Laporan Tugas Besar 1 IF2211 Strategi Algoritma
Semester II tahun 2024/2025

Disusun Oleh :
13523134 Sebastian Enrico Nathanael
13523142 Nathanael Rachmat
13523154 Theo Kurniady

Program Studi Teknik Informatika

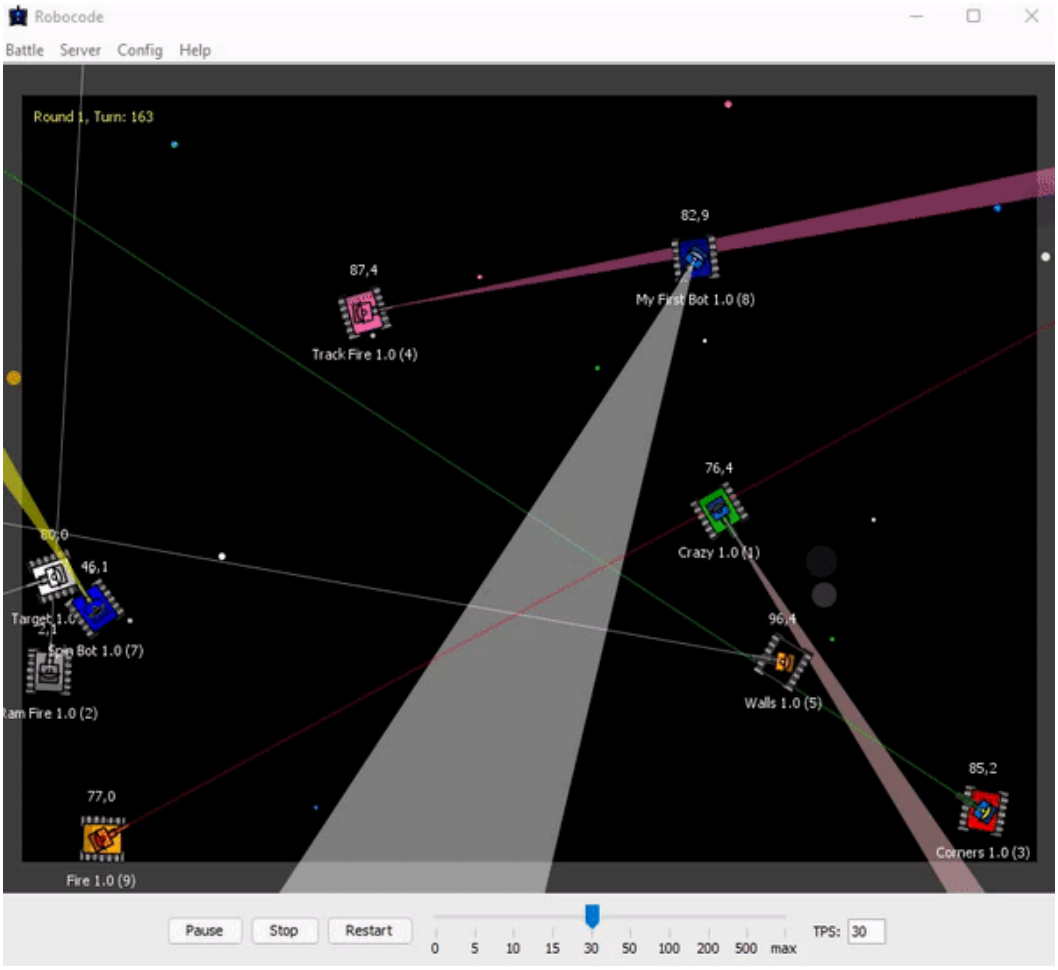
Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2025

Daftar Isi

Bab I
Deskripsi Tugas



Gambar 1.1 Robocode Tank Royale

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

Pada Tugas Besar pertama Strategi Algoritma ini, 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 melewatkan turn tersebut. Jika bot melewatkan turn, ia tidak akan bisa menyesuaikan gerakannya atau menembakkan senjatanya karena server tidak menerima perintah tepat waktu sebelum turn berikutnya dimulai.

3. Energi

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

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

4. Peluru

   Semakin banyak energi (daya tembak) yang digunakan untuk menembakkan peluru, semakin berat peluru tersebut dan semakin lambat gerakannya. Namun, peluru yang lebih berat juga menghasilkan lebih banyak kerusakan dan memungkinkan bot mendapatkan lebih banyak energi saat mengenai bot musuh.

   Seperti disebutkan sebelumnya, peluru yang lebih berat akan bergerak lebih lambat. Ini berarti akan membutuhkan waktu lebih lama untuk mencapai target, meningkatkan risiko peluru tidak mengenai sasaran. Sebaliknya, peluru yang lebih ringan bergerak lebih cepat, sehingga lebih mudah mengenai target, tetapi peluru ringan tidak memberikan banyak poin energi saat mengenai bot musuh.

5. Panas Meriam (Gun Heat)

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

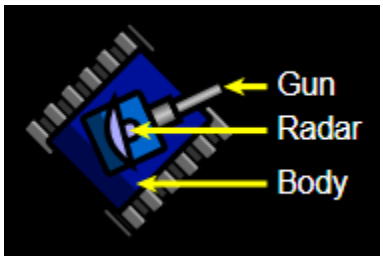
6. Tabrakan

Perlu diperhatikan bahwa bot akan menerima kerusakan jika menabrak dinding (batas arena), yang disebut wall damage. Hal yang sama juga terjadi jika bot bertabrakan dengan bot lain.

Jika bot menabrak bot musuh dengan bergerak maju, ini disebut ramming (menabrak dengan sengaja), yang akan memberikan sedikit skor tambahan bagi bot yang menyerang.

7. Bagian Tubuh Tank

   Tubuh tank terdiri dari 3 bagian:



   Body adalah bagian utama dari tank yang digunakan untuk menggerakkan tank.

   Gun digunakan untuk menembakkan peluru dan dapat berputar bersama *body* atau independen dari *body*.

   Radar digunakan untuk memindai posisi musuh dan dapat berputar bersama *body* atau independen dari *body*.

8. Pergerakan

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

9. Berbelok

   Seperti yang disebutkan sebelumnya, bagian tubuh, turret (meriam), dan radar dapat berputar secara independen satu sama lain. Jika turret atau radar tidak diputar, maka keduanya akan mengarah ke arah yang sama dengan tubuh bot.

   Setiap bagian tubuh memiliki kecepatan putar yang berbeda. Radar adalah bagian tercepat dan dapat berputar hingga 45 derajat per giliran, yang berarti dapat berputar 360 derajat dalam 8 giliran. Turret dan meriam dapat berputar hingga 20 derajat per giliran.

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

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

10. Pemindaian

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

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

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

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

11. Skor

    Pada akhir pertempuran, setiap bot akan diranking berdasarkan total skor yang diperoleh masing-masing bot selama keseluruhan pertempuran. Tentunya, tujuan utama pada tugas

besar ini adalah membuat bot yang memberikan skor setinggi mungkin. Berikut adalah rincian komponen skor pada pertempuran:

- **Bullet Damage:** Bot mendapatkan **poin sebesar *damage*** yang dibuat kepada bot musuh menggunakan peluru.
- **Bullet Damage Bonus:** Apabila peluru berhasil membunuh bot musuh, bot mendapatkan **poin sebesar 20% dari *damage*** yang dibuat kepada musuh yang terbunuh.
- **Survival Score:** Setiap ada bot yang mati, bot lainya yang masih bertahan pada ronde tersebut mendapatkan **50 poin.**
- **Last Survival Bonus:** Bot terakhir yang bertahan pada suatu ronde akan mendapatkan **10 poin** dikali dengan banyaknya musuh.
- **Ram Damage:** Bot mendapatkan **poin sebesar 2 kalinya *damage*** yang dibuat kepada bot musuh dengan cara menabrak.
- **Ram Damage Bonus:** Apabila musuh terbunuh dengan cara ditabrak, bot mendapatkan **poin sebesar 30% dari *damage*** yang dibuat kepada musuh yang terbunuh.

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

Untuk informasi lebih lengkap, silahkan buka dokumentasi Tank Royale pada link berikut.

Bab II
Dasar Teori

2.1 Algoritma Greedy

Algoritma Greedy adalah pendekatan dalam pemrograman yang memecahkan persoalan optimasi dengan cara yang tampaknya “rakus”. Pendekatan ini berfokus pada menyelesaikan masalah optimasi dengan cara memilih solusi terbaik pada setiap langkah tanpa mempertimbangkan konsekuensi jangka panjang. Pengambilan keputusan sekarang dengan harapan bahwa setiap langkah akan membawa kita lebih dekat ke solusi akhir yang optimal. Dalam konteks greedy, kita selalu memilih opsi yang paling menguntungkan saat ini tanpa mempertimbangkan konsekuensi di masa depan. Ini mirip dengan mengambil sejumlah uang tunai yang tersedia dari mesin ATM tanpa memikirkan bagaimana pengeluaran itu akan memengaruhi saldo akhir.

2.2 Robocode Tank Royale

Algoritma greedy dapat diimplementasikan ke dalam berbagai permainan seperti Robocode Tank Royale. Permainan ini mewajibkan pemain membuat bot untuk melakukan pertarungan otomatis dengan bot lainnya. Untuk membuat sebuah bot, ada beberapa hal yang perlu diperhatikan, terutama cara kerjanya. Setiap bot memiliki deskripsi yang diperoleh dari file .json, yang mengandung berbagai hal seperti nama, versi, pembuat, dan banyak lagi. Bot melakukan pertarungan otomatis dengan looping pada fungsi run() pada umumnya. Untuk melakukan berbagai aksi dalam permainan ini, terdapat API yang menyediakan metode-metode yang dapat digunakan dalam game ini.

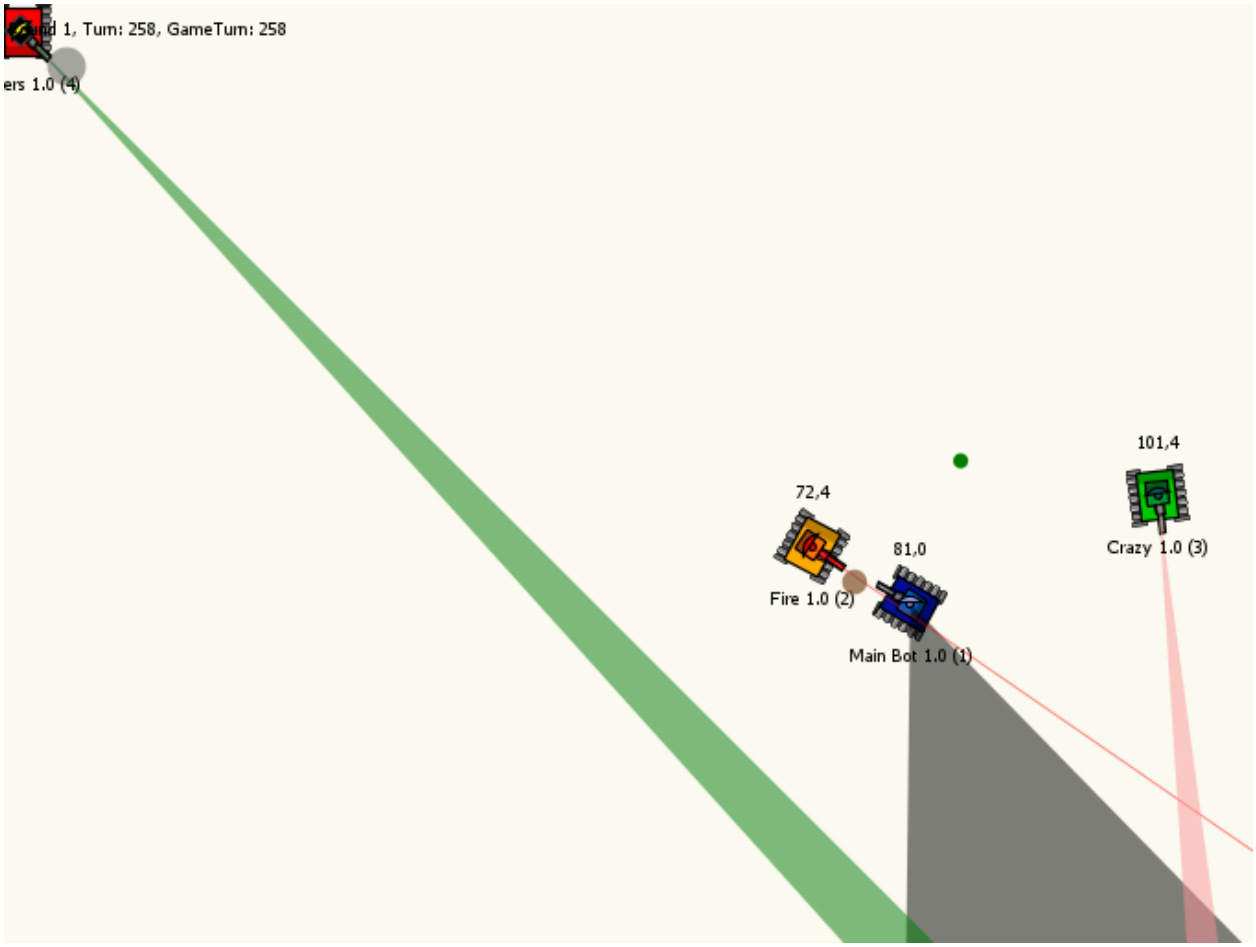
Setiap bot memiliki 3 bagian yang bisa dikendalikan, yaitu badan, senapan/gun, dan radar. Dalam loop, bot dapat diprogram untuk melakukan berbagai hal dengan bagian-bagian tersebut. Badan adalah *hitbox* dari setiap bot yang harus dipertahankan pemain. Pemain dapat menggunakan badan bot untuk maju ke depan/belakang dan belok kiri/kanan dalam satuan derajat. Untuk melakukan serangan ke musuh, diperlukan senapan yang mampu menyerang dari jauh. Senapan dapat menembakan peluru, yang *damage*nya dapat diatur untuk menyesuaikan energi yang dikeluarkan. Senapan juga dapat dibelokkan seperti badan. Untuk mendeteksi musuh, diperlukan radar. Radar adalah bagian yang penting dalam algoritma greedy karena variasi dari taktik yang dapat diperoleh dari mendeteksi bagian musuh, seperti ID, jaraknya, dan energinya.

Bab III
Aplikasi Strategi Greedy

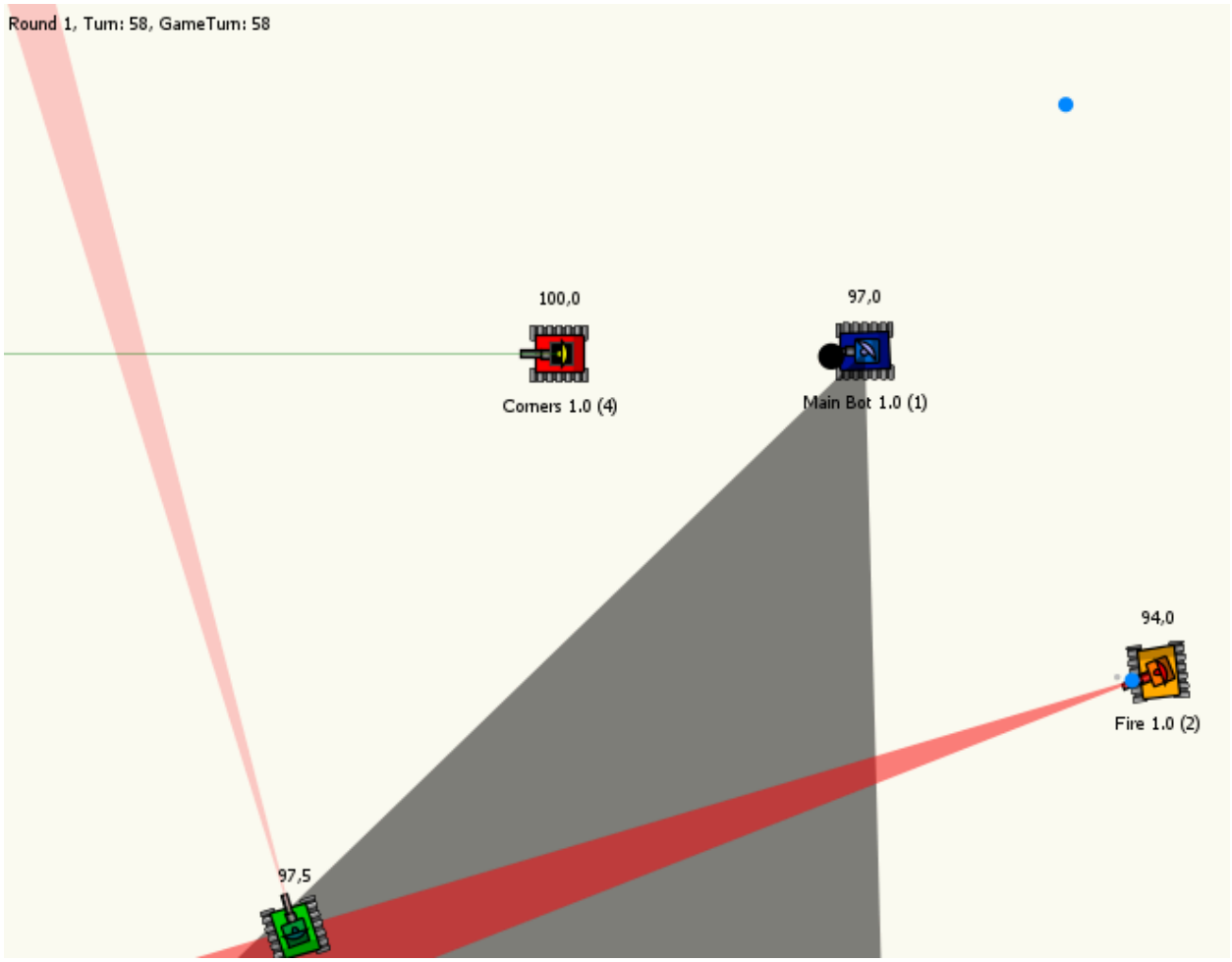
3.1 Eksplorasi Alternatif Solusi Greedy

3.1.1 Greedy by Nearest Target

Strategi Greedy by Nearest Target adalah pendekatan yang mana bot selalu mengejar dan menyerang musuh terdekat berdasarkan jarak, tanpa mempertimbangkan langkah jangka panjang. Setiap kali bot memindai lawan, ia memperbarui daftar musuh yang terdeteksi dan memilih target dengan jarak paling pendek menggunakan perhitungan Euclidean. Setelah target ditentukan, bot mengarahkan tubuh dan senjatanya ke arah musuh tersebut serta bergerak mendekatinya hingga mencapai jarak optimal untuk menembak. Strategi ini efektif dalam situasi pertempuran cepat, namun memiliki kelemahan karena bot dapat terjebak dalam pola yang mudah ditebak, seperti terus-menerus mengejar satu musuh tanpa memperhitungkan ancaman dari bot lain atau kondisi medan.



Gambar 3.1.1.1 Ilustrasi bot akan menarget bot dengan jarak terdekat



Gambar 3.1.1.2 Ilustrasi bot akan menarget bot dengan jarak terdekat, menghiraukan jumlah energi bot lain

a. Mapping Elemen Greedy

Elemen Algoritma	**Deskripsi**
Himpunan Kandidat	Seluruh bot musuh yang terdeteksi radar (satu map)
Himpunan Solusi	Sebuah bot musuh yang memiliki jarak terdekat dengan bot ini
Fungsi Solusi	Bot musuh yang terdekat dengan bot ini akan menjadi target sehingga bot akan mulai mendekat atau menjaga jarak dengan bot musuh dan mulai menyerang

Fungsi Seleksi	Pilih bot musuh dengan membandingkan jarak antara semua bot yang ter-scan radar dan memilih bot dengan jarak terpendek
Fungsi Kelayakan	Memeriksa bila bot musuh ada di dalam jangkauan tembak dan belum mati
Fungsi Objektif	Memaksimalkan poin dengan menarget bot terdekat untuk meminimalisir waktu gerak dan menembak pada jarak yang aman untuk menembak

b. Analisis Efisiensi Solusi

Dengan menggunakan strategi ini, bot akan mencari musuh terdekat dan musuh yang telah mati akan hilang dari list target. Strategi ini bergantung pada loop utama yang melibatkan scan radar 360° untuk scan seluruh musuh dalam map (O(1)), mengupdate daftar *enemies* (O(n)), memilih musuh terdekat sebagai target (O(n)), menjaga jarak dengannya, dan menembak. Setiap loop yang terlibat dalam algoritma ini hanya berlangsung secara linear. Kompleksitas waktu algoritma:

$$\mathbf{T(n) = O(n)}$$

N = jumlah musuh yang terdeteksi oleh radar dan tersimpan dalam daftar *enemies*

Struktur utama yang digunakan untuk mencatat daftar musuh yang ada di dalam map adalah tipe data EnemyInfo, yang menunjukkan nama dan koordinat setiap musuh. Kompleksitas memori algoritma:

$$\mathbf{S(n) = O(n)}$$

N = jumlah musuh yang terdeteksi oleh radar dan tersimpan dalam daftar *enemies*

c. Analisis Efektivitas Solusi

Strategi Greedy by Nearest Target memiliki **kelebihan** sebagai berikut.

- Menargetkan musuh terdekat untuk meminimalisir waktu pergerakan.
- Menjaga jarak optimal untuk menjaga posisi optimal.

Strategi Greedy by Nearest Target memiliki **kekurangan** sebagai berikut.

- Bot akan sulit dalam menentukan target bila ada 2 atau lebih bot yang memiliki jarak yang sama
- Bot dapat merubah target bila ada bot lain yang mendekat sehingga sulit menghabisi bot

3.1.2 Greedy by Lowest Health

Strategi Greedy by Lowest Health merupakan algoritma dimana bot selalu memilih musuh dengan HP (*health point*) paling rendah sebagai target serangan utama. Dengan begitu maka lawan dengan *health* paling rendah dapat dikalahkan terlebih dahulu. Dimulai dengan bot melakukan scan terhadap musuh di sekitarnya dan mendapatkan data health dari mereka. Menyimpan data health dan jarak setiap bot yang terdeteksi. Lalu bot akan mulai menargetkan dan menembak bot dengan health terendah. Menyerang hingga target mati, Jika target mati langsung mencari musuh lain dengan health terendah.

a. Mapping Elemen Greedy

Elemen Algoritma	**Deskripsi**
Himpunan Kandidat	Seluruh bot musuh yang berhasil terdeteksi melalui radar pada arena
Himpunan Solusi	Satu bot musuh terakhir yang terdeteksi dan kemudian dikunci sebagai target untuk diserang.
Fungsi Solusi	Bot dengan *health* terendah akan menjadi taget dan akan mulai menembak bot tersebut.
Fungsi Seleksi	Pilih bot dengan health terkecil terlebih dahulu, kemudian yang terdekat jika ada health yang sama.
Fungsi Kelayakan	Target layak jika berhasil dideteksi, belum mati, dan menembak jika masih dalam jangkauan radar.
Fungsi Objektif	Mengurangi jumlah lawan dengan menyerang bot yang paling mudah dikalahkan yaitu bot dengan health terendah. Menjaga kelangsungan hidup (survivability) dan memaksimalkan damage sehingga bisa mendapatkan poin dengan maksimal.

b. Analisis Efisiensi Solusi

Efisiensi solusi dalam strategi “Greedy by Lowest Health” pada Robocode Tank Royale dapat dinilai dari kompleksitas waktu dan penggunaan sumber daya dalam menentukan target dan mengeksekusi serangan. Algoritma ini menggunakan struktur data

dictionary untuk menyimpan informasi musuh, sehingga pencarian target menggunakan operasi OrderBy(), yang memiliki kompleksitas algoritma :

O(n log n)

n = jumlah musuh yang terdeteksi.

Proses seleksi target dilakukan setiap kali ada bot terdeteksi, yang dapat meningkatkan overhead jika banyak musuh dalam pertandingan. Namun, karena strategi ini memilih target berdasarkan health terendah dan jarak terdekat, keputusan dibuat dengan cepat tanpa eksplorasi global, sehingga efisiensi tetap lebih baik dibandingkan pendekatan berbasis pencarian penuh.

c. Analisis Efektivitas Solusi

Strategi Greedy by Lowest Health memiliki **kelebihan** sebagai berikut.

- Menargetkan musuh dengan health terendah, sehingga lebih cepat mengeliminasi lawan yang lemah.
- Mengurangi jumlah musuh secara bertahap, sehingga semakin sedikit bot yang tersisa dalam pertandingan.
- Memanfaatkan musuh yang sedang dalam kondisi kritis, mengurangi ancaman dari bot yang memiliki daya tahan rendah.

Strategi Greedy by Lowest Health memiliki **kekurangan** sebagai berikut.

- Tidak mempertimbangkan ancaman dari musuh yang lebih kuat, sehingga dapat diserang oleh bot dengan strategi agresif.
- Jika musuh yang memiliki health terendah berada jauh, bot akan menghabiskan waktu untuk mendekati target dan rentan diserang oleh musuh lainnya.

3.1.3 Greedy by Evade

Strategi Greedy by Evading merupakan sebuah algoritma yang mana sebuah bot selalu bergerak melingkari robot musuh yang terakhir terkena *scan*. Hal ini mewujudkan segala peluru yang ditembak robot musuh tidak mencapai kita, memanfaatkan pengurangan energi melalui peluru musuh yang terkena tembok. Saat *match* dimulai, robot ini akan '*reset*' untuk memutar tembakan (sekaligus arah *scanner*) ke arah 90 derajat dari arah robot, memindai seluruh arena, dan putar badan robotnya sebesar 90 derajat dari arah robot musuh. Hal ini membuat pergerakan lingkaran yang dilakukan lebih mudah dilakukan. Bot akan terus bergerak searah jarum jam maupun sebaliknya, dan bergerak sesuai dengan persamaan matematika untuk menentukan *turn rate* dari bot kita:

$$TurnRate = \frac{180 \times Speed}{\pi \times radius}$$

Sebagai tambahan, ada beberapa konfigurasi seperti pergantian arah pergerakan (jarum jam dan sebaliknya setiap kali menabrak tembok, menabrak musuh, dan terkena peluru. Selain

itu jika dalam sekian waktu radar tidak menemukan robot musuh, robot ini akan *reset* seperti ketika *match* dimulai. Tembakan juga dilakukan oleh bot ini, tetapi *power*-nya ditentukan berdasarkan jarak, semakin dekat maka *power*-nya akan semakin besar secara eksponensial.

a. Mapping Elemen Greedy

Elemen Algoritma	**Deskripsi**
Himpunan Kandidat	Seluruh bot musuh yang berhasil terdeteksi melalui radar atau akibat tabrakan.
Himpunan Solusi	Satu bot musuh terakhir yang terdeteksi dan kemudian dikunci sebagai target.
Fungsi Solusi	Bot akan mengunci musuh terakhir yang terdeteksi sebagai target, lalu mengatur arah gerakan dan kecepatan agar dapat bergerak mengorbit musuh tersebut pada radius tertentu.
Fungsi Seleksi	Greedy terhadap musuh terakhir yang terlihat (baik dari radar atau tabrakan), tanpa mempertimbangkan kondisi musuh lain.
Fungsi Kelayakan	Target layak jika berhasil dideteksi, belum mati, dan menembak jika sudah sesuai dan jarak antar bot sudah cukup dekat.
Fungsi Objektif	Menjaga kelangsungan hidup (survivability) dan memaksimalkan damage dengan terus bergerak secara melingkar (untuk menghindari tembakan langsung) dan tetap menembak dari jarak relatif aman. Strategi ini juga menghindari diam terlalu lama tanpa target dengan melakukan reset jika tidak menemukan musuh dalam waktu lama.

b. Analisis Efisiensi Solusi

Dengan menggunakan strategi ini, bot tidak memilih target berdasarkan keseluruhan musuh di map, namun langsung mengunci musuh terakhir yang terdeteksi oleh radar atau saat terjadi tabrakan. Setelah target di-lock, bot akan bergerak secara melingkar mengitari musuh tersebut sambil menembak secara adaptif berdasarkan jarak.

Strategi ini bergantung pada:

- Loop utama yang menjalankan radar 360° untuk mendeteksi musuh (O(1))
- Lock langsung ke satu musuh terakhir tanpa membandingkan kandidat lain (O(1))
- Hitung arah orbit dan radius (O(1))

- Tembakan pintar (SmartFire) berbasis jarak (O(1))

Oleh karena itu, kompleksitas waktu algoritma:

$$\mathbf{T(n) = O(n)}$$

N = jumlah musuh yang terdeteksi oleh radar dan tersimpan dalam daftar *enemies*

Setiap loop hanya memproses satu musuh aktif yang terdeteksi terakhir, tanpa penyimpanan list musuh. Oleh karena itu, kompleksitas memori algoritma:

$$\mathbf{S(n) = O(n)}$$

N = jumlah musuh yang terdeteksi oleh radar dan tersimpan dalam daftar *enemies*

c. Analisis Efektivitas Solusi

Strategi Greedy by Evading memiliki **kelebihan** sebagai berikut.

- Efektif menghindari peluru musuh karena gerakan orbit membuat bot sulit diprediksi dan ditembak langsung.
- Menjaga jarak aman sambil tetap bisa menyerang melalui SmartFire.
- Adaptif terhadap serangan karena bot akan mengganti arah jika terkena peluru, tabrakan, atau menabrak tembok.
- Reset otomatis jika dalam waktu tertentu musuh tidak ditemukan, memastikan bot tidak diam terlalu lama tanpa target.

Strategi Greedy by Evading memiliki **kekurangan** sebagai berikut.

- Hanya menarget satu musuh terakhir, tanpa mempertimbangkan musuh lain yang mungkin lebih lemah atau lebih dekat.
- Ada kemungkinan terjebak di ujung arena, membuat robot merubah arah pergerakan secara mengulang dan menguras energi robot.
- Bergantung pada deteksi radar, jika musuh terlalu cepat atau menghindar, bot bisa kehilangan target dan harus reset lagi.
- Tidak agresif dalam penyisiran seluruh musuh; hanya fokus pada satu titik saat itu saja.

3.1.4 Greedy by Aggressive Circling

Gereja Greedy yang disebut "Aggressive Circling" adalah algoritma yang mana pergerakan bot selalu ke arah yang diperkirakan berdasarkan sebuah persamaan lingkaran di sekeliling musuh yang dipantulkan. Setiap kali radar menemukan musuh, radar diputar ke arah sebaliknya agar radar tetap mengarah ke musuh. Bot akan menembak jika sudut tembakan sudah tepat. Posisi target lingkaran dihitung dengan memanfaatkan sudut dasar antara bot dan musuh, ditambah offset sebesar ±45 derajat tergantung arah gerak. Hal ini membuat bot memutari musuh dalam pola spiral atau orbit yang agresif sambil menembak pelurunya. Sebagai tambahan, terdapat konfigurasi smart fire, yang mana *power* tembakan naik secara eksponensial tergantung jarak antara kedua bot. Selain itu, terdapat konfigurasi *max speed* agar robot dapat berputar dengan smooth jika sudut bot yang di-*scan* terlalu jauh.

a. Mapping Elemen Greedy

Elemen Algoritma	**Deskripsi**
Himpunan Kandidat	Seluruh bot musuh yang terdeteksi melalui radar
Himpunan Solusi	Sebuah bot musuh yang terakhir terdeteksi dan menjadi target oleh dan di-*lock* oleh bot kita.
Fungsi Solusi	Menetapkan bot musuh yang terakhir terdeteksi sebagai target dan menentukan posisi orbit di sekitar musuh dengan offset ±45°, lalu menggerakkan bot ke posisi tersebut sambil menembak bila memungkinkan.
Fungsi Seleksi	Memilih musuh berdasarkan hasil *scan* dan tabrakan musuh, dan mengunci target tersebut langsung tanpa membandingkan kandidat lain (greedy terhadap yang paling baru terlihat)
Fungsi Kelayakan	Musuh dianggap layak jika berhasil di-*scan*, menembak jika sudut tembakan mengarah ke musuh, dan jika musuh belum mati.
Fungsi Objektif	Memaksimalkan jumlah poin dan *survivability* dengan melingkari musuh dari jarak tetap, menembak jika arah tembakan searah dan menembak dengan kekuatan berdasarkan jarak, serta mengatur arah dan kecepatan agar *lock* ke target bot.

b. Analisis Efisiensi Solusi
Strategi ini hanya menyimpan satu target musuh yang aktif (terakhir terlihat). Tidak perlu menyimpan atau memperbarui daftar musuh, dan semua perhitungan dilakukan secara lokal terhadap satu musuh. Bot melakukan orbit berdasarkan trigonometri sederhana (atan2, cos, sin) yang bersifat konstan. Pengulangan utama yang dilakukan oleh bot seperti berikut ini:
- Deteksi musuh (O(1))
- Perhitungan sudut dan posisi orbit (O(1))
- Penyesuaian arah dan kecepatan (O(1))

Oleh karena itu, kompleksitas waktu yang dihasilkan adalah sebagai berikut:

$$T(n) = O(1)$$

n = jumlah musuh yang terlihat di radar

Bot hanya menyimpan informasi target terakhir (posisi koordinat x dan y). Maka, kompleksitas memori yang dihasilkan adalah sebagai berikut ini:

$$S(n) = O(1)$$

n = jumlah musuh yang terlihat di radar

c. Analisis Efektivitas Solusi

Berdasarkan

Strategi Greedy by Aggressive Circling memiliki **kelebihan** sebagai berikut.

- Tidak perlu menyimpan atau membandingkan banyak musuh.
- Bergerak secara terus-menerus mengorbit target, membuat posisi sulit diprediksi oleh musuh diam.
- Bisa menyesuaikan kecepatan dan arah berdasarkan posisi relatif ke musuh.

Strategi Greedy by Aggressive Circling memiliki **kekurangan** sebagai berikut.

- Hanya fokus ke satu musuh terakhir, membuat bot tidak bisa memindai di sekitar dan dapat terkena tembak dari musuh lain.
- Tidak memilih target optimal, hanya berdasarkan musuh yang terlihat terakhir
- Jika terkena tembok, bot harus memutar badannya terlebih dahulu agar bisa lanjut bergerak, membuat bot *vulnerable* dalam posisi tersebut

3.2 Strategi Greedy Utama

Berdasarkan berbagai solusi yang telah disampaikan, terdapat kelebihan dan kekurangan masing-masing algoritma. Dikarenakan lokasi *spawn* yang *random* sehingga situasi setiap ronde tidak tentu, maka bot dapat dipertimbangkan dengan pendekatan greedy yang sesuai dengan situasi yang terjadi di map dan menentukan aksi yang akan diambil.

Bab IV
Implementasi dan Pengujian

4.1 Implementasi Alternatif Solusi Greedy

4.1.1 Greedy by Nearest Target

```
Program Bot Menggunakan Algoritma Greedy By Nearest Target

KAMUS

List_of_EnemyInfo enemies : { Id : integer,
                              X : double,
                              Y : double }
Double TARGET_DISTANCE ← 70
GreedyNTBot ← inisialisasi("GreedyNTBot.json")

function BearingTo(double x, double y)
{Mengembalikan sudut relatif antara arah bot dengan titik (x, y)}

function DistanceTo(double x, double y)
{Mengembalikan jarak antara bot dengan titik (x, y)}

procedure TurnRadarRight(double bearing)
{Memutar radar ke kanan sebesar angle derajat}
{I/S: menerima input bearing, yaitu sudut untuk diputar}
{F/S: Radar diputar ke kanan sebesar sudut bearing}

procedure SetTurnLeft(double bearing)
{Memutar bot ke kiri sebesar angle derajat}
{I/S: menerima input bearing, yaitu sudut untuk diputar}
{F/S: Radar diputar ke kiri sebesar sudut bearing}

procedure SetForward(double x)
{Mengatur bot agar bergerak maju ke arah hadapan sejauh x satuan}
{I/S: menerima input x, yaitu jarak yang akan ditempuh ke depan}
{F/S: Bot mulai bergerak maju sejauh x satuan}

procedure SetBack(double x)
{Mengatur bot agar bergerak mundur dari arah hadapan sejauh x
satuan}
{I/S: menerima input x, yaitu jarak yang akan ditempuh ke belakang}
{F/S: Bot mulai bergerak mundur sejauh x satuan}

procedure Fire(double power)
{Menembakkan peluru dengan kekuatan tertentu}
{I/S: menerima input power, yaitu kekuatan peluru dalam satuan
```

```
energi (biasanya antara 0.1 hingga 3)}
{F/S: Bot menembakkan peluru ke arah senjata dengan kekuatan power}


ALGORITMA

procedure Run()   // gerakan utama
    Clear enemies
    while isRunning do
        TurnRadarRight(360°)
        If (enemies.count > 0) then
            MoveToNearestEnemy()

procedure OnScannedBot(event e)   // menentukan bot menembak atau
update list musuh
    distance ← DistanceTo(e.X, e.Y)
    bearing ← BearingTo(e.X, e.Y)

    if -10 < bearing < 10 AND distance < 300 then
        Fire(3)

    scannedId ← e.ScannedBotId
    found ← false

    i traversal [0..enemies.count]
        If enemies[i].Id = scannedId then
            enemies[i] ← new EnemyInfo(scannedId, e.X, e.Y)
            found ← true
            Break

    if found = false then
        Add new enemy (scannedId, e.X, e.Y) to enemies list

procedure OnBotDeath(event e)
    Remove enemy from enemies where enemy.Id = e.VictimId

procedure MoveToNearestEnemy()
    if enemy list is empty then
        →

    nearestEnemy ← enemies[0]
    minDistance ← DistanceTo(nearestEnemy.X, nearestEnemy.Y)

    enemy traversal enemies:
        distance ← DistanceTo(enemy.X, enemy.Y)
        if distance < minDistance then
            minDistance ← distance
            nearestEnemy ← enemy

    enemyBearing ← BearingTo(nearestEnemy.X, nearestEnemy.Y)
```

```
    TurnLeft(NormalizeBearing(enemyBearing))  // mengarahkan bot ke
lawan

    if minDistance > TARGET_DISTANCE then
        Forward(minDistance - TARGET_DISTANCE)
    else
        Back(TARGET_DISTANCE - minDistance)

Function NormalizeBearing(double angle)
    while angle < -180 do
        angle ← angle + 360
    while angle > 180 do
        angle ← angle - 360
    → angle

procedure OnHitBot(event e)
    bearing ← BearingTo(e.X, e.Y)

    if -10 < bearing < 10 then
        Fire(3)

    if e.IsRammed = true then
        TurnLeft(10)

procedure Main()
    GreedyNTBot.Start()
```

4.1.2 Greedy by Lowest Health

```
PROGRAM Bot dengan Algoritma Greedy by Lowest Health


KAMUS
List_of_EnemyInfo enemies : { Id : integer,
                              Energy : double,
                              X : double,
                              Y : double }

function BearingTo(double x, double y)
{Mengembalikan sudut relatif antara arah bot dengan titik (x, y)}

function DistanceTo(double x, double y)
{Mengembalikan jarak antara bot dengan titik (x, y)}

procedure TurnRadarRight(double bearing)
{Memutar radar ke kanan sebesar angle derajat}
{I/S: menerima input bearing, yaitu sudut untuk diputar}
{F/S: Radar diputar ke kanan sebesar sudut bearing}
```

```
procedure SetTurnLeft(double bearing)
{Memutar bot ke kiri sebesar angle derajat}
{I/S: menerima input bearing, yaitu sudut untuk diputar}
{F/S: Radar diputar ke kiri sebesar sudut bearing}

procedure SetForward(double x)
{Mengatur bot agar bergerak maju ke arah hadapan sejauh x satuan}
{I/S: menerima input x, yaitu jarak yang akan ditempuh ke depan}
{F/S: Bot mulai bergerak maju sejauh x satuan}

procedure SetBack(double x)
{Mengatur bot agar bergerak mundur dari arah hadapan sejauh x satuan}
{I/S: menerima input x, yaitu jarak yang akan ditempuh ke belakang}
{F/S: Bot mulai bergerak mundur sejauh x satuan}

procedure Fire(double power)
{Menembakkan peluru dengan kekuatan tertentu}
{I/S: menerima input power, yaitu kekuatan peluru dalam satuan energi
(biasanya antara 0.1 hingga 3)}
{F/S: Bot menembakkan peluru ke arah senjata dengan kekuatan power}


Algoritma
LowHPBot ← inisialisasi("LowHPBot.json")

procedure Run()
    Clear enemies list
    AdjustRadarForBodyTurn ← false

    while IsRunning do
        TurnRadarRight(360)

        if enemies.count > 1 then
            MoveToTarget()
        else
            target ← enemies[0]
            distance ← DistanceTo(target.X, target.Y)
            bearing ← BearingTo(target.X, target.Y)

            SetTurnRight(NormalizeBearing(bearing))

            if distance > LIMIT_TARGET_DISTANCE then
                SetTurnLeft(NormalizeBearing(bearing))
                SetForward(distance - LIMIT_TARGET_DISTANCE)
            else if distance < (LIMIT_TARGET_DISTANCE / 2) then
                SetBack(30)

            if -10 < bearing < 10 AND distance < 300 then
                Fire(3)


procedure OnScannedBot(event e)
    distance ← DistanceTo(e.X, e.Y)
    bearing ← BearingTo(e.X, e.Y)

    if -10 < bearing < 10 AND distance < 300 then
```

```
        Fire(3)

    scannedId ← e.ScannedBotId
    found ← false

    i traversal [0..enemies.count]
        If enemies[i].Id = scannedId then
            enemies[i] ← new EnemyInfo(scannedId, e.X, e.Y)
            found ← true
            Break

    if found = false then
        Add new enemy (scannedId, e.X, e.Y) to enemies list


procedure OnBotDeath(event e)
    Remove enemy from enemies where enemy.Id = e.VictimId


procedure MoveToTarget()



procedure MoveToTarget()
    if enemies is empty then
        return

    lowestHPenemy ← enemies[0]
    minHP ← lowestHPenemy.Energy

    for each enemy in enemies do
        HP ← enemy.Energy
       if HP < minHP then
            minHP ← HP
            lowestHPenemy ← enemy

    minDistance ← DistanceTo(lowestHPenemy.X, lowestHPenemy.Y)
    enemyBearing ← BearingTo(lowestHPenemy.X, lowestHPenemy.Y)

    SetTurnLeft(NormalizeBearing(enemyBearing))

    if minDistance > LIMIT_TARGET_DISTANCE then
        SetForward(minDistance - LIMIT_TARGET_DISTANCE)
    else
        SetBack(LIMIT_TARGET_DISTANCE - minDistance)


Function NormalizeBearing(double angle)
    while angle < -180 do
        angle ← angle + 360
    while angle > 180 do
        angle ← angle - 360
    → angle

procedure OnHitBot(event e)
```

```
    bearing ← BearingTo(e.X, e.Y)

    if -10 < bearing < 10 then
        Fire(3)

    if e.IsRammed = true then
        TurnLeft(10)


procedure Main()
    LowHPBot.Start()
```

4.1.3 Greedy by Evade

```
PROGRAM Gevade
{Program ini menggunakan pendekatan greedy dengan strategi menghindar (evade)
dengan gerakan melingkar di sekitar musuh yang terakhir terlihat.}
KAMUS
targetX, targetY, currentX, currentY, dx, dy, radius, bearing, desiredTurn:
double
shrinkRate: double ← 25
clockwise: boolean
resetting: boolean
lockIn: boolean
turns: integer

function DistanceTo(double x, double y)
{Mengembalikan jarak antara bot dengan titik (x, y)}

function BearingTo(double x, double y)
{Mengembalikan sudut relatif antara arah bot dengan titik (x, y)}

function Normalize(double bearing)
{Mengembalikan sudut dalam rentang [-180°, 180°] dari sudut input}

procedure TurnLeft(double angle)
{Memutar bot ke kiri sebesar sudut tertentu}
{I/S: menerima input angle, yaitu sudut rotasi ke kiri dalam derajat}
{F/S: Bot diputar ke kiri sebesar angle derajat}

procedure TurnGunLeft(double angle)
{Memutar senjata ke kiri sebesar sudut tertentu}
{I/S: menerima input angle, yaitu sudut rotasi senjata ke kiri dalam derajat}
{F/S: Senjata diputar ke kiri sebesar angle derajat}

procedure TurnRadarLeft(double angle)
{Memutar radar ke kiri sebesar sudut tertentu}
{I/S: menerima input angle, yaitu sudut rotasi radar ke kiri dalam derajat}
{F/S: Radar diputar ke kiri sebesar angle derajat}
```

```
procedure TurnRadarRight(double angle)
{Memutar radar ke kanan sebesar sudut tertentu}
{I/S: menerima input angle, yaitu sudut rotasi radar ke kanan dalam derajat}
{F/S: Radar diputar ke kanan sebesar angle derajat}

procedure SetForward(double distance)
{Mengatur bot agar bergerak maju sejauh tertentu}
{I/S: menerima input distance, yaitu jarak yang akan ditempuh ke depan}
{F/S: Bot mulai bergerak maju sejauh distance satuan}

procedure SetBack(double distance)
{Mengatur bot agar bergerak mundur sejauh tertentu}
{I/S: menerima input distance, yaitu jarak yang akan ditempuh ke belakang}
{F/S: Bot mulai bergerak mundur sejauh distance satuan}

procedure Fire(double power)
{Menembakkan peluru dengan kekuatan tertentu}
{I/S: menerima input power, yaitu kekuatan peluru dalam satuan energi
(biasanya antara 0.1 hingga 3)}
{F/S: Bot menembakkan peluru ke arah senjata dengan kekuatan power}

ALGORITMA
procedure Start()
{Memulai pertandingan dan menginisialisasi bot}
  instance baru dari Gevade
  Gevade.Start()

procedure Run()
{Mengatur alur utama pergerakan bot saat pertandingan berjalan}
  ResetEvent.run(onStart = true)
  while IsRunning do
    if not resetting then
      if clockwise then Forward(10000)
      else Forward(-10000)
    else Forward(0)

procedure OnScannedBot(event e)
{Menangani kejadian ketika radar mendeteksi keberadaan bot lawan}
  LockInEvent.set(e.X, e.Y)
  if not resetting then
    SmartFire(e.X, e.Y)

procedure OnHitBot(event e)
{Menangani kejadian ketika bot menabrak bot lain}
  if not resetting then
    LockInEvent.set(e.X, e.Y)
    clockwise ← not clockwise

procedure OnHitWall(event e)
{Menangani kejadian ketika bot menabrak dinding arena}
  if not resetting then
    clockwise ← not clockwise

procedure OnHitByBullet(event e)
{Menangani kejadian ketika bot terkena peluru dari musuh}
  if not resetting then
```

```
    clockwise ← not clockwise

procedure SmartFire(enemyX, enemyY)
{Menghitung kekuatan tembakan berdasarkan jarak musuh dan melakukan tembakan}
  distance ← DistanceTo(enemyX, enemyY)
  bulletPower ← 3 × exp(-0.01 × distance)
  Fire(bulletPower)

procedure LockInEvent.set(x, y)
{Mengunci target pada koordinat (x, y) dan menandai status terkunci}
  targetX ← x
  targetY ← y
  lockIn ← true
  turns ← 0

function LockInEvent.check()
{Mengecek apakah sudah terlalu lama mengunci target, dan memicu reset jika
perlu}
  if lockIn then
    if turns > 100 then
      lockIn ← false
      turns ← 0
      ResetEvent.run()
      → true
    else
      turns ← turns + 1
  → false

procedure ResetEvent.run(onStart = false)
{Melakukan proses pengaturan ulang posisi dan status bot}
  resetting ← true
  targetX ← arena width / 2
  targetY ← arena height / 2
  clockwise ← true

  if (onStart = true) then
    TurnGunLeft(90)
    TurnRadarLeft(360)
    desiredTurn ← Normalize(BearingTo(targetX, targetY) - 90)
    TurnLeft(desiredTurn)
    done()
  else
    SetTurnRadarLeft(360)
    desiredTurn ← Normalize(BearingTo(targetX, targetY) - 90)
    SetTurnLeft(desiredTurn)
    Tambahkan CustomEvent bernama "RadarDone"

procedure ResetEvent.done()
{Menandakan bahwa proses reset telah selesai}
  resetting ← false

function ResetEvent.isResetting()
{Mengecek apakah bot sedang dalam kondisi reset}
  → resetting

procedure OnTick()
```

```
{Dipanggil setiap tick atau siklus jalannya pertandingan untuk mengecek status
bot dan mengatur gerak lanjutan}
  if ResetEvent.isResetting() or LockInEvent.check() then →
  dx ← targetX - currentX
  dy ← targetY - currentY
  radius ← sqrt(dx² + dy²)
  radius ← max(1, radius - shrinkRate)
  TurnRate ← (180 × Speed) / (π × radius)

procedure OnCustomEvent(event)
{Menangani event kustom yang telah ditambahkan saat reset, misalnya RadarDone}
  if ResetEvent.isResetting() then
    if event.Name = "RadarDone" then
      bearing ← Normalize(GunDirection - RadarDirection)
      TurnRadarLeft(bearing)
      desiredTurn ← Normalize(BearingTo(targetX, targetY) - 90)
      TurnLeft(desiredTurn)
      ResetEvent.done()
    Remove(CustomEvent)
```

4.1.4 Greedy by Aggressive Circling

```
PROGRAM GaggressiveCircling
{Program ini menggunakan pendekatan algoritma greedy aggressive circling}
{seperti yang telah dijelaskan di atas.}
KAMUS
TargetX, TargetY, currentX, currentY, dx, dy, CircleX, CircleY: double
Bearing,  Radius, MaxSpeed, TurnRate, baseAngle, offsetAngle, finalAngle:
double
Forward: integer
clockwiseRadar: boolean

function GunBearingTo(double x, double y)
{Mengembalikan sudut relatif dari arah tembakan menuju koordinat x dan y}

function BearingTo(double x, double y)
{Mengembalikan sudut relatif antara arah bot dengan titik (x, y)}

function Normalize(double bearing)
{Mengembalikan sudut dalam rentang [-180°, 180°] dari sudut input}

function exp(double number)
{Fungsi eksponensial e^number untuk perhitungan kekuatan peluru}

procedure TurnRadarLeft(double bearing)
{Memutar radar ke kiri sebesar angle derajat}
{I/S: menerima input bearing, yaitu sudut untuk diputar}
{F/S: Radar diputar ke kiri sebesar sudut bearing}

procedure SetForward(double x)
{Mengatur bot agar bergerak maju ke arah hadapan sejauh x satuan}
```

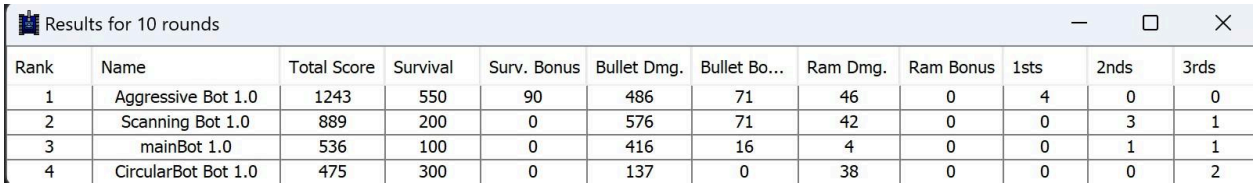
```
{I/S: menerima input x, yaitu jarak yang akan ditempuh ke depan}
{F/S: Bot mulai bergerak maju sejauh x satuan}

procedure SetTurnGunLeft(double bearing)
{Memutar senjata ke kiri sebesar bearing derajat}
{I/S: menerima input bearing, yaitu sudut rotasi ke kiri dari posisi senjata
saat ini}
{F/S: Senjata akan diputar ke kiri sebesar bearing derajat}

procedure Fire(double power)
{Menembakkan peluru dengan kekuatan tertentu}
{I/S: menerima input power, yaitu kekuatan peluru dalam satuan energi
(biasanya antara 0.1 hingga 3)}
{F/S: Bot menembakkan peluru ke arah senjata dengan kekuatan power}

ALGORITMA
procedure Start() {memulai sebuah match}
    Create a new instance of Gaggresive
    Gaggresive.Start()

procedure Run()
{gerakan utama dari algoritma ini}
    clockwiseRadar ← false
    while isRunning do
        TurnRadarLeft(360°)

procedure OnScannedBot(event e)
{setiap kali bot dipindai, lock target}
    LockTarget(e.X, e.Y)

procedure OnHitBot(event e)
{setiap kali bot menabrak bot lain, lock target}
    LockTarget(e.X, e.Y)

procedure OnHitWall(event e)
{setiap kali bot menabrak tembok, ganti arah pergerakkan robot}
    SetForward(-100)
    ChangeDirection()
    output("Ouch! I hit a wall, must turn back!")

procedure OnHitByBullet(event e)
{setiap kali bot terkena peluru, ganti arah pergerakkan robot}
    ChangeDirection()
    output("Ouch! a bullet hit me!")

procedure TurnRadar()
{mengganti arah pergerakkan radar}
    if clockwiseRadar then
        TurnRadarLeft(-360)
    else
        TurnRadarLeft(360)
    clockwiseRadar ← not clockwiseRadar

procedure LockTarget(TargetX, TargetY)
{Mengunci target pada koordinat tertentu, menembak jika memungkinkan, dan}
{mulai melakukan gerakan memutari target dengan arah ±45° terhadap posisi}
```

```
{musuh}
{I/S: menerima input TargetX dan TargetY, yaitu posisi musuh hasil deteksi}
{radar atau tabrakan}
{F/S: Bot akan mengarahkan senjata ke musuh, menembak jika sudut tembak}
{mengarah ke target, lalu mulai bergerak membentuk lintasan lingkaran di}
{sekitar target dengan offset sudut ±45°, menyesuaikan kecepatan dan arah}
{putaran tubuh}
    Set gun direction towards (TargetX, TargetY)
    if absolute(GunBearingTo(TargetX, TargetY)) < 7.5 then
        Fire with power = 3 × exp(-0.01 × DistanceTo(TargetX, TargetY))

    TurnRadar()

    dx ← currentX - TargetX
    dy ← currentY - TargetY
    baseAngle ← atan2(dy, dx)
    offsetAngle ← Forward × 45°
    finalAngle ← baseAngle + offsetAngle (in radians)

    CircleX ← TargetX + Radius × cos(finalAngle)
    CircleY ← TargetY + Radius × sin(finalAngle)

    SetForward(10,000)

    Bearing ← Normalize(BearingTo(CircleX, CircleY))
    if abs(Bearing) > 45 then
        MaxSpeed ← 2
    else
        MaxSpeed ← 1000
    TurnRate ← Bearing

procedure ChangeDirection()
{mengganti arah pergerakkan robot}
    Forward ← -Forward

function DistanceTo(TargetX, TargetY)
{mengembalikan jarak bot ke titik (x,y)}
    → sqrt((TargetX - currentX)² + (TargetY - currentY)²)
```

4.2 Pengujian Bot

Pengujian dilakukan dengan mengadu keempat bot dengan pendekatan algoritma greedy masing-masing. Pengujian dilakukan sebanyak 3 kali per 10 ronde. Hasil pengujian sebagai berikut.

Testing 1 :



Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Aggressive Bot 1.0	1243	550	90	486	71	46	0	4	0	0
2	Scanning Bot 1.0	889	200	0	576	71	42	0	0	3	1
3	mainBot 1.0	536	100	0	416	16	4	0	0	1	1
4	CircularBot Bot 1.0	475	300	0	137	0	38	0	0	0	2

Testing 2 :

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Aggressive Bot 1.0	1362	550	90	571	80	70	0	4	0	1
2	mainBot 1.0	837	200	0	560	64	13	0	1	2	1
3	CircularBot Bot 1.0	647	350	30	200	9	56	0	0	2	2
4	Scanning Bot 1.0	494	100	0	352	10	32	0	0	1	1

Testing 3 :

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	mainBot 1.0	1577	450	60	960	87	19	0	2	1	3
2	Scanning Bot 1.0	1562	450	0	928	139	44	0	3	1	2
3	Aggressive Bot 1.0	1356	550	90	555	42	119	0	1	3	1
4	CircularBot Bot 1.0	939	500	30	316	13	80	0	1	2	1

MainBot di gambar adalah nearest bot
*tapi ada revisi bot utama adalah Aggressive Bot 1.0

4.3 Analisis Hasil Pengujian

Strategi Greedy by Aggressive Circling berhasil mendapatkan nilai optimal ketika melawan bot yang lain,

Strategi Greedy by Nearest Target berhasil mendapatkan nilai optimal dibawah nearest bot.

Dari hasil Pengujian Bot, didapatkan data bahwa Aggressive bot berhasil meraih 2 kali kemenangan dari 3 pertandingan. Dengan kemampuan survival yang tinggi dan dapat memberikan damage yang cukup besar.

Algoritma ini tidak perlu menyimpan atau membandingkan banyak musuh, bisa bergerak secara terus-menerus mengorbit target, membuat posisi sulit diprediksi oleh musuh diam, dan bisa menyesuaikan kecepatan dan arah berdasarkan posisi relatif ke musuh.

Bab V
Kesimpulan dan Saran

5.1 Kesimpulan

Algoritma terbaik dari 4 Algoritma Greedy yang dibuat adalah Strategi Greedy by Aggressive Circling. Berdasarkan analisis hasil pengujian, bot dengan Algoritma adalah bot yang terbaik karena bot dapat meraih 2 kemenangan dalam 3 pertandingan melawan bot bot dengan algoritma lainnya.

Setiap kali radar menemukan musuh, radar diputar ke arah sebaliknya agar radar tetap mengarah ke musuh. Bot akan menembak jika sudut tembakan sudah tepat. Posisi target lingkaran dihitung dengan memanfaatkan sudut dasar antara bot dan musuh, ditambah offset sebesar ±45 derajat tergantung arah gerak. Hal ini membuat bot memutari musuh dalam pola spiral atau orbit yang agresif sambil menembak pelurunya. Sebagai tambahan, terdapat konfigurasi smart fire, yang mana *power* tembakan naik secara eksponensial tergantung jarak antara kedua bot. Selain itu, terdapat konfigurasi *max speed* agar robot dapat berputar dengan smooth jika sudut bot yang di-*scan* terlalu jauh.

Dengan algoritma ini, terdapat berbagai kelebihan seperti Tidak perlu menyimpan atau membandingkan banyak musuh, bergerak secara terus-menerus mengorbit target, membuat posisi sulit diprediksi oleh musuh diam, dan bisa menyesuaikan kecepatan dan arah berdasarkan posisi relatif ke musuh.

5.1 Saran

Saran untuk kedepannya adalah agar bot tidak menyangkut di tembok.

Lampiran

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.		✓

Link Github: TKurr/Tubes1_MasugiEnjoyer

Daftar Pustaka

https://fikti.umsu.ac.id/algoritma-greedy-pengertian-jenis-dan-contoh-program/
https://robowiki.net/wiki/Robocode_Documentation
https://robocode.sourceforge.io/docs/robocode/
https://robocode-dev.github.io/tank-royale/tutorial/beyond-the-basics.html#main-loop