik untuk suatu fungsi seleksi sangat eksplisit dijelaskan untuk suatu strategi *greedy*. Misalnya, pada permasalahan penukaran uang, sangat eksplisit pemilihan geraknya adalah koin yang nilainya tertinggi. Tapi, strategi *meta-greedy* ini masih bersifat sangat *high-level*, karena heuristik untuk strategi suatu bot di Robocode Tank Royale tidak dapat dituliskan dalam suatu formula eksplisit. Jika menggunakan strategi alternatif lebih kompleks yang telah dibahas sebelumnya seperti machine learning, heuristik tersebut tidak bisa dikenali (seperti suatu *blackbox* saja). Untungnya, manfaat penggunaan strategi *greedy* untuk permainan Robocode Tank Royale ini adalah bahwa strategi tersebut dapat diterjemahkan menjadi suatu Bahasa Indonesia yang lebih dimengerti. Lalu, penerjemahan

tersebut memungkinkan kita untuk memecah heuristik yang dipilih menjadi sub-persoalan yang sendiri-sendiri dapat diselesaikan dengan strategi greedy juga!

Sebagai contoh, bot sampel `Corners` dapat dikatakan menerapkan strategi greedy dengan heuristik: “menembak secara acak pada musuh manapun yang terdeteksi saat saya sudah tiba di sudut arena.” Kemudian, heuristik tersebut, dan sebenarnya fungsi seleksi apapun yang dipilih, pasti harus membahas beberapa sub-persoalan berikut, yang sendiri-sendiri dapat diselesaikan dengan algoritma greedy.

1. Mengatur `firePower` yang ditembakkan bot.

Sebagai contoh, pemain dapat mengatur jumlah energi yang ditembakkan oleh bot berdasarkan jarak dengan musuh untuk memaksimalkan `BulletDamage`. Ini dapat menghasilkan strategi berikut.

| Komponen <i>Greedy</i> | Penerapan di Robocode Tank Royale |
|------------------------|---|
| Himpunan Kandidat | Selang kontinu $[0, \infty)$ <code>firePower</code> yang mungkin. |
| Himpunan Solusi | <code>FirePower</code> yang dipilih. |
| Fungsi Solusi | Cek apakah bot yang ditembak terkena bullet. |
| Fungsi Seleksi | Pilih <code>firePower</code> yang lebih besar dari biasa jika lebih dekat dengan suatu musuh. |
| Fungsi Kelayakan | Memastikan <code>firePower</code> dihitung hanya setelah mendeteksi musuh. |
| Fungsi Objektif | <code>BulletDamage</code> ingin dimaksimalkan. |

2. Mengatur pilihan gerak bot.

Sebagai contoh, pemain dapat menentukan pilihan gerak bot seperti untuk mengejar musuh jika bot memiliki energi lebih kecil dari energi bot sendiri. Ini dapat menghasilkan strategi berikut.

| Komponen <i>Greedy</i> | Penerapan di Robocode Tank Royale |
|------------------------|---|
| Himpunan Kandidat | Selang kontinu $[0, 360)$ arah untuk body dan $[0, \infty)$ jarak untuk maju. |
| Himpunan Solusi | Arah body dan jarak maju yang dipilih. |

| | |
|------------------|--|
| Fungsi Solusi | Cek apakah bot mendekati musuh. |
| Fungsi Seleksi | Pilih arah body dan jarak maju yang mendekati musuh. |
| Fungsi Kelayakan | Memastikan bot musuh memiliki energi lebih kecil dari sendiri sebelum bertindak. |
| Fungsi Objektif | Jarak dengan musuh ingin diminimalkan. |

3. Mengatur bot yang akan ditembak.

Sebagai contoh, pemain dapat menentukan pilihan bot musuh yang akan ditembak berdasarkan apakah bot cukup dekat (supaya peluang *miss* lebih kecil). Ini dapat menghasilkan strategi berikut.

| Komponen <i>Greedy</i> | Penerapan di Robocode Tank Royale |
|------------------------|--|
| Himpunan Kandidat | Semua bot yang masih hidup. |
| Himpunan Solusi | Bot dengan energi lebih kecil dari energi sendiri yang memiliki energi terkecil. |
| Fungsi Solusi | Cek apakah bot memiliki energi terkecil dari semua yang energinya lebih kecil dari energi sendiri. |
| Fungsi Seleksi | Pilih bot dengan energi lebih sedikit dari energi sendiri. |
| Fungsi Kelayakan | Memastikan bot musuh memiliki energi lebih sedikit dari energy sendiri. |
| Fungsi Objektif | Energi musuh yang dipilih ingin diminimalkan. |

Perhatikan bahwa sub-persoalan tersebut diselesaikan dengan suatu strategi *meso-greedy* yang walaupun fungsi objektifnya sendiri bukan pemaksimalan poin yang diinginkan, sebenarnya membantu heuristik bot untuk mencapai hal tersebut secara tidak langsung. Untuk setiap alternatif bot-bot berikut, ditentukan suatu heuristik khusus sebagai strategi greedy, yang walaupun solusi *meta-greedy*-nya sama, menerapkan beberapa strategi *meso-greedy* secara khusus pada sub-persoalan yang ada pada fungsi seleksinya.

3.2 ngegasTron

ngegasTron merupakan bot yang dibuat dengan menerapkan algoritma greedy, dengan strategi untuk menyerang musuh sembari menghindari musuh untuk meningkatkan

survivability. Dalam hal ini heuristik yang digunakan ada dua, yaitu Greedy terhadap Bullet Damage dan Greedy terhadap Survival Score.

3.2.1 Greedy Terhadap Bullet Damage (Firepower)

Pada bagian ini, bot akan menentukan jumlah energi yang ditembakkan berdasarkan jarak musuh yang terdeteksi serta energi (HP) dari bot. Hal ini bertujuan untuk membuat peluru dapat mencapai bot musuh dengan cukup cepat sehingga musuh tidak sempat menghindar, serta supaya energi dari bot tidak cepat habis. Untuk jumlah energi yang ditembakkan dihitung melalui rumus berikut.

$$Firepower = 5 - \frac{Distance}{200}$$

Namun, terdapat kondisi khusus ketika jumlah energi yang ditembakkan akan selalu konstan, yaitu ketika jarak musuh ≤ 30 dari bot maka akan ditembakkan energi sejumlah 75% dari energi bot yang tersisa. Selanjutnya, ketika energi bot < 30 maka akan ditembakkan energi sejumlah 1.

Berikut merupakan komponen greedy terhadap bullet damage pada bot ngegasTron.

| Komponen Greedy | Penerapan di Robocode Tank Royale |
|------------------------|--|
| Himpunan Kandidat | Selang kontinu $[0, \infty)$ firePower yang mungkin. |
| Himpunan Solusi | FirePower berdasarkan rumus seleksi. |
| Fungsi Solusi | Cek jarak dan apakah peluru kena musuh. |
| Fungsi Seleksi | Pilih firePower berdasarkan rumus di atas yang bergantung pada jarak dan rageFactor. |
| Fungsi Kelayakan | Memastikan bot target terdeteksi. |
| Fungsi Objektif | BulletDamage pribadi ingin dimaksimalkan. |

3.2.2 Greedy Terhadap Survival Score

Pada bagian ini, bot akan menentukan pilihan gerakan selanjutnya berdasarkan kondisi yang dialaminya saat itu. Bot akan menentukan pilihannya berdasarkan jumlah HP yang tersisa serta jarak musuh yang terdeteksi oleh bot. Pilihan yang dijadikan kandidat adalah bot yang fokus menembak musuh atau bot yang juga mengejar musuh untuk memperoleh HP.

Berikut merupakan komponen terhadap survivability pada bot ngegasTron.

| Komponen Greedy | Penerapan di Robocode Tank Royale |
|------------------------|--|
|------------------------|--|

| | |
|-------------------|---|
| Himpunan Kandidat | Jumlah energi yang ditembakkan bot berdasarkan jarak dan juga pilihan bot untuk mengejar musuh berdasarkan HP bot |
| Himpunan Solusi | Gerakkan bot berdasarkan kondisi yang telah ditetapkan. |
| Fungsi Solusi | Cek apakah energi (HP) bot bertambah |
| Fungsi Seleksi | Pilih tindakan bot untuk menanggapi musuh sesuai kondisi yang ditetapkan. |
| Fungsi Kelayakan | Memastikan bot dapat menyerang musuh untuk memperoleh energi |
| Fungsi Objektif | Memaksimalkan potensi perolehan energi |

3.2.3 Analisis Efisiensi dan Efektivitas ngegasTron

Bot ngegasTron menggunakan pendekatan greedy yang cukup efektif dalam menghadapi musuh dengan reaksi cepat dan agresif. Strategi greedy dari bot terlihat dari respons instan ketika mendeteksi musuh, bot akan segera menghentikan pergerakan, menyesuaikan posisi meriam dengan offset sederhana untuk mengantisipasi pergerakan musuh, dan menembak berdasarkan jarak serta kondisi energi secara langsung. Pendekatan ini memungkinkan bot untuk memanfaatkan peluang serangan seketika tanpa menunggu perhitungan jangka panjang, yang mengoptimalkan respons di situasi yang dinamis.

Namun, efisiensi dari bot juga memiliki keterbatasan, karena tidak melibatkan prediksi mendalam atau analisis situasi strategis yang lebih kompleks, bot kurang optimal saat menghadapi lawan yang lebih adaptif atau ketika perlu mempertimbangkan manuver bertahan dan pengaturan posisi secara strategis. Secara keseluruhan, pendekatan greedy memberikan keunggulan reaktivitas yang tinggi tetapi mungkin mengorbankan efektivitas jangka panjang dalam skenario pertempuran yang lebih kompleks.

3.3 DonatMerah

DonatMerah merupakan bot yang dibuat dengan menerapkan algoritma greedy. Dalam hal ini, heuristik khusus yang digunakan ada dua.

1. Pertama, Bot Donat Merah Greedy terhadap lintasan yang dilalui → bot menghitung jarak ke empat titik di tepi lingkaran ($0^\circ, 90^\circ, 180^\circ, 270^\circ$) berdasarkan pusat arena dan radius yang diinginkan. Bot akan menuju ke titik terdekat dari keempat titik tersebut. Bot akan bergerak membentuk lintasan lingkaran. Selama pergerakan ada

- faktor koreksi agar bot dapat stabil membentuk lingkaran dengan radius yang diinginkan dan bergerak secara tangensial. Bot tetap melakukan scan dan menembak.
2. Kedua, Bot DonatMerah Greedy terhadap Firepower (seberapa besar bullet damage) dan Ramming → bot akan memberikan firepower bergantung terhadap jarak antara bot dengan bot lawan. Jika bot lawan berada di jarak yang kurang dari 2 kali radius yang diinginkan maka bot akan menembak dengan firepower besar. Sebaliknya, maka bot akan menembak dengan firepower yang tidak besar. Penembakan ini terus dilakukan bersamaan dengan bot yang bergerak melintasi lintasan lingkaran. Bot juga akan mencoba menabrak bot lawan yang memiliki energy kurang dari 20 ketika berhasil di scan, kemudian akan kembali mengikuti lintasan lingkaran.

Dengan strategi ini, diharapkan bot dapat bertahan dengan terus membuat lintasan lingkaran bersamaan dengan menembak bot lain yang ada di sekitarnya.

3.3.1 Greedy Gerak Melingkar

DonatMerah akan fokus untuk bergerak melingkar dengan pusat lingkarannya ialah pusat arena dengan radius setengah kali minimum antara lebar dan tinggi arena. Saat bergerak melingkar bot fokus untuk menembak sebanyak mungkin bot lain yang berada di daerah sapuan radarnya.

Berikut merupakan pemetaan menjadi elemen-elemen algoritma greedy

| Komponen Greedy | Penerapan di Donat Merah |
|------------------------|--|
| Himpunan Kandidat | Titik-titik yang ada pada tepi lintasan lingkaran (empat titik dengan sudut 0° , 90° , 180° , 270°) yang merupakan opsi reposisi ketika bot menyimpang atau terjebak. |
| Himpunan Solusi | Pilihan titik dengan jarak terdekat dari posisinya saat itu sebagai titik awal lintasan lingkaran. |
| Fungsi Solusi | Pemeriksaan apakah bot sudah berhasil mencapai titik reposisi yang diinginkan yaitu pada salah satu tepi lingkaran. Selain itu, pemeriksaan apakah bot terus berada dalam lintasan lingkaran. |
| Fungsi Seleksi | Menghitung jarak antara posisi bot saat ini dengan masing-masing kandidat titik tepi lingkaran dan memilih titik dengan jarak terpendek. |
| Fungsi Kelayakan | Kandidat titik dipilih hanya jika jaraknya sesuai dengan rentang toleransi yang diinginkan (perbedaan dengan radius kurang dari suatu nilai toleransi radius). |

| | |
|-----------------|---|
| Fungsi Objektif | Maksimalkan survivability, bot harus tetap berada pada lintasan lingkaran dan menghindari dinding sehingga peluang untuk bertahan hidup meningkat (tidak mudah dideteksi dan dikejar bot lawan). Sehingga bot mendapat poin survive sampai akhir round. |
|-----------------|---|

3.3.2 Greedy Penembakan dan Penabrakan

DonatMerah akan menembak dengan firepower ditentukan dari jaraknya ke bot lawan. Batas yang ditentukan ialah 2 kali radius atau dengan kata lain seperti daerah lingkaran yang berpusat pada DonatMerah dengan jari-jari 2 kali radius.

Berikut merupakan pemetaan menjadi elemen-elemen algoritma greedy

| Komponen <i>Greedy</i> | Penerapan di Donat Merah |
|------------------------|--|
| Himpunan Kandidat | Nilai firepower antara 1 dan 3 yang dipilih berdasarkan jarak radius dari bot ke bot lain. |
| Himpunan Solusi | Nilai firepower yang dipilih yaitu 3 saat jaraknya kurang dari 2 kali radius atau 1 saat jaraknya lebih dari 2 kali radius. |
| Fungsi Solusi | Pemeriksaan apakah bot sudah menembak bot lawan dengan nilai firepower yang sesuai. |
| Fungsi Seleksi | Memilih nilai firepower 3 jika jarak target kurang dari 2 kali radius (seperti lingkaran yang berpusat di bot) atau memilih nilai firepower 1 jika jarak target lebih dari 2 kali radius. Strategi tersebut diterapkan dengan asumsi bahwa menembak dengan firepower tinggi pada target yang dekat akan menghasilkan damage maksimal, sedangkan untuk target jauh firepower yang lebih rendah karena akurasi tembakan yang tidak terlalu baik. |
| Fungsi Kelayakan | Cek apakah target apakah memiliki energi yang ada dibawah batas tertentu. |
| Fungsi Objektif | Maksimalkan bullet damage, bot memilih firepower secara optimal berdasarkan jarak dengan harapan menghasilkan damage yang tinggi pada target ketika peluang terkena lebih besar (jaraknya dekat). Sehingga, poin akan didapat dari banyaknya bullet damage yang berhasil kena ke bot lawan. |

3.3.3 Analisis Efisiensi dan Efektivitas DonatMerah

Strategi Greedy pada bot DonatMerah dapat dikatakan kurang efektif terutama pada penembakan dan penabrakan. Hal tersebut disebabkan karena tidak adanya proses

memprediksi gerakan bot lawan dengan lebih presisi agar tembakan dapat lebih banyak mengenai bot lawan, selain itu proses penabrakan masih kurang menonjol karena bot lawan yang dapat mudah memprediksi gerakan menabrak dari DonatMerah. Strategi Greedy gerak melingkar masih belum efisien karena proses koreksi radius masih kurang mulus sehingga masih ada kemungkinan bot tidak bergerak dengan radius konstan tetapi error untuk radiusnya sendiri dipaksa untuk kecil sehingga muncul masalah lain seperti bot kebingungan akan bergerak berlawanan arah jarum jam atau searah jarum jam (hal ini pula disebabkan menabrak dengan dinding di bagian bawah).

3.4 Avenger

Bot Avenger adalah konsep suatu bot yang menetapkan heuristik berbasis “balas dendam”. Pada dasarnya, strategi ini akan berlari-lari mengelilingi arena dengan berupaya untuk menjauhi bot yang dideteksinya. Tetapi, jika bot ditembak atau ditabrak musuh, dia akan menjadi agresif dan berupaya untuk membunuh bot yang memicunya itu. Ini menghasilkan beberapa subpersoalan *greedy*.

3.4.1. Gerakan Lari

Strategi Avenger ini akan bergerak menjauhi bot yang dideteksinya untuk memaksimalkan SurvivalScore, bahkan jika suatu jalur yang penuh bot-bot lain adalah alternatif yang lebih baik untuk menghindar tembakan. Secara praktis ini berarti memutarkan body ke arah terbalik dari musuh yang dideteksi. Ini menghasilkan sistem greedy seperti berikut.

| Komponen <i>Greedy</i> | Penerapan di Robocode Tank Royale |
|------------------------|--|
| Himpunan Kandidat | Selang kontinu [0, 360) arah untuk SetTurnRadarLeft. |
| Himpunan Solusi | Arah SetTurnRadarLeft yang dipilih berlawanan arah dengan musuh. |
| Fungsi Solusi | Cek apakah bot menjauhi musuh. |
| Fungsi Seleksi | Pilih arah SetTurnRadarLeft yang berlawanan arah dengan musuh. |
| Fungsi Kelayakan | Memastikan musuh terdeteksi sebelum berpindah arah, dan bahwa bot belum memiliki target. |
| Fungsi Objektif | Jarak dengan musuh ingin dimaksimalkan. |

3.4.2 Balas Dendam

Strategi Avenger ini juga akan bergerak membunuh suatu target yang telah memicunya dengan menembak atau menabrak. Pemilihan target ini sendiri juga bersifat greedy, karena bisa saja jika tujuannya untuk memaksimalkan bullet damage atau ram damage, lebih baik untuk memilih target berdasarkan energi mereka, atau kedahuluan deteksi mereka. Ini menghasilkan strategi greedy berikut.

| Komponen Greedy | Penerapan di Robocode Tank Royale |
|-------------------|--|
| Himpunan Kandidat | Semua bot yang masih hidup. |
| Himpunan Solusi | Musuh baru yang menabrak atau menembak bot. |
| Fungsi Solusi | Cek apakah bot telah menembak atau menabrak. |
| Fungsi Seleksi | Pilih bot yang telah menembak atau menabraknya. |
| Fungsi Kelayakan | Memastikan bot belum memiliki target sebelumnya. |
| Fungsi Objektif | Bullet Damage pribadi ingin dimaksimalkan, sedangkan punya musuh ingin diminimalkan (agar menang). |

3.4.3 Fire Power

Terakhir, strategi Avenger juga akan menerapkan skema *greedy* terhadap fire power. Ini bertujuan untuk memaksimalkan bullet damage, tetapi juga dengan tidak terlalu boros. Secara spesifik, digunakan rumus sebagai berikut.

$$\text{firePower} = 4 * \exp(-\text{targetDistance} / (250 + \text{rageFactor}))$$

Perhatikan bahwa `firePower` bergantung negatif eksponensial dengan jarak terhadap musuh. Ini agar bullet damage lebih besar saat peluang bullet akan kena musuh lebih tinggi, yaitu saat ditembak dari jarak yang lebih dekat. Adapun suatu `rageFactor` yang akan membuat `firePower` lebih besar. Kapan ini digunakan? `RageFactor` akan ditingkatkan setiap kali bullet yang ditembak bot berhasil kena musuh, seolah-olah “mengomporkan” amarah bot, sehingga menghasilkan bullet damage yang lebih besar. Berikut skema *greedy*-nya.

| Komponen Greedy | Penerapan di Robocode Tank Royale |
|-------------------|---|
| Himpunan Kandidat | Selang kontinu $[0, \infty)$ <code>firePower</code> yang mungkin. |
| Himpunan Solusi | <code>FirePower</code> berdasarkan rumus seleksi. |
| Fungsi Solusi | Cek jarak dan apakah peluru kena musuh. |

| | |
|------------------|---|
| Fungsi Seleksi | Pilih <code>firePower</code> berdasarkan rumus di atas yang bergantung pada jarak dan <code>rageFactor</code> . |
| Fungsi Kelayakan | Memastikan bot target terdeteksi. |
| Fungsi Objektif | <code>BulletDamage</code> pribadi ingin dimaksimalkan. |

3.4.4 Analisis Efisiensi dan Efektivitas Avenger

Pendekatan *greedy* dapat efektif dalam situasi di mana respons cepat terhadap ancaman langsung diperlukan, seperti saat bot diserang dan perlu segera membala untuk mengurangi risiko lebih lanjut. Dengan fokus pada target yang baru saja menyerang, bot dapat memanfaatkan informasi terbaru tentang posisi dan perilaku musuh, meningkatkan peluang keberhasilan serangan balik.

Namun, pada sisi efisiensi, meskipun respons cepat dapat mengurangi waktu komputasi dan penyederhanaan logika keputusan, pendekatan *greedy* memiliki keterbatasan dalam skenario yang lebih kompleks. Bot mungkin mengabaikan musuh lain yang lebih berbahaya atau peluang strategis lain karena terlalu fokus pada target saat ini. Selain itu, tanpa mempertimbangkan konteks yang lebih luas, bot dapat terjebak dalam pola perilaku yang dapat dieksploitasi oleh musuh yang lebih adaptif.

3.5 Asteroid Destroyer

Bot Asteroid Destroyer adalah suatu konsep strategi *greedy* yang melibatkan banyak strategi-strategi sebelumnya. Secara kasar, strategi melibatkan pencarian suatu target musuh, lalu target tersebut akan dikejar dan ditembak. Adapun bahwa pemilihan target adalah yang pertama dideteksi, lalu target bisa berubah jika terdeteksi musuh dengan energi yang lebih kecil. Berikutnya, pengejaran juga melibatkan *strafing* saat bot sudah dekat dengan target, yang berupa melingkari target agar sulit ditembak balik. Saat target sudah memiliki energi cukup kecil, bot juga akan menabraknya dalam upaya untuk mendapatkan ram bonus damage. Terakhir, bot juga akan menembak bullet dengan arah berdasarkan prediksi linier.

3.5.1 Pemilihan Target

Strategi Asteroid Destroyer selalu agresif. Strategi ini memilih bot pertama yang dideteksi sebagai musuh awal, lalu secara iteratif akan menggantikannya dengan bot dengan energi

yang lebih rendah. Jika bot target sudah mati, skema ini diulang. Ini menghasilkan strategi *greedy* untuk pemilihan target berdasarkan heuristik energi dan kedahuluan.

| Komponen Greedy | Penerapan di Robocode Tank Royale |
|------------------------|---|
| Himpunan Kandidat | Semua bot yang masih hidup. |
| Himpunan Solusi | Musuh yang pertama dideteksinya, atau musuh dengan energi yang lebih kecil jika sudah ada target. |
| Fungsi Solusi | Cek apakah ada bot lain yang di-scan dengan energi lebih sedikit. |
| Fungsi Seleksi | Ambil bot yang di-scan atau menabraknya dengan energi lebih kecil dari target terkini. |
| Fungsi Kelayakan | Memastikan apakah sudah memiliki target dan apakah bot yang di-scan memiliki energi lebih kecil. |
| Fungsi Objektif | Bullet Damage pribadi ingin dimaksimalkan. |

3.5.2 Pengejaran Target

Strategi Asteroid Destroyer juga mengejar target yang telah dipilih secara langsung. Ini berarti dia menoleh ke arah target lalu maju. Tetapi, jika bot sudah lumayan dekat dengan targetnya, selain mengikutinya langsung, bot akan melakukan *strafing* yang berupa bergerak melingkar targetnya. Ini supaya bot lebih sulit ditembak balik. Adapun bahwa jika target sudah memiliki energi yang kecil, bot sebaiknya menabrak untuk mendapatkan ram damage dan bonusnya juga, yakni untuk berhenti *strafing*. Ini menghasilkan strategi *greedy* untuk pengejaran target berdasarkan heuristik jarak dan energi.

| Komponen Greedy | Penerapan di Robocode Tank Royale |
|------------------------|--|
| Himpunan Kandidat | Selang kontinu $[0, 360)$ arah untuk body dan $[0, \infty)$ jarak untuk maju. |
| Himpunan Solusi | Arah body dan jarak maju yang dipilih. |
| Fungsi Solusi | Cek apakah bot <i>strafe</i> atau <i>chase</i> berdasarkan jarak dan energi. |
| Fungsi Seleksi | Pilih arah body dan jarak maju yang mendekati musuh, tetapi jika musuh dekat ambil arah yang tegak lurus dengan arah ke musuh. |
| Fungsi Kelayakan | Memeriksa apakah bot memiliki target yang sedang di-scan. |

| | |
|-----------------|--|
| Fungsi Objektif | Jarak dengan musuh ingin diminimalkan, tetapi juga energi pribadi ingin dipertahankan. |
|-----------------|--|

3.5.3 Fire Power

Strategi Asteroid Destroyer menghitung fire power dari bullet berdasarkan tiga faktor berdasarkan rumus berikut.

$$\text{FirePower} = (\text{DistanceFactor} + \text{SpeedFactor}) * \text{EnergyFactor}$$

Akan dijelaskan satu-satu.

1. Ada faktor jarak antara bot dengan target yang mirip dengan bot Avenger tetapi tanpa *rage factor*. Dia bersifat negatif eksponensial.

$$\text{DistanceFactor} = 3 * \exp(-\text{TargetDistance} / 250)$$

2. Ada faktor kecepatan target. Lebih lambat musuh, lebih besar *fire power*. Ini karena target yang lebih lambat lebih mudah ditembak, sehingga peningkatan *fire power* akan bermanfaat. Dia bersifat linier.

$$\text{SpeedFactor} = 1 - (\text{TargetSpeed} / 8)$$

3. Ada faktor energi pribadi dari Asteroid Destroyer yang bersifat multiplikatif. Ini agar bot tidak akan menghabiskan terlalu banyak energi untuk menembak saat energi bot sudah sedikit. Dia bersifat negatif eksponensial.

$$\text{EnergyFactor} = 1 - \exp(-\text{MyEnergy} / 25)$$

Ini menghasilkan strategi greedy untuk *fire power* seperti berikut.

| Komponen Greedy | Penerapan di Robocode Tank Royale |
|-------------------|---|
| Himpunan Kandidat | Selang kontinu $[0, \infty)$ <i>firePower</i> yang mungkin. |
| Himpunan Solusi | <i>FirePower</i> berdasarkan rumus seleksi. |
| Fungsi Solusi | Cek kesesuaian <i>firePower</i> dengan jarak, kecepatan, dan energi. |
| Fungsi Seleksi | Pilih <i>firePower</i> berdasarkan rumus di atas yang bergantung pada jarak, kecepatan, dan energi. |
| Fungsi Kelayakan | Memastikan bot target terdeteksi. |
| Fungsi Objektif | BulletDamage pribadi ingin dimaksimalkan. |

3.5.2 Prediksi Linier

Terakhir, strategi Asteroid Destroyer melibatkan prediksi linier saat ingin menembak bullet kepada target. Prediksi tersebut menggunakan sifat iteratif mirip Metode Newton seperti rumus rekuren berikut.

$$d_n = \sqrt{(X_n - X)^2 + (Y_n - Y)^2}$$

$$t_n = \frac{d_n}{v_b}$$

$$X_{n+1} = X_0 + \cos(\theta) \cdot S \cdot t_n$$

$$Y_{n+1} = Y_0 + \sin(\theta) \cdot S \cdot t_n$$

Di sini, d_n adalah jarak, X dan Y adalah posisi target, X_n dan Y_n adalah posisi prediksi target, θ sudut dengan target, v_b adalah kecepatan bullet yang ditembak, dan S kecepatan target.

Penggunaan strategi ini mengasumsikan bahwa target selalu bergerak secara linier. Sering bot tidak bergerak secara linier, sehingga strategi *greedy* terhadap gerakan linier tidak selalu menghasilkan prediksi paling optimal. Tetapi, harapannya adalah bahwa untuk waktu yang cukup kecil (yaitu 1 turn), asumsi ini cukup bagus. Strategi ini bertujuan untuk memaksimalkan Bullet Damage pribadi dengan meminimalkan *missed shots*.

| Komponen <i>Greedy</i> | Penerapan di Robocode Tank Royale |
|------------------------|---|
| Himpunan Kandidat | Selang kontinu $[0, \infty)$ untuk posisi X dan Y target. |
| Himpunan Solusi | Posisi yang diprediksikan secara linier. |
| Fungsi Solusi | Cek bahwa selisih prediksi ke- n dan $n + 1$ sudah cukup kecil. |
| Fungsi Seleksi | Pilih prediksi berdasarkan relasi rekuren di atas. |
| Fungsi Kelayakan | Pastikan target telah di-scan agar mendapatkan posisi dan kecepatan target. |
| Fungsi Objektif | Bullet Damage pribadi ingin dimaksimalkan. |

3.5.4 Analisis Efisiensi dan Efektivitas Asteroid Destroyer

Bot Asteroid Destroyer menunjukkan efektivitas yang tinggi melalui pendekatan greedy yang digunakan, terutama karena bot mampu merespons dan menargetkan musuh secara cepat. Saat mendeteksi musuh lain, bot akan segera menetapkan target, yang diutamakan

berdasarkan perbandingan energi, serta menghitung jarak dan menggunakan metode prediksi linear iteratif untuk memperkirakan posisi musuh. Kombinasi berbagai strategi tersebut, bersama dengan pergantian mode secara dinamis antara *chasing* dan *strafing*, memungkinkan bot untuk menyerang secara agresif dan efektif dalam situasi pertempuran yang berubah dengan cepat.

Pendekatan greedy yang digunakan menghasilkan pengambilan keputusan yang cepat dan ringan secara komputasi, karena hanya melibatkan perhitungan matematika sederhana untuk menentukan arah dan kecepatan. Namun, efisiensi bot juga terbatas karena bot cenderung terfokus pada parameter target yang telah ditetapkan, yaitu energi, tanpa mempertimbangkan konteks yang lebih luas. Akibatnya, meskipun bot ini responsif dan cepat dalam situasi pertempuran langsung, bot mungkin kurang optimal dalam menghadapi situasi strategis yang kompleks atau lawan yang memiliki taktik adaptif, yang dapat mengakibatkan penggunaan sumber daya secara suboptimal dalam jangka panjang.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Strategi Pada Bot

Strategi *greedy* yang telah dibahas, beserta ide-ide heuristik khusus yang dicetuskan diimplementasikan pada bot tank di Robocode Tank Royale menggunakan bahasa C#.

4.1.1 Implementasi Strategi Pada Bot ngegasTron

Berikut implementasi strategi ngegasTron di Robocode Tank Royale.

4.1.1.1 Pseudocode

Procedure ResetState()

Kamus Lokal:

paused : boolean
enemyDetected : boolean
movingForward : boolean

Algoritma:

paused \leftarrow false
enemyDetected \leftarrow false
movingForward \leftarrow true

Procedure AimTarget(targetX, targetY)

Kamus Lokal:

bearing : real

Algoritma:

bearing \leftarrow bearingTo(targetX, targetY)
if (bearing \geq 0) then
 turnGunLeft(bearing)
else
 turnGunRight(abs(bearing))
endif

WaitFor(new TurnCompleteCondition(this)) //menunggu hingga gerakan sebelumnya selesai

Procedure StopMovement()*Kamus Lokal:**Algoritma:*

stop()

Procedure ResumeMovement()*Kamus Lokal:**Algoritma:*

paused ← false

enemyDetected ← false

Procedure ReverseDirection()*Kamus Lokal:*

movingForward : boolean

Algoritma:

if (movingForward = true) then

setBack(40000)

movingForward ← false

else

setForward(40000)

movingForward ← true

endif

Procedure OnScannedBot(event)*Kamus Lokal:*

distance : real

targetX, targetY : real (diambil dari event)

a, b : real // konstanta untuk mengatasi pergerakan musuh dan lokasi penembakan awal yang tidak sesuai

Algoritma:

paused ← true

enemyDetected ← true

StopMovement()

SetTurnGunLeft(-12)

If (targetX>=0) then

a ← 6

else

a ← -6

endif

```

If (targetY>=0) then
    b ← 6
else
    b ← -6
endif

distance ← distanceTo(targetX, targetY)
if (distance ≤ 30) then
    AimTarget(targetX, targetY)
    WaitFor(new TurnCompleteCondition(this)) //menunggu hingga gerakan sebelumnya
selesai
    fire(energy * 0.75)
    WaitFor(new TurnCompleteCondition(this)) //menunggu hingga gerakan sebelumnya
selesai

else if ((energy ≥ 30) and (distance ≤ 1000)) then
    AimTarget(targetX, targetY)
    WaitFor(new TurnCompleteCondition(this)) //menunggu hingga gerakan sebelumnya
selesai
    fire( 5 - (distance / 200))
    WaitFor(new TurnCompleteCondition(this)) //menunggu hingga gerakan sebelumnya
selesai

else if (energy < 30) then
    AimTarget(targetX, targetY)
    WaitFor(new TurnCompleteCondition(this)) //menunggu hingga gerakan sebelumnya
selesai
    fire(1)
    setForward(200)
    WaitFor(new TurnCompleteCondition(this)) //menunggu hingga gerakan sebelumnya
selesai
endif

ResumeMovement()

```

Procedure OnHitWall()

Kamus Lokal:

Algoritma:

ReverseDirection()

Procedure OnHitBot(event)

Kamus Lokal:

Algoritma:

if (event.isrammed = true) then

```
    ReverseDirection()  
endif
```

Procedure Run()

Kamus Lokal:

 movingForward, enemyDetected, paused : boolean

Algoritma:

```
// inisialisasi warna dan state awal  
set bodyColor, turretColor, radarColor, bulletColor, scanColor sesuai keinginan  
movingForward ← true  
while (bot is running) do  
    if ((enemyDetected = false) and (paused = false)) then  
        setTurnLeft(5)  
        setTurnGunLeft(5)  
        setForward(300)  
    endif  
    go()          // eksekusi perintah yang tertunda  
endwhile
```

Main Program

Kamus Lokal:

Algoritma:

```
ResetState() pada saat gamestarted dan roundstarted  
ngegasTron_Run()
```

4.1.1.2 Struktur Data

Pada bot ngegasTron, digunakan struktur data sebagai berikut.

- a. bool movingForward, berfungsi untuk menyimpan status gerak maju bot.
- b. bool enemyDetected, berfungsi untuk menyimpan status apakah ada musuh yang terdeteksi radar atau tidak.
- c. bool paused, berfungsi untuk menghentikan pergerakan *default* bot karena terjadi *event* tertentu atau ada musuh yang terdeteksi.
- d. Selanjutnya, digunakan atribut bot sebagai properti warna dari bot.
- e. Class TurnCompleteCondition, merupakan kelas yang berguna untuk menunggu hingga *turn* selesai dan melakukan perintah selanjutnya.

4.1.1.3 Fungsi

Fungsi utama dan Konstruktor:

- a. ngegasTron() (Konstruktor)

Memanggil konstruktor basis untuk melakukan inisialisasi bot dengan informasi dari file konfigurasi (.json).

- b. Test()

Mengecek apakah *turn* bot telah selesai. Fungsi ini mengembalikan nilai true jika sisa perputaran bot (bot.TurnRemaining) sama dengan 0

4.1.1.4 Prosedur

Bot ngegasTron, memiliki prosedur sebagai berikut.

1. Main()

Memulai program dan membuat instance bot serta memulai eksekusi perintah dengan memanggil Start()

2. ResetState()

Mengatur ulang nilai variabel status (paused, enemyDetected, movingForward) ke kondisi awal setiap permainan atau ronde dimulai.

3. ResumeMovement()

Mengatur ulang status agar bot dapat melanjutkan gerakannya setelah mengalami *event* interaksi dengan musuh.

4. AimTarget(double x, double y)

Menghitung sudut yang diperlukan supaya senjata bot dapat mengarah pada koordinat target. Fungsi ini memanfaatkan fungsi BearingTo yang tersedia pada API robocode.

5. StopMovement()

Menghentikan gerakan bot dengan memanfaatkan fungsi Stop().

6. ReverseDirection()

Mengubah arah gerak bot jika bot menabrak *wall* atau bot musuh.

7. OnScannedBot()

Menangani kejadian ketika terdapat bot musuh yang terdeteksi. Tindakan yang akan diambil oleh bot sesuai dengan kondisional yang ditentukan.

8. OnHitWall()

Menangani kejadian ketika bot menabrak batas arena.

9. OnHitBot()

Menangani kejadian ketika bot menabrak musuh.

10. Run()

Menangani warna bot, gerakan yang dilakukan bot pada kondisi awal, serta melakukan eksekusi terhadap perintah yang telah di set melalui method Go().

Selanjutnya, bot ini memiliki alur algoritma sebagai berikut.

1. Inisialisasi dan Setup

Pada saat memulai permainan atau ronde, maka fungsi ResetState() akan dipanggil, sehingga setiap variabel status berada pada *initial condition*.

2. Loop Utama

Pada loop utama, selama bot aktif, maka program akan mengecek variabel status (paused dan enemyDetected), jika kedua variabel tersebut bernilai *false*, maka bot akan menjalankan gerakan defaultnya, yaitu bergerak lurus dengan sedikit berbelok ke kiri serta selalu memutar radar.

3. Penanganan event OnScannedBot

Ketika bot musuh terdeteksi pada radar, maka variabel status (paused dan enemyDetected) di set bernilai *true*, selanjutnya bot akan menjalankan tiga kemungkinan gerakan berdasarkan kondisi yang ditetapkan, yaitu.

- a. Jarak ≤ 30

Pada kondisi ini, bot akan menembakkan energi sejumlah 75% dari energi totalnya kepada musuh.

- b. Jarak ≤ 1000 dan Energi ≥ 30

Pada kondisi ini, bot akan menembakkan energi dengan persamaan matematis

$$\text{Firepower} = 5 - \frac{\text{jarak}}{200}$$

- c. Jarak ≤ 200 dan Energi < 30

Pada kondisi ini, bot akan cenderung bergerak ke arah musuh supaya dapat terjadi tabrakan dan menembak dengan energi konstan sebesar 1.

4. Penanganan event tabrakan

Ketika bot menabrak *wall* atau bot musuh, maka bot akan diperintahkan untuk bergerak dengan arah yang terbalik dari gerakan sebelumnya.

5. Kondisi turn selesai

Class ini bertujuan untuk memastikan bahwa semua perintah pada bot selesai dalam *turn* tersebut sebelum melanjutkan perintah pada *turn* berikutnya.

4.1.2 Implementasi Strategi Pada Bot DonatMerah

Berikut implementasi strategi Donat Merah di Robocode Tank Royale.

4.1.2.1 Pseudocode

Procedure Main()

Kamus Lokal:

(Tidak ada variabel tambahan, hanya memanggil konstruktor dan Start)

Algoritma:

new DonatMerah()
Start()

Procedure DonatMerah.Run()

Kamus Lokal:

red : color
centerX, centerY, minDim, radius : real
candidates : array of PointF (4 titik)
minDistance : real
minIndex : integer
dx, dy, d : real
angleToCenter, desiredHeading, headingError : real
currentDistance, distanceError, correction : real
initialized : boolean (field)

Algoritma:

red \leftarrow (255,0,0)
BodyColor \leftarrow red
TurretColor \leftarrow red
RadarColor \leftarrow red
ScanColor \leftarrow red
BulletColor \leftarrow red
AdjustGunForBodyTurn \leftarrow true
AdjustRadarForBodyTurn \leftarrow false
AdjustRadarForGunTurn \leftarrow false

if (initialized = false) then
 centerX \leftarrow ArenaWidth / 2
 centerY \leftarrow ArenaHeight / 2
 minDim \leftarrow min(ArenaWidth, ArenaHeight)
 radius \leftarrow 0.5 \times (minDim / 2)

candidates[0] \leftarrow (centerX + radius, centerY)
candidates[1] \leftarrow (centerX, centerY - radius)
candidates[2] \leftarrow (centerX - radius, centerY)
candidates[3] \leftarrow (centerX, centerY + radius)

minDistance \leftarrow sangat besar
minIndex \leftarrow 0

```

i ← 0
while (i < 4) do
    dx ← candidates[i].X - X
    dy ← candidates[i].Y - Y
    d ← √(dx2 + dy2)
    if (d < minDistance) then
        minDistance ← d
        minIndex ← i
    endif
    i ← i + 1
endwhile

MoveTo(candidates[minIndex].X, candidates[minIndex].Y)
initialized ← true
endif

while (IsRunning) do
    AvoidWallsInLoop()

    angleToCenter ← DirectionTo(centerX, centerY)
    desiredHeading ← NormalizeRelativeAngle(angleToCenter - 90)
    headingError ← NormalizeRelativeAngle(desiredHeading - Direction)
    SetTurnRight(headingError)

    currentDistance ← DistanceTo(centerX, centerY)
    distanceError ← currentDistance - radius
    correction ← 0
    if (|distanceError| > RADIUS_TOLERANCE) then
        correction ← -distanceError × RADIUS_CORRECTION_FACTOR
        correction ← clamp(correction, -50, 50)
    endif

    SetForward(correction)
    SetTurnRadarRight(RADAR_TURN_STEP)

    Go()
    AvoidWallsInLoop()
endwhile

```

Procedure OnScannedBot(e)

Kamus Lokal:

targetDistance : real
 firePower : real
 radarDirection, gunDirection : real
 smoothingFactor : real

Algoritma:

```
targetDistance ← √((e.X - X)2 + (e.Y - Y)2)
firePower ← jika (targetDistance < 2 × radius) maka 3 else 1

if (e.Energy < 20) then
    targetDistance ← √((e.X - X)2 + (e.Y - Y)2)
    firePower ← jika (targetDistance < 2 × radius) maka 3 else 1
    SetForward(targetDistance + 50)
    gunDirection ← GunBearingTo(e.X, e.Y)
    Fire(firePower)
    Go()
    RepositionToDonatMerah()
else
    radarDirection ← RadarBearingTo(e.X, e.Y)
    gunDirection ← GunBearingTo(e.X, e.Y)
    smoothingFactor ← 0.25
    adjustedRadarTurn ← smoothingFactor × radarDirection
    adjustedGunTurn ← smoothingFactor × gunDirection
    SetTurnRadarLeft(adjustedRadarTurn)
    SetTurnGunLeft(adjustedGunTurn)
    Fire(firePower)
    Go()
endif
```

Procedure MoveTo(targetX, targetY)

Kamus Lokal:

bearing : real
distance : real

Algoritma:

```
bearing ← BearingTo(targetX, targetY)
SetTurnRight(bearing)
Go()
distance ← DistanceTo(targetX, targetY)
SetForward(distance)
Go()
```

Procedure RepositionToDonatMerah()

Kamus Lokal:

candidates : array of PointF(4)
minDistance : real
minIndex : integer
dx, dy, d : real

Algoritma:

```
candidates[0] ← (centerX + radius, centerY)
candidates[1] ← (centerX, centerY - radius)
candidates[2] ← (centerX - radius, centerY)
candidates[3] ← (centerX, centerY + radius)
```

```
minDistance ← sangat besar
```

```
minIndex ← 0
```

```
i ← 0
```

```
while (i < 4) do
```

```
    dx ← candidates[i].X - X
```

```
    dy ← candidates[i].Y - Y
```

```
    d ←  $\sqrt{dx^2 + dy^2}$ 
```

```
    if (d < minDistance) then
```

```
        minDistance ← d
```

```
        minIndex ← i
```

```
    endif
```

```
    i ← i + 1
```

```
endwhile
```

```
MoveTo(candidates[minIndex].X, candidates[minIndex].Y)
```

Procedure OnHitBot(e)

Kamus Lokal:

```
(data event e)
```

Algoritma:

```
MoveTo(e.X, e.Y)
```

```
SetTurnLeft(BearingTo(e.X, e.Y) + 90)
```

```
SetBack(100)
```

Procedure OnHitWall(e)

Kamus Lokal:

```
(data event e)
```

Algoritma:

```
AvoidWallsInLoop()
```

Procedure AvoidWallsInLoop()

Kamus Lokal:

```
avoided : boolean
```

Algoritma:

```
avoided ← false
if (X < WALL_MARGIN) then
    SetStop()
    SetBack(100)
    SetTurnRight(90)
    avoided ← true
else if (X > ArenaWidth - WALL_MARGIN) then
    SetStop()
    SetBack(100)
    SetTurnLeft(90)
    avoided ← true
else if (Y < WALL_MARGIN) then
    SetStop()
    SetBack(100)
    SetTurnRight(90)
    avoided ← true
else if (Y > ArenaHeight - WALL_MARGIN) then
    SetStop()
    SetBack(100)
    SetTurnLeft(90)
    avoided ← true
endif

if (avoided) then
    Go()
endif
```

4.1.2.2 Struktur Data

Pada bot DonatMerah, digunakan struktur data sebagai berikut.

- a. double centerX, berfungsi untuk menyimpan absis pusat arena (titik tengah arena).
- b. double centerY, berfungsi untuk menyimpan ordinat pusat arena (titik tengah arena).
- c. double minDim, berfungsi untuk menyimpan nilai terkecil antara lebar dan tinggi arena, digunakan untuk menghitung radius.
- d. double radius, berfungsi untuk menyimpan nilai jari-jari yang akan digunakan untuk membuat lintasan melingkar di sekitar pusat arena.
- e. bool initialized, berfungsi untuk menyimpan status apakah inisialisasi awal (pemilihan titik tepi lingkaran awal) sudah dilakukan atau belum.
- f. double WALL_MARGIN, berfungsi untuk menyimpan jarak aman minimal dari dinding agar bot tidak terlalu dekat dengan tepi arena.

- g. double RADIUS_CORRECTION_FACTOR, berfungsi untuk menyimpan koefisien pengoreksi posisi bot agar selalu menjaga jarak sejauh radius dari pusat.
- h. double RADIUS_TOLERANCE, berfungsi untuk menyimpan nilai toleransi error jarak ke pusat sebelum bot melakukan koreksi.
- i. double RADAR_TURN_STEP, berfungsi untuk menyimpan nilai besar putaran radar setiap giliran.

4.1.2.3 Fungsi

DonatMerah() (Konstruktor), memanggil konstruktor basis untuk melakukan inisialisasi bot dengan informasi dari file konfigurasi (.json).

4.1.2.4 Prosedur

Prosedur diantaranya :

a. Run()

Fungsi utama yang dieksekusi saat bot dijalankan. Pada awalnya, bot menghitung pusat arena, menentukan radius lintasan, dan menghitung empat kandidat titik pada tepi lingkaran (0° , 90° , 180° , 270°). Bot kemudian memilih titik terdekat dari posisi awal dan dipindahkan ke titik tersebut (melalui prosedur MoveTo()). Setelah itu, bot masuk ke dalam loop utama. Memanggil prosedur untuk menghindari dinding (AvoidWallsInLoop()). Menghitung sudut ke pusat arena dan menentukan heading agar bot bergerak melingkar (dengan logika “desiredHeading” yang dihasilkan dari angleToCenter- 90°). Mengoreksi jarak agar tetap berada pada radius yang diinginkan. Mengirim perintah pergerakan (SetForward) dan perintah radar (SetTurnRadarRight). Menjalankan perintah tersebut dengan Go(). Kemudian, kembali memanggil AvoidWallsInLoop() untuk memastikan bot tidak tersangkut dinding.

b. OnScannedBot(event e)

Saat bot mendekripsi musuh, menghitung jarak target dan menentukan firepower berdasarkan jarak apakah kurang dari 2 kali radius. Jika energi bot lawan di bawah suatu batas tertentu, bot mendekati target (MoveTo) lalu menembak dan melakukan reposisi dengan RepositionToDonatMerah(). Jika tidak, bot melakukan penyesuaian perputaran radar dan gun dengan faktor smoothing, lalu menembak dengan firepower yang telah ditentukan.

c. OnHitBot(event e)

Saat terjadi tabrakan dengan bot lain, bot mendekati posisi tabrakan, melakukan putaran (dengan menambahkan 90° ke sudut hadapnya) dan mundur.

d. OnHitWall(event e)

Saat bot menabrak dinding, bot memanggil prosedur AvoidWallsInLoop() untuk menghindari atau memperbaiki posisi sehingga tidak tersangkut dinding.

e. MoveTo(targetX, targetY)

Menghitung bearing (sudut) dari posisi bot ke titik target. Mengatur perintah putar (SetTurnRight) ke bearing tersebut, lalu memanggil Go() untuk mengirim perintah. Menghitung jarak ke titik target dan mengatur perintah maju (SetForward()) dengan jarak tersebut, kemudian memanggil Go() lagi. Prosedur ini bertujuan untuk memindahkan bot ke titik tujuan.

f. RepositionToDoNotMerah()

Menghitung ulang empat kandidat titik pada tepi lingkaran. Menghitung jarak dari posisi bot saat ini ke masing-masing empat titik kandidat tersebut. Menentukan kandidat terdekat berdasarkan jarak terkecil. Memanggil MoveTo() untuk menggerakkan bot ke titik kandidat terdekat tersebut. Prosedur ini bertujuan untuk mengembalikan bot ke posisi untuk bergerak melingkar.

g. AvoidWallsInLoop()

Mengecek apakah koordinat X atau Y bot berada di dalam margin dinding. Jika ya, menghentikan pergerakan dengan SetStop(), kemudian mengatur perintah mundur SetBack dan putar SetTurnRight atau SetTurnLeft sebesar 90° untuk menjauh dari dinding. Memanggil Go() untuk mengeksekusi perintah jika ada perintah penghindaran dinding. Prosedur ini memastikan bot tidak akan tersangkut ke dinding.

4.1.3 Implementasi Strategi Pada Bot Avenger

Berikut implementasi strategi Avenger di Robocode Tank Royale.

4.1.3.1 Pseudocode

Procedure Main()

Kamus Lokal:

(Tidak ada variabel tambahan, hanya memanggil konstruktur dan Start)

Algoritma:

new DonatMerah()

Start()

Procedure Run()

Kamus Lokal:

```

lastScannedTick : integer
targetId : integer
targetLocation : tuple of reals (X, Y)
rageFactor : real
BotMode : enumerated string (antara Search atau Revenge)
mode : string
AdjustGunForBodyTurn, AdjustRadarForBodyTurn, AdjustRadarForGunTurn : boolean
TurnNumber : integer (turn sekarang)

```

Algoritma:

```
set bodyColor, turretColor, radarColor, bulletColor, scanColor sesuai keinginan
```

```

AdjustGunForBodyTurn ← true
AdjustRadarForBodyTurn ← true
AdjustRadarForGunTurn ← true

```

```

targetId ← -1
lastScannedTick ← 0

```

```

while (bot is running) do
    // saat sedang berlari
    if (targetId < 0 or TurnNumber - lastScannedTick > 1) then
        mode ← BotMode.Search
        SetTurnRadarLeft(infinity)
    endif
    // saat sudah memiliki target tetapi belum ditemukan
    if (targetId > 0 and mode == BotMode.Search) then
        lastScannedTick ← TurnNumber
        SetTurnRadarLeft(infinity)
        SetTurnLeft(BearingTo(targetLocation.X, targetLocation.Y))
        SetForward(100)
    endif
    Go() // eksekusi perintah yang tertunda

```

Procedure OnScannedBot(e)

Kamus Lokal:

```

lastScannedTick : integer
targetId : integer
targetLocation : tuple of reals (X, Y)
rageFactor : real
BotMode : enumerated string (antara Search atau Revenge)
mode : string
AdjustGunForBodyTurn, AdjustRadarForBodyTurn, AdjustRadarForGunTurn : boolean
TurnNumber : integer (turn sekarang)
targetDistance : real
firePower : real
targetPosition : tuple of reals (X, Y)
radarDirection, gunDirection, moveDirection : reals

```

e : ScannedBotEvent

Algoritma:

```
if (targetId < 0)
    SetTurnLeft(-GunBearingTo(e.X, e.Y));
    SetForward(100)
endif
if (targetId = e.ScannedBotId)
    mode ← BotMode.Revenge
    lastScannedTick ← TurnNumber
    targetLocation ← (e.X, e.Y)

    targetDistance ← Math.Sqrt(Math.Pow(e.X - X, 2) + Math.Pow(e.Y - Y, 2))
    firePower ← 4 * Math.Exp(-targetDistance / (250 + rageFactor))
    targetPosition ← LinearPrediction(e, targetDistance / (20 - (3 * firePower)))

    radarDirection ← RadarBearingTo(e.X, e.Y)
    gunDirection ← GunBearingTo(targetPosition.X, targetPosition.Y)
    moveDirection ← BearingTo(targetPosition.X, targetPosition.Y)

    SetTurnRadarLeft(radarDirection)
    SetTurnGunLeft(gunDirection)
    Fire(firePower);

    if (targetDistance > 50) then
        SetTurnLeft(moveDirection)
        SetForward(targetDistance / 2)
    endif

    if (radarDirection = 0) then
        Rescan()      // lakukan scan kembali
    endif
```

Procedure OnHitByBullet(e)

Kamus Lokal:

targetId : integer
e : HitByBulletEvent

Algoritma:

```
if (targetId < 0) then
    SwitchTargets(e.Bullet.OwnerId, e.Bullet.X, e.Bullet.Y)
endif
```

Procedure OnBotDeath(e)

Kamus Lokal:

e : BotDeathEvent

```
targetId : integer  
rageFactor : real  
mode : string  
BotMode : enumerated string (antara Search atau Revenge)
```

Algoritma:

```
if (e.VictimId = targetId) then  
    targetId ← -1  
    rageFactor ← 0  
    mode ← BotMode.Search  
endif
```

Procedure OnHitBot(e)

Kamus Lokal:

```
e : HitBotEvent  
targetId : integer
```

Algoritma:

```
if (targetId < 0) then  
    SwitchTargets(e.VictimId, e.X, e.Y)  
endif  
SetTurnLeft(BearingTo(e.X, e.Y) + 90)  
SetBack(100)
```

Procedure OnBulletHit(e)

Kamus Lokal:

```
e : BulletHitBotEvent  
rageFactor : real
```

Algoritma:

```
if (e.VictimId = targetId) then  
    rageFactor ← rageFactor + 100  
else then  
    rageFactor ← 0  
endif
```

Procedure SwitchTargets(id, X, Y)

Kamus Lokal:

```
lastScannedTick : integer  
targetId : integer  
targetLocation : tuple of reals (X, Y)  
id : integer  
X, Y : real
```

Algoritma:

```

lastScannedTick ← TurnNumber
targetId ← id
targetLocation ← (X, Y)
SetTurnRadarLeft(RadarBearingTo(X, Y))
SetTurnGunLeft(GunBearingTo(X, Y))
SetForward(100)

```

Function LinearPrediction(scannedBot, time)

Kamus Lokal:

theta : real
 predictedX, predictedY : reals
 scannedBot : ScannedBotEvent
 time : real

Algoritma:

```

theta ← pi * scannedBot.Direction / 180.0

predictedX ← scannedBot.X + cos(theta) * scannedBot.Speed * time
predictedY ← scannedBot.Y + sin(theta) * scannedBot.Speed * time

return (predictedX, predictedY)

```

4.1.3.2 Struktur Data

Pada bot Avenger, digunakan struktur data sebagai berikut.

- a. int lastScannedTick. Berfungsi untuk menyimpan turn terakhir target (musuh yang memicu amarah Avenger) terdeteksi, jadi dapat digunakan untuk mengetahui kapan target hilang.
- b. int targetId. Berfungsi untuk menyimpan bot yang dipilih menjadi target akibat mekanisme “balas dendam”.
- c. tuple double targetLocation. Tuple berisi lokasi bot musuh yang memicunya, sehingga bot bisa berjalan ke arah tersebut untuk mencari di mana lokasi pemicu. Ini bukan *backtracking*, karena nilai tersebut hanya disimpan sekali saat terdeteksi Avenger ditembak atau ditabrak.
- d. int rageFactor. Berfungsi untuk menyimpan nilai rage factor yang telah dijelaskan pada strategi *greedy* Avenger. Nilai ini ditingkatkan tiap kali Avenger berhasil menembak targetnya.
- e. enum BotMode. Menduduki nilai antara “Search” atau “Revenge”, yang berfungsi untuk *keep track* apakah Avenger sudah menemukan bot targetnya, sehingga diketahui

kapan untuk terus meng-scan untuk target, atau mengejar musuh yang sudah ditemukan.

4.1.3.3 Fungsi

Berikut fungsi utama dan konstruktor Avenger.

- a. `Avenger()` (Konstruktor)

Memanggil konstruktor basis untuk melakukan inisialisasi bot dengan informasi dari file konfigurasi (.json).

Berikut fungsi lainnya.

- b. `LinearPrediction(ScannedBotEvent scannedBot, double time)`

Fungsi ini yang mengimplementasikan prediksi linier sangat sederhana (berbeda dengan Asteroid Destroyer), yang hanya memprediksi berdasarkan kecepatan dan lokasi target pada setelah `time` turns telah berjalan.

4.1.3.4 Prosedur

Berikut beberapa prosedur yang meng-*handle* respons terhadap apa yang dialami bot Avenger.

- a. `static void Main()`

Memulai program dan membuat instance bot serta memulai eksekusi perintah dengan memanggil `Start()`.

- b. `Run()`

Ini prosedur utama bot, yang pertama dijalankan di awal setiap ronde. Beberapa settingan untuk bot seperti warna, penggunaan radar dan gun, juga atribut bot berada di sini. Adapun loop `while` yang terus berjalan selama bot masih hidup. Untuk bot Avenger, ini digunakan untuk mengecek apakah bot sedang harus *search* atau *run* pada suatu turn.

- c. `OnScannedBot(ScannedBotEvent e)`

Ini prosedur yang terpanggil saat bot mendeteksi musuh. Di sini terletak logika untuk pemilihan target berdasarkan “balas dendam”, pengejaran target, dan perhitungan *fire power*.

- d. `OnHitByBullet(HitByBulletEvent e)`

Ini prosedur yang terpanggil saat bot tertembak. Maka, jika bot belum ada musuh, “balas dendam”-nya akan terpicu.

- e. `OnBotDeath(BotDeathEvent e)`

Ini prosedur yang terpanggil saat suatu bot mati. Jika bot tersebut adalah bot target, Avenger kembali ke mode pelarian diri.

f. `OnHitBot(HitBotEvent e)`

Ini prosedur yang terpanggil saat bot tertabrak. Maka, jika bot belum ada musuh, “balas dendam”-nya akan terpicu.

g. `OnBulletHit(BulletHitBotEvent e)`

Ini prosedur yang terpanggil saat Avenger berhasil menembak musuh. Jika berhasil menembak target, rage factor akan dinaikkan.

h. `SwitchTargets(int id, double X, double Y)`

Ini prosedur untuk skema “balas dendam”, yaitu untuk memilih suatu target yang memicu Avenger.

4.1.4 Implementasi Strategi Pada Asteroid Destroyer

Berikut implementasi strategi Asteroid Destroyer di Robocode Tank Royale.

4.1.4.1 Pseudocode

Procedure Main()

Kamus Lokal:

(Tidak ada variabel tambahan, hanya memanggil konstruktor dan Start)

Algoritma:

```
new AsteroidDestroyer()  
Start()
```

Procedure Run()

Kamus Lokal:

ScannedTurn: integer
TargetId: integer
TargetEnergy : real
Polarity : integer (1 atau -1)
BotMode : enumerated string (antara Scanning, Chasing, Strafing)
Mode : string
ModeCooldown : integer
AdjustGunForBodyTurn, AdjustRadarForBodyTurn, AdjustRadarForGunTurn : boolean
TurnNumber : integer (turn sekarang)
ArenaWidth, ArenaHeight : integers

Algoritma:

set bodyColor, turretColor, radarColor, bulletColor, scanColor sesuai keinginan

```

AdjustGunForBodyTurn ← true
AdjustRadarForBodyTurn ← true
AdjustRadarForGunTurn ← true

ScannedTurn ← 0

while (IsRunning) do
    if (TurnNumber - ScannedTurn > 1) then
        Mode ← BotMode.Scanning
    endif

    if (Mode = BotMode.Scanning) then
        TargetId ← -1
        SetTurnRadarLeft(infinity)
        SetTurnLeft(BearingTo(ArenaWidth/2, ArenaHeight/2))
        SetForward(100)
    endif

    Go()      // eksekusi perintah yang tertunda
endwhile

```

Procedure OnScannedBot(e)

Kamus Lokal:

ScannedTurn: integer
 TargetId: integer
 TargetEnergy : real
 Polarity : integer (1 atau -1)
 BotMode : enumerated string (antara Scanning, Chasing, Strafing)
 Mode : string
 ModeCooldown : integer
 AdjustGunForBodyTurn, AdjustRadarForBodyTurn, AdjustRadarForGunTurn : boolean
 TurnNumber : integer (turn sekarang)
 ArenaWidth, ArenaHeight : integers
 TargetDistance : real
 DistanceFactor, SpeedFactor, EnergyFactor : real
 FirePower : real
 TargetPosition : tuple of reals (X, Y)
 RadarDirection, GunDirection, MoveDirection : reals
 e : ScannedBotEvent

Algoritma:

```

if (TargetId < 0) then
    TargetId ← e.ScannedBotId
    TargetEnergy ← e.Energy
endif

if (TargetId = e.ScannedBotId) then
    ScannedTurn = TurnNumber

```

```

TargetEnergy ← e.Energy
TargetDistance ← Math.Sqrt(Math.Pow(e.X - X, 2) + Math.Pow(e.Y - Y, 2))

DistanceFactor ← 3 * Math.Exp(-TargetDistance / 250)
SpeedFactor ← 1 - (e.Speed / 8)
EnergyFactor ← 1 - Math.Exp(-Energy / 25)
FirePower ← (DistanceFactor + SpeedFactor) * EnergyFactor

TargetPosition ← LinearPrediction(e, FirePower)

RadarDirection ← RadarBearingTo(e.X, e.Y)
GunDirection ← GunBearingTo(TargetPosition.X, TargetPosition.Y)
MoveDrecion ← BearingTo(TargetPosition.X, TargetPosition.Y)

SetTurnRadarLeft(RadarDirection)
SetTurnGunLeft(GunDirection)
Fire(FirePower)

if (ModeCooldown > 15) then
    ModeCooldown ← 0
    if (TargetDistance > 150 || e.Energy < 25) then
        Mode ← BotMode.Chasing
    else then
        Mode ← BotMode.Strafing
    endif
else then
    ModeCooldown ← ModeCooldown + 1
endif

if (Mode = BotMode.Chasing) then
    SetTurnLeft(MoveDrecion)
    SetForward(TargetDistance / 2)
else then
    SetTurnLeft(MoveDrecion + 90)
    SetForward(Polarity * 100)
endif

if (RadarDirection = 0) then
    Rescan()

else if (e.Energy < TargetEnergy) then
    TargetId ← e.ScannedBotId
    TargetEnergy ← e.Energy
endif

```

Procedure OnBotDeath(e)

Kamus Lokal:

e : BotDeathEvent
TargetId : integer

Algoritma:

```
if (e.VictimId = TargetId) then
    TargetId ← -1
endif
```

Procedure OnHitBot(e)

Kamus Lokal:

e : HitBotEvent
TargetId : integer
TargetEnergy : real
Polarity : integer (1 or -1)

Algoritma:

```
Polarity ← - Polarity
SetBack(100)
SetTurnRadarLeft(RadarBearingTo(e.X, e.Y) + 90)
if (e.Energy < TargetEnergy) then
    TargetId ← e.VictimId
    TargetEnergy ← e.Energy
    SetTurnRadarLeft(RadarBearingTo(e.X, e.Y))
endif
Go()
```

Procedure OnHitWall(e)

Kamus Lokal:

e : HitWallEvent

Algoritma:

```
Polarity ← - Polarity
```

Function LinearPrediction(scannedBot, FirePower)

Kamus Lokal:

bulletSpeed : real
predictedX, predictedY : reals
enemyDirection : real
prevDistance : real
distance : real
time : real
scannedBot : ScannedBotEvent
FirePower : real

Algoritma:

```

bulletSpeed ← 20 - (3 * firePower)
predictedX ← scannedBot.X
predictedY ← scannedBot.Y
enemyDirection ← pi * scannedBot.Direction / 180.0

prevDistance ← infinity

for (i from 1 to 30) do
{
    distance = sqrt((predictedX - X)^2 +(predictedY - Y)^2)
    time = distance / bulletSpeed

    if (|distance - prevDistance| < 1.0) then
        break
    endif

    prevDistance = distance

    predictedX ← scannedBot.X + cos(enemyDirection) * scannedBot.Speed * time
    predictedY ← scannedBot.Y + sin(enemyDirection) * scannedBot.Speed * time
}

return (predictedX, predictedY)

```

4.1.4.2 Struktur Data

Pada bot Avenger, digunakan struktur data sebagai berikut.

- a. int ScannedTurn. Berfungsi untuk menyimpan turn terakhir target terdeteksi, jadi dapat digunakan untuk mengetahui kapan target hilang.
- b. int TargetId. Digunakan untuk menyimpan bot yang menjadi target Asteroid Destroyer. Ini berganti saat Asteroid Destroyer menemukan target lain, sedangkan nilai -1 menandakan sedang tidak ada target.
- c. double TargetEnergy. Berisi nilai energi bot target sekarang, sehingga bisa dibandingkan dengan bot yang di-scan untuk menentukan apakah perlu ganti target.
- d. enum BotMode. Bernilai antara “Scanning”, “Chasing”, dan “Strafing” untuk mengetahui gerakan yang perlu dilakukan.
- e. BotMode Mode. Botmode yang sedang diduduki bot sekarang.
- f. int ModeCooldown. Suatu *cooldown*, yaitu interval waktu, untuk transisi antara mode. Ini digunakan agar transisi antara mode tidak terlalu mendadak, dan juga untuk menyelesaikan bug di mana bot stuck di tempat.

4.1.4.3 Fungsi

Berikut fungsi utama dan konstruktor Asteroid Destroyer.

- a. `AsteroidDestroyer()` (Konstruktor)

Memanggil konstruktor basis untuk melakukan inisialisasi bot dengan informasi dari file konfigurasi (.json).

Berikut fungsi lainnya.

- b. `LinearPrediction(ScannedBotEvent scannedBot, double time)`

Fungsi ini yang mengimplementasikan prediksi linier yang lebih advanced daripada Avenger. Prediksi ini berdasarkan rumus rekuren yang telah dijelaskan sebelumnya, sehingga terlihat bahwa digunakan `for` loop untuk mendapatkan prediksi yang baik.

4.1.4.4 Prosedur

Berikut beberapa prosedur yang meng-*handle* respons terhadap apa yang dialami bot Avenger.

- a. `Main()`

Memulai program dan membuat instance bot serta memulai eksekusi perintah dengan memanggil `Start()`.

- b. `Run()`

Ini prosedur utama bot, yang pertama dijalankan di awal setiap ronde. Beberapa settingan untuk bot seperti warna, penggunaan radar dan gun, juga atribut bot berada di sini. Adapun loop `while` yang terus berjalan selama bot masih hidup. Untuk bot Asteroid Destroyer, ini digunakan untuk mengecek apakah bot sedang harus *searching* suatu target.

- c. `OnScannedBot(ScannedBotEvent e)`

Ini prosedur yang terpanggil saat bot mendeteksi musuh. Di sini terletak logika untuk pemilihan target berdasarkan energi yang lebih kecil. Juga ada perhitungan *fire power* serta penembakan yang akan dilakukan. Gerakan bot terhadap target juga dikendalikan di sini berdasarkan jarak dengan target.

- d. `OnBotDeath(BotDeathEvent e)`

Ini prosedur yang terpanggil saat suatu bot mati. Jika bot tersebut adalah bot target, Asteroid Destroyer kembali mencari target lain.

- e. `OnHitBot(HitBotEvent e)`

Ini prosedur yang terpanggil saat bot tertabrak dengan bot lain. Bot akan putar balik saat menabrak tembok, tetapi juga mengecek apakah akan menggantikan target ke bot yang ditabraknya jika bot tersebut memiliki energi lebih kecil.

f. OnHitWall(HitWallEvent e)

Ini prosedur yang terpanggil saat bot tertabrak tembok. Bot akan putar balik saat menabrak tembok.

4.2 Pengujian Setiap Strategi

Berikut, diujikan setiap strategi bot yang telah dirancang dalam Robocode Tank Royale.

4.2.1 Pengujian setiap Bot (1 vs 1 vs 1 vs 1)

Berikut merupakan video demonstrasi setiap bot yang ditandingkan 1 vs 1 vs 1 vs 1.

👉 Demonstrasi 1 vs 1 vs 1 vs 1.mp4

| Results for 5 rounds | | | | | | | | | | | | |
|----------------------|------------------------|-------------|----------|-------------|-------------|--------------|----------|-----------|------|------|------|--|
| Rank | Name | Total Score | Survival | Surv. Bonus | Bullet Dmg. | Bullet Bonus | Ram Dmg. | Ram Bonus | 1sts | 2nds | 3rds | |
| 1 | Asteroid Destroyer 1.1 | 1731 | 600 | 120 | 805 | 138 | 40 | 28 | 4 | 0 | 0 | |
| 2 | ngegasTron 1.0 | 708 | 450 | 30 | 192 | 9 | 26 | 0 | 0 | 3 | 1 | |
| 3 | Avenger 1.0 | 704 | 250 | 0 | 399 | 35 | 19 | 0 | 1 | 1 | 2 | |
| 4 | Donat Merah 1.0 | 472 | 200 | 0 | 264 | 0 | 8 | 0 | 0 | 1 | 2 | |

Terlihat bahwa bot Asteroid Destroyer sebagai bot utama dapat memperoleh kemenangan pada akhir permainan, yang merupakan poin kumulatif dari setiap ronde yang dijalankan, dan mampu memperoleh posisi pertama dalam 4 ronde dari total 5 ronde yang dijalankan. Hal ini menunjukkan bahwa bot dapat memilih solusi optimum lokal yang mampu membawa pada solusi yang menghampiri solusi optimum global, karena bot mampu memenangkan hampir setiap ronde yang diadakan dan memperoleh posisi pertama dalam akumulasi poin.

Ketika melawan bot alternatif yang dirancang dengan strategi greedy, bot Asteroid Destroyer mampu memperoleh solusi yang optimal global atau menghampirinya. Hal ini ditunjukkan karena bot mampu memilih musuh yang menjadi target sesuai dengan strategi greedy yang diterapkan, yaitu selalu memperbarui target ketika ditemukan musuh dengan Energi lebih rendah. Selanjutnya, bot juga mampu mengejar target dengan optimal dan bergerak melingkar saat jarak sudah terlalu dekat, hal ini mampu memastikan target dapat terserang dengan optimal dan serangan target dapat dihindari. Lalu, Firepower yang ditembakkan oleh bot juga sudah menunjukkan solusi yang optimum lokal dan menghampiri optimum global, hal ini

dikarenakan firepower yang ditembakkan sudah sesuai dengan berbagai parameter yang ditentukan sehingga dapat mengenai target serta tidak mengurangi energi bot secara cepat, dan peluru yang ditembakkan memiliki akurasi yang tinggi. Terakhir, prediksi linear yang diterapkan pada bot mampu memilih solusi optimum lokal yang menghampiri optimum global, karena peluru yang ditembakkan dengan prediksi yang diberikan mampu mengenai target dengan akurasi yang tinggi.

4.2.2 ngegasTron

Berikut, merupakan video demonstrasi dari bot ngegasTron melawan bot-bot yang disediakan pada template.

▶ Demonstrasi ngegasTron.mp4

| Results for 10 rounds | | | | | | | | | | | | |
|-----------------------|----------------|-------------|----------|-------------|-------------|--------------|----------|-----------|------|------|------|--|
| Rank | Name | Total Score | Survival | Surv. Bonus | Bullet Dmg. | Bullet Bonus | Ram Dmg. | Ram Bonus | 1sts | 2nds | 3rds | |
| 1 | ngegasTron 1.0 | 1722 | 800 | 120 | 615 | 98 | 89 | 0 | 5 | 1 | 0 | |
| 2 | Ram Fire 1.0 | 1268 | 300 | 30 | 492 | 0 | 323 | 123 | 1 | 2 | 2 | |
| 3 | Track Fire 1.0 | 839 | 200 | 0 | 586 | 53 | 0 | 0 | 0 | 3 | 0 | |
| 4 | Corners 1.0 | 626 | 450 | 0 | 168 | 6 | 1 | 0 | 0 | 0 | 4 | |

Terlihat bahwa bot ngegasTron dapat memperoleh kemenangan pada akhir permainan, yang merupakan poin kumulatif dari setiap ronde yang dijalankan. Melalui demonstrasi yang dijalankan, solusi yang dipilih oleh bot belum mencapai solusi optimal global. Penggunaan strategi greedy untuk bullet power belum mencapai solusi optimum global. Hal ini ditunjukkan dari bullet yang seringkali ditembakkan tapi tidak mengenai musuh. Peluru yang meleset terjadi karena posisi *gun* saat menembak tidak tepat menuju musuh atau musuh yang bergerak ke arah lain ketika peluru ditembakkan.

Selanjutnya, penggunaan strategi greedy untuk survivability juga belum memperoleh solusi optimum global. Hal ini ditunjukkan dari bot yang cepat kehilangan energi dan tidak banyak memperoleh energi meskipun sudah memilih pilihan berdasarkan strategi greedy yang diterapkan.

4.2.3 Donat Merah

Berikut, merupakan video demonstrasi dari bot Donat Merah melawan bot-bot yang disediakan pada template.

▶ Simulasi DonatMerah.mp4

Results for 10 rounds

| Rank | Name | Total Score | Survival | Surv. Bonus | Bullet Dmg. | Bullet Bo... | Ram Dmg. | Ram Bonus | 1sts | 2nds | 3rds |
|------|-----------------|-------------|----------|-------------|-------------|--------------|----------|-----------|------|------|------|
| 1 | Donat Merah 1.0 | 989 | 350 | 60 | 480 | 47 | 52 | 0 | 2 | 1 | 1 |
| 2 | Crazy 1.0 | 465 | 300 | 0 | 152 | 1 | 11 | 0 | 0 | 3 | 0 |
| 3 | Corners 1.0 | 386 | 200 | 30 | 150 | 3 | 2 | 0 | 1 | 0 | 2 |
| 4 | Fire 1.0 | 160 | 50 | 0 | 108 | 1 | 0 | 0 | 1 | 0 | 1 |

Terlihat bahwa bot Donat Merah dapat memperoleh kemenangan pada akhir permainan, yang merupakan poin kumulatif dari setiap ronde yang dijalankan.

Melalui demonstrasi yang dijalankan, solusi yang dipilih oleh bot belum mencapai solusi optimal global. Hal tersebut terjadi karena proses greedy penembakan dan penabrakan masih belum presisi. Selain itu, faktor koreksi untuk lintasan lingkaran pun masih belum sempurna terutama saat bot menabrak dinding bagian bawah. Namun, terlihat dari hasil permainan melawan bot sample masih cukup baik performanya.

4.2.4 Avenger

Berikut, merupakan video demonstrasi dari bot Avenger melawan bot-bot yang disediakan pada template.

🎥 Demonstrasi Avenger.mkv

Results for 5 rounds

| Rank | Name | Total Score | Survival | Surv. Bonus | Bullet Dmg. | Bullet Bonus | Ram Dmg. | Ram Bonus | 1sts | 2nds | 3rds |
|------|----------------|-------------|----------|-------------|-------------|--------------|----------|-----------|------|------|------|
| 1 | Avenger 1.0 | 1812 | 700 | 120 | 768 | 127 | 59 | 38 | 5 | 0 | 0 |
| 2 | Corners 1.0 | 833 | 500 | 30 | 296 | 7 | 0 | 0 | 0 | 2 | 3 |
| 3 | Track Fire 1.0 | 776 | 200 | 0 | 576 | 0 | 0 | 0 | 0 | 1 | 1 |
| 4 | Ram Fire 1.0 | 776 | 100 | 0 | 372 | 18 | 258 | 27 | 0 | 2 | 1 |

Bot Avenger, secara umum, memenangkan semua permainannya, walaupun tidak setiap ronde. Berdasarkan hasil poin kumulatif yang didapatkan, terlihat bahwa skor ram damage dan bullet damage lumayan tinggi akibat strategi “balas dendam” yang mengimplementasikan greedy pada seleksi dan penembakan musuh. Optimasi lokal berupa fokus menarget seorang musuh pemicu menjaminkan bullet damage. Tetapi, sepertinya strategi menjauhi diri dari musuh tidak seefektif yang dikira (terutama di *late-game*, ini hanya memperpanjang ronde). Mungkin mekanisme greedy untuk melaikkan diri dari bot sekitar dapat diperbaiki untuk menjamin optimum lokal dari survival skor tercapai. Terlihat juga bahwa strategi yang terlalu fokus pada musuh pemicu membuat bot mudah terjebak saat dihalangi bot lain, sehingga dia mudah mati secara tidak sengaja. Ini berarti strategi greedy tersebut paling baik digunakan saat arena lebih *sparse*.

4.2.5 Asteroid Destroyer

Berikut, merupakan video demonstrasi dari bot Asteroid Destroyer melawan bot-bot yang disediakan pada template.

▶ Demo AsteroidDestroyer.mp4

| 1 | Asteroid Destroyer 1.1 | 4534 | 1550 | 280 | 2115 | 400 | 169 | 20 | 8 | 0 | 0 | 0 |
|---|------------------------|------|------|-----|------|-----|-----|----|---|---|---|---|
| 2 | Crazy 1.0 | 1279 | 1000 | 0 | 244 | 0 | 35 | 0 | 0 | 2 | 4 | |
| 3 | Corners 1.0 | 978 | 650 | 0 | 316 | 6 | 6 | 0 | 0 | 2 | 2 | |
| 4 | Fire 1.0 | 889 | 450 | 0 | 424 | 8 | 7 | 0 | 0 | 2 | 0 | |
| 5 | Ram Fire 1.0 | 874 | 300 | 0 | 334 | 22 | 198 | 19 | 0 | 2 | 2 | |

Terlihat bahwa Asteroid Destroyer mencapai skor yang jauh lebih tinggi daripada bot sample. Ini menunjukkan bot Asteroid Destroyer sangat mendominasi. Perhatikan bahwa skor dari bullet damage, survival points, dan ram damage semua tinggi. Ini menunjukkan bahwa strategi greedy yang diterapkan sangat merata dan mempergunakan semua cara mendapatkan poin dengan baik. Bahkan, Asteroid Destroyer sangat produktif menghasilkan poin sehingga berlangsung 8 ronde sebelum game berakhir, lebih banyak daripada bot alternatif sebelumnya.

Terlihat bahwa sifat greedy yang agresif dari Asteroid Destroyer menghasilkan bullet damage dan bonusnya dengan efektif. Bahkan ini menjadi sumber poin terbesar. Hasil tersebut masuk akal karena banyak heuristik khusus Asteroid Destroyer bertujuan untuk memaksimalkan bullet damage sebagai bullet damage, terutama sistem pemilihan target, perhitungan fire power, dan prediksi linier. Adapun bahwa survival skor sangat tinggi walaupun Asteroid Destroyer tidak memiliki strategi pasif. Ini seperti pernyataan: “Offense is the greatest defense”. Tapi, tentu heuristik seperti pemindahan target, perhitungan fire power berdasarkan energi pribadi, dan strategi *strafing* membantu. Terakhir, terlihat bahwa ram damage lumayan besar, dan bahkan hampir menyaingi strategi Ramfire. Ini akibat strategi greedy untuk menabrak musuh dengan energi rendah, yang menunjukkan bahwa suatu pilihan kecil dan berdampak besar dalam algoritma greedy.

Strategi greedy yang diterapkan Asteroid Destroyer lumayan fleksibel, dan masih efektif dalam banyak kondisi (setidaknya saat ditandingkan dengan bot alternatif dan bot sampel). Tapi, Asteroid Destroyer terkadang sedikit kesulitan saat ada bot yang menghalangi jalurnya ke targetnya. Ini sedikit diselesaikan dengan strategi perpindahan target, karena bot yang menghalangi akan tidak sengaja kehilangan energi saat ditabrak dan ditembak. Namun, solusi

ini tidak selalu efektif. Tetapi, secara umum, pilihan greedy yang diambil Asteroid Destroyer secara lokal pada setiap turn, menghasilkan poin maksimal pada akhir ronde.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Melalui perancangan dan implementasi strategi pada pembuatan bot Robocode Tank Royale, dapat disimpulkan bahwa algoritma greedy dapat dengan efektif mencapai kondisi menang dalam setiap ronde permainan. Penerapan strategi greedy dalam bot dapat meningkatkan survivability dan damage score pada bot. Terlihat melalui berbagai demonstrasi yang dilakukan, bot dengan strategi greedy mampu menjadi pemenang dalam berbagai ronde melawan berbagai bot dengan strategi algoritma yang berbeda-beda. Hal ini menunjukkan bahwa solusi optimum lokal pada permainan tank royale dapat dengan cukup efisien mengantarkan bot untuk mencapai solusi optimum global atau mendekatinya.

Selanjutnya, dapat disimpulkan bahwa diterapkan suatu struktur *meta-greedy* yang memaksimalkan poin total permainan. Fungsi seleksinya sendiri dipecah menjadi sub persoalan mengenai gerakan, prediksi, fire damage, target, dan lainnya. Setiap sub persoalan tersebut diselesaikan dengan heuristik *meso-greedy* sendiri. Setelah eksperimentasi heuristik, diperoleh strategi greedy yang mengoptimalkan semua aspek perhitungan skor secara menyeluruh, walaupun secara lokal, adalah yang paling efektif untuk memenangkan permainan. Strategi agresif yang mengejar dari bot Avenger, strategi perhitungan fire power yang greedy dari bot ngegasTron, dan gerakan melingkar dari Donat Merah, bergabung untuk menghasilkan strategi holistik yang efektif seperti dilakukan pada Asteroid Destroyer.

5.2 Saran

Untuk bot ngegasTron, peningkatan dalam bot yang dapat dilakukan adalah dengan penambahan elemen prediksi gerakan musuh dan penyesuaian yang lebih dinamis terhadap kondisi pertempuran. Dengan melakukan optimasi waktu reaksi, kelola energi yang lebih efisien, dan pergerakan yang lebih adaptif untuk menghindari jebakan, maka bot dapat tetap agresif, tapi dapat melakukan pergerakan yang lebih cerdas sehingga energi (HP) dapat lebih terjaga sepanjang permainan.

Untuk bot DonatMerah, peningkatan dalam bot yang dapat dilakukan adalah dengan penambahan prediksi untuk penembakan agar dapat lebih akurat. Adanya kejanggalan ketika

bot menabrak dinding bagian bawah dan mencoba bergerak melingkar kembali. Hal itu mungkin terjadi karena bot kebingungan apakah akan bergerak tangensial dengan berlawanan arah jarum jam atau searah jarum jam.

Untuk bot Avenger, peningkatan yang dapat dilakukan adalah dengan penerapan analisis multi-parameter dalam proses pengambilan keputusan, sehingga bot tidak hanya mengandalkan perhitungan jarak dan nilai *firePower*, tetapi juga mempertimbangkan variabel lain seperti kecepatan musuh, tren pergerakan, dan kondisi medan pertempuran secara real-time. Penerapan algoritma prediktif yang lebih kompleks, seperti filter Kalman, dapat meningkatkan akurasi estimasi posisi musuh sehingga respons bot lebih tepat sasaran dan adaptif. Selain itu, penyempurnaan logika pergantian target, dengan menetapkan ambang batas yang lebih adaptif untuk switching targets dan pengaturan variabel *rageFactor*, dapat mengoptimalkan alokasi Energi secara efisien, sehingga bot dapat memaksimalkan potensi serangan dan pertahanannya dalam menghadapi lawan yang memiliki taktik adaptif dan kompleks.

Untuk bot Asteroid Destroyer, peningkatan yang dapat dilakukan adalah dengan menerapkan multi-parameter dalam pemilihan target, seperti dengan menambahkan parameter jarak relatif, kecepatan, dan tren pergerakan musuh, sehingga bot tidak hanya bergantung pada perbandingan energi. Selanjutnya, optimalisasi mode transisi antara *Chasing* dan *Strafing* dapat dilakukan melalui penggunaan algoritma prediktif yang lebih kompleks seperti filter Kalman, untuk meningkatkan akurasi estimasi posisi musuh dan mengurangi pemborosan peluru. Selain itu, penyempurnaan logika cooldown dan evaluasi konteks pertempuran secara real-time akan memungkinkan alokasi energi dan gerakan yang lebih adaptif, sehingga bot dapat bekerja dengan lebih stabil dan responsif dalam menghadapi musuh dengan taktik adaptif.

LAMPIRAN

Berikut lampiran *checklist*.

| 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. | ✓ | |

Berikut lampiran tautan.

| | |
|-------------------|---|
| Repository GitHub | github.com/Azekhiel/Tubes1_LulusPanganril |
| Video YouTube | Tubes 1 Stima 2025 Kelompok LulusPanganril |

DAFTAR PUSTAKA

Mount, D., & Eastman, R. (2018). *AI Decision Making [Catatan kuliah]*. Diakses 23 Maret 2025, dari

<https://www.cs.umd.edu/class/spring2018/cmsc425/Lects/lect21-ai-dec-making.pdf>

Munir, R. (2024). *Algoritma Greedy (Bagian 1) [PDF]*. Diakses 23 Maret 2025, dari
[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)

Munir, R. (2024). *Algoritma Greedy (Bagian 2) [PDF]*. Diakses 23 Maret 2025, dari
[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)

Munir, R. (2024). *Algoritma Greedy (Bagian 3) [PDF]*. Diakses 23 Maret 2025, dari
[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)

The Robowiki. (n.d.). *Beyond the Basics*. Diakses 23 Maret 2025, dari
<https://robocode-dev.github.io/tank-royale/tutorial/beyond-the-basics.html#the-robowiki>