

**IF2211 – Strategi Algoritma**  
**Laporan Tugas Besar 1: Algoritma Greedy**



Dipersiapkan oleh:

**Pemburu Tankjil**

Muhammad Aulia Azka (13523137)

Fachriza Ahmad Setiyono (13523162)

Filbert Engyo (13523163)

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**JL. GANESA 10, BANDUNG 40132**

**2025**

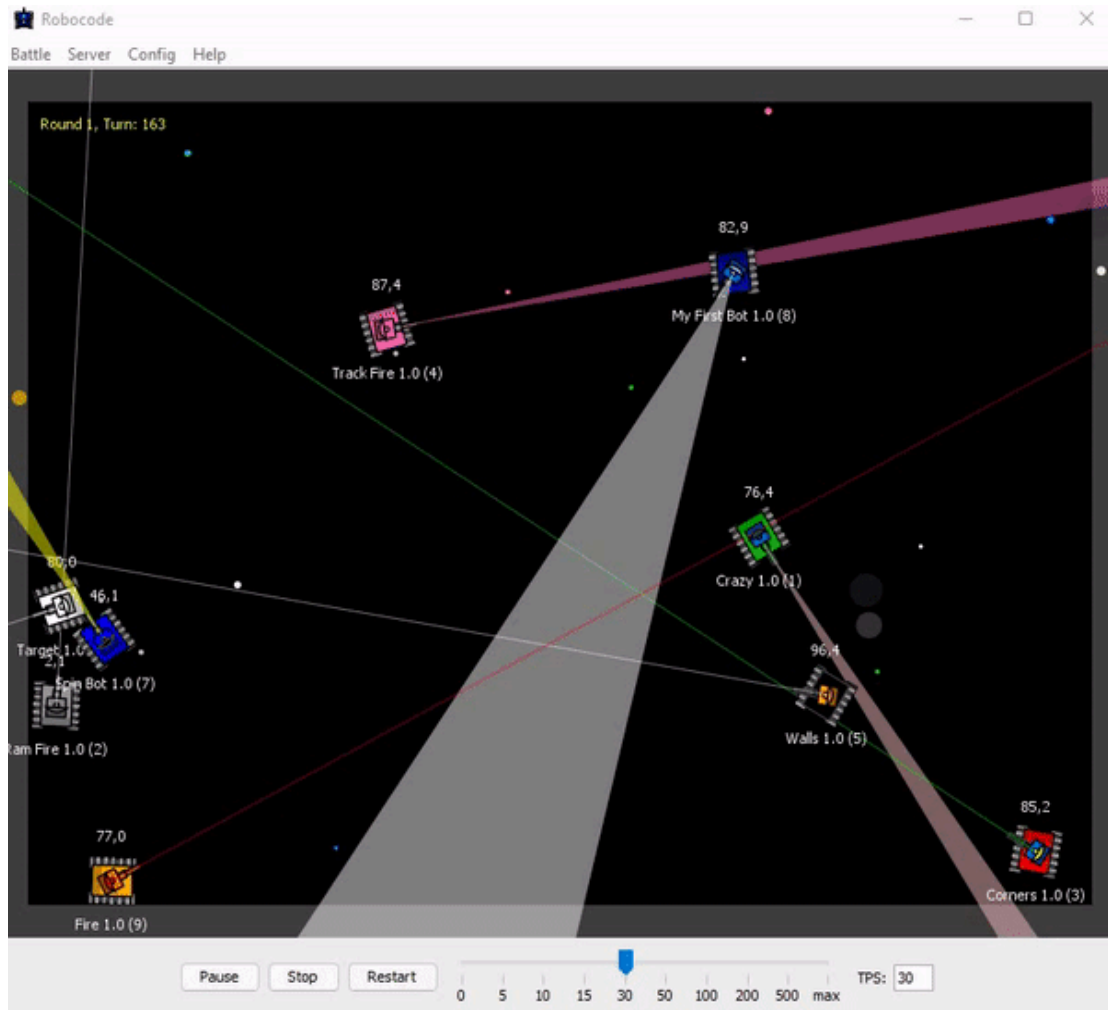
## Daftar Isi

<b>Daftar Isi</b>	<b>2</b>
<b>Daftar Gambar</b>	<b>3</b>
<b>1. Deskripsi Tugas</b>	<b>4</b>
<b>2. Landasan Teori</b>	<b>10</b>
2.1. Algoritma Greedy	10
2.2. Robocode	10
<b>3. Aplikasi Strategi Greedy</b>	<b>12</b>
3.1. Elemen Greedy pada Robocode	12
3.2. Alternatif Solusi Greedy	12
3.2.1. Bot Bhh	12
3.2.2. Bot Chh	13
3.2.3. Bot PemburuAlatreon	16
3.2.4. Bot DrinkAndDrive	17
3.3. Alternatif Solusi Pilihan	18
<b>4. Implementasi dan Pengujian</b>	<b>19</b>
4.1. Implementasi	19
4.1.1. Bot Bhh	19
4.1.2. Bot Chh	20
4.1.3. Bot PemburuAlatreon	25
4.1.4. Bot DrinkAndDrive	27
4.2. Pengujian dan Analisis	33
4.2.1. Bot Bhh	34
4.2.2. Bot Chh	34
4.2.3. Bot PemburuAlatreon	34
4.2.4. Bot DrinkAndDrive	35
<b>5. Kesimpulan dan Saran</b>	<b>36</b>
5.1. Kesimpulan	36
5.2. Saran	36
<b>Lampiran</b>	<b>37</b>
<b>Daftar Pustaka</b>	<b>37</b>

## Daftar Gambar

Gambar 1.1. Robocode Tank Royale	4
Gambar 1.7.1. Bagian Tank Robocode	6
Gambar 1.10.1. Full Scan Radar	8
Gambar 1.10.2. Still Radar 🦴 🦴	8
Gambar 4.2.1. Hasil pengujian ke-1	33
Gambar 4.2.2. Hasil pengujian ke-2	33
Gambar 4.2.3. Hasil pengujian ke-3	33

## 1. 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:

### 1.1. Rounds dan Turns

Pertempuran dapat terdiri dari beberapa rounds. Secara default, satu pertempuran berisi 10 rounds, di mana setiap rounds akan memiliki pemenang dan yang kalah.

Setiap round dibagi menjadi beberapa turns, yang merupakan unit waktu terkecil. Satu turn adalah satu ketukan waktu dan satu putaran permainan. Jumlah turn dalam satu round tergantung pada berapa lama waktu yang dibutuhkan hingga hanya tersisa bot terakhir yang bertahan.

Pada setiap turn, sebuah bot dapat:

- Menggerakkan bot, memindai musuh, dan menembakkan senjata.
- Bereaksi terhadap peristiwa seperti saat bot terkena peluru atau bertabrakan dengan bot lain atau dinding.
- Perintah untuk bergerak, berputar, memindai, menembak, dan sebagainya dikirim ke server untuk setiap turn.

Perlu diperhatikan bahwa [API \(Application Programming Interface\)](#) bot resmi secara otomatis mengirimkan niat bot ke server di balik layar, sehingga Anda tidak perlu mengkhawatirkannya, kecuali jika Anda membuat API Bot sendiri.

Pada setiap turn, bot akan secara otomatis menerima informasi terbaru tentang posisinya dan orientasinya di medan perang. Bot juga akan mendapatkan informasi tentang bot musuh ketika mereka terdeteksi oleh pemindai.

Perlu diketahui bahwa game engine yang akan digunakan pada tugas besar ini tidak mengikuti aturan default mengenai komponen Round & Turns.

### 1.2. Batas Waktu Giliran

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

Setiap kali turn baru dimulai, penghitung waktu ulang diatur ulang dan mulai berjalan. Jika batas waktu tercapai dan bot tidak mengirimkan pergerakannya untuk turn tersebut, maka tidak ada perintah yang dikirim ke server. Akibatnya, bot akan 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.

### 1.3. Energi

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

- Bot akan kehilangan energi jika ditembak atau ditabrak oleh bot musuh.
- Bot juga akan kehilangan energi jika menembakkan meriamnya.

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

#### 1.4. Peluru

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

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

#### 1.5. Panas Meriam (*Gun Heat*)

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

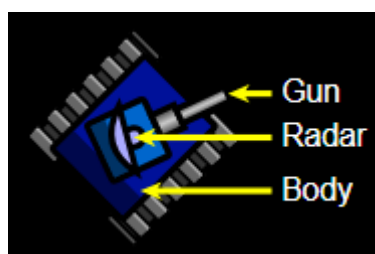
#### 1.6. Tabrakan

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

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

#### 1.7. Bagian Tubuh Tank

Tubuh tank terdiri dari 3 bagian:



Gambar 1.7.1. Bagian Tank Robocode

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

#### 1.8. Pergerakan

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

#### 1.9. Berbelok

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

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

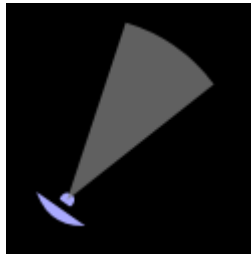
Bagian paling lambat adalah tubuh tank, yang dalam kondisi terbaik dapat berputar hingga 10 derajat per giliran. Namun, ini bergantung pada kecepatan bot saat ini. Semakin cepat bot bergerak, semakin lambat kemampuannya untuk berbelok.

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

#### 1.10. Pemindaian

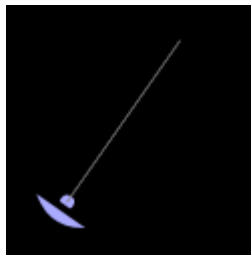
Aspek penting dalam Robocode adalah memindai bot musuh menggunakan radar. Radar dapat mendeteksi bot dalam jangkauan hingga 1200 piksel. Musuh yang berada lebih dari 1200 piksel dari bot tidak dapat terdeteksi atau dipindai oleh radar.

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



*Gambar 1.10.1. Full Scan Radar*

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.



*Gambar 1.10.2. Still Radar* 🏴‍☠️ 🏴‍☠️

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

### 1.11. Skor

Pada akhir pertempuran, setiap bot akan diberi *ranking* 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.

## 2. Landasan Teori

### 2.1. *Algoritma Greedy*

Algoritma *greedy* adalah salah satu jenis algoritma yang mengimplementasikan strategi heuristik untuk membuat pilihan optimal lokal dengan harapan mencapai pilihan optimal global di akhir. Karena karakteristik dari algoritma *greedy*, solusi yang dihasilkan belum tentu yang paling optimal, tetapi dapat menghasilkan solusi optimal lokal yang mendekati solusi optimal global dalam jangka waktu yang wajar.

Algoritma *greedy* membentuk solusi langkah per langkah. Pada setiap langkah, banyak pilihan yang bisa diambil. Namun, pilihan yang terbaik biasanya tidak bisa didapat dengan mudah. Oleh karena itu, algoritma *greedy* memilih solusi optimal lokal berdasarkan suatu heuristik yang dipilih. Namun, jika memang terbukti bahwa algoritma *greedy* tidak bisa menghasilkan solusi optimal global untuk suatu kasus, biasanya algoritma ini tetap digunakan karena lebih cepat atau efisien dibanding strategi algoritma lain.

Elemen-elemen yang terdapat pada suatu algoritma *greedy*:

- **Himpunan Kandidat (  $C$  ):** Kumpulan elemen yang memiliki potensi untuk membentuk solusi dari suatu masalah.
- **Himpunan Solusi (  $S$  ):** Kumpulan solusi yang dipilih dari himpunan kandidat  $C$  yang telah memenuhi beberapa kriteria tertentu yang menyatakan suatu solusi dan  $S$  dioptimasi oleh fungsi objektif.
- **Fungsi Solusi:** Fungsi untuk menentukan apakah pilihan himpunan solusi  $S$  sudah memberikan solusi.
- **Fungsi Seleksi:** Fungsi untuk menentukan pilihan solusi dari himpunan kandidat  $C$ .
- **Fungsi Kelayakan:** Fungsi untuk mengecek apakah solusi yang sedang dipilih bersamaan dengan himpunan solusi  $S$  masih memenuhi batasan-batasan masalah.
- **Fungsi Objektif:** Fungsi untuk menentukan kualitas solusi yang dihasilkan dari himpunan solusi  $S$ . Fungsi ini bertujuan untuk menentukan apakah solusi yang dihasilkan sudah optimal berdasarkan kriteria yang telah ditentukan.

### 2.2. *Robocode*

Robocode adalah sebuah permainan pemrograman dengan tujuan untuk mengembangkan robot (bot) tank perang untuk berperang dengan bot tank lainnya. Bot tank diprogram menggunakan bahasa Java atau C# dan berperang secara *real-time* di server.

Pemain harus membuat program untuk mengatur gerakan selama perang dan meraih kemenangan dengan cara menghindari dari tembakan dan menembak musuh menggunakan bantuan radar.

Sebuah bot diimplementasikan dalam kelas yang meng-*extend* kelas Bot. Gerakan bot diprogram didalam sebuah *loop* dalam fungsi run. Seluruh perintah yang ada didalam satu iterasi *loop* tersebut akan dijalankan tiap turn. Namun, jika perintah yang dijalankan melebihi batas waktu tiap turn-nya, maka bot akan tidak melakukan gerakan sama sekali di turn tersebut. Sehingga, bot harus diprogram seefisien mungkin untuk menghindari hal tersebut.

Untuk berinteraksi dengan medan perang, fungsi-fungsi *event* akan dipanggil untuk bot terkait. Misal *event* bot terdeteksi radar dan *event* bot menabrak bot lainnya. Fungsi tersebut menerima objek *event* terkait yang berisi informasi mengenai kejadian tersebut, sehingga bot bisa bereaksi sesuai keinginan pemain.

Tentunya banyak strategi algoritma yang bisa digunakan bot untuk memenangkan pertempuran. Dalam kasus ini, algoritma *greedy* akan digunakan. Untuk mengimplementasikan algoritma *greedy* ke dalam gerakan bot, maka perlu dilakukan analisis terhadap komponen-komponen dalam permainan robocode agar dapat dilakukan pengoptimalan solusi yang dipilih. Hal ini akan dibahas lebih lanjut di bab selanjutnya.

Setelah bot selesai diprogram, bot bisa dijalankan di dalam robocode dengan langkah-langkah berikut:

1. Masukkan *parent folder* direktori bot yang dibuat ke dalam konfigurasi direktori bot di dalam robocode GUI.
2. Mulai konfigurasi *battle*.
3. *Boot* bot yang ingin dimainkan, tunggu hingga bot berhasil di-*boot* (ditandai dengan bot masuk ke tabel kiri bawah).
4. Pilih bot yang sudah di-*boot* dan mulai *battle*.

Setelah *battle* dimulai, *battle* akan berlanjut hingga 10 ronde. Di akhir tiap ronde, akan ada pemenang ronde, tetapi hasil akhir yang diperhitungkan untuk robocode tank royale adalah dari hasil akumulasi poin. Seperti yang telah dijelaskan di bab sebelumnya, ada banyak cara untuk mendapatkan poin. Metode pengumpulan poin ini yang bisa dioptimalisasi menggunakan algoritma *greedy*.

### 3. Aplikasi Strategi Greedy

#### 3.1. Elemen Greedy pada Robocode

Untuk bisa mendesain dan merumuskan algoritma greedy yang efisien, maka elemen-elemen greedy pada robocode perlu diidentifikasi. Ada banyak bagian dari permainan robocode yang dapat dicari solusinya dengan metode algoritma *greedy*, seperti *greedy by energy* dan *greedy by distance* untuk mencari suatu target bot, Ada juga algoritma *greedy* untuk menjaga keberlangsungan hidup bot seperti *greedy by furthest distance* dimana bot akan pergi ke tempat sejauh mungkin dari kumpulan bot musuh lainnya.

#### 3.2. Alternatif Solusi Greedy

##### 3.2.1. Bot Bhh

Untuk alternatif pertama, bot dibuat untuk memaksimalkan poin **bullet damage** dan **ram damage** (beserta bonus poin yang terkait jika bot berhasil membunuh bot musuh). Alternatif ini muncul karena berdasarkan percobaan, bot yang hanya mengandalkan penembakan *bullet* untuk mengumpulkan poin seringkali gagal atau kurang efektif karena *bullet* jarang dan sulit mengenai bot musuh.

Bot akan terus memutar *radar* hingga bot musuh pertama kali ditemukan. Setelah menemukan musuh, bot akan melakukan *radar lock* dan akan terus menembak dan mengejar bot tersebut. Ketika bot menabrak musuh, bot akan memutar dan sedikit mundur sehingga bot bisa melakukan *ram* lagi di turn selanjutnya. Jika bot musuh tersebut mati, maka bot akan mengulangi proses dari awal, yaitu memutar *radar*. Proses ini diulang hingga ronde selesai atau bot mati.

Untuk memaksimalkan poin-poin yang telah disebutkan dari rancangan bot diatas, maka bot Bhh harus memilih bot musuh untuk dijadikan target. Maka, didefinisikan elemen-elemen *greedy* saat memilih bot target sebagai berikut:

- **Himpunan Kandidat ( C )**: Semua bot yang terdeteksi *radar*. Jika dalam satu turn sapuan radar mencakup lebih dari 1 bot, maka seluruh bot tersebut akan masuk ke dalam himpunan kandidat.
- **Himpunan Solusi ( S )**: Bot yang terpilih untuk dijadikan target *ram* dan tembak.
- **Fungsi Solusi**: Bot terpilih valid.
- **Fungsi Seleksi**: Bot pertama yang terdeteksi *radar*. Jika dalam satu turn sapuan radar mencakup lebih dari 1 bot, maka hasil *event scan* terakhir di dalam turn tersebut yang akan memilih bot.
- **Fungsi Kelayakan**: Semua bot yang terdeteksi *radar* valid.
- **Fungsi objektif**: Memaksimalkan poin dari **bullet damage** dan **ram damage**.

Bot ini diasumsikan bisa mendapat skor yang banyak dari **ram damage** dan **ram bonus**. Dengan terus menjepit bot musuh ke tembok, *ram* bisa terus dilakukan tiap turn-nya hingga bot musuh mati. Namun, saat bot mencari bot musuh untuk di-*ram*, ada kemungkinan bot juga mati tertembak oleh musuh, sehingga bot ini kurang efisien jika tidak berhasil menangkap bot musuh.

### 3.2.2. *Bot Chh*

Untuk alternatif kedua, bot dirancang untuk memaksimalkan kedua **survival score** dan **bullet damage**. Hal ini tentunya sulit karena untuk menambahkan poin dari *bullet damage*, diperlukan energi atau “nyawa” bot untuk menembak. Selain itu, untuk memaksimalkan *survival score*, bot harus bisa menghindari dari berbagai bahaya yang ada di medan perang, yaitu *bullet* dari bot lain, tabrakan dengan bot lain, dan tabrakan dengan tembok.

Untuk memudahkan pengimplementasian algoritma, maka bot dibuat dengan sistem *state machine*. Sistem ini memungkinkan untuk memisahkan secara logis aksi-aksi yang dilakukan bot berdasarkan *state* atau keadaan bot pada saat itu. Bot ini memiliki 3 *state*, yaitu **EVADE**, **BATTLE**, dan **RUN**, setiap *state* memiliki heuristiknya masing-masing dengan tujuan yang berbeda.

#### 3.2.2.1. EVADE

Di *state* ini, bot akan berusaha untuk memaksimalkan *survival score* dengan menghindari dari *bullet* musuh serta tembok. Walau begitu, bot akan tetap menembakkan *bullet* untuk mendapat beberapa poin.

Ada dua algoritma *greedy* yang digunakan di *state* ini. Algoritma *greedy* pertama adalah algoritma untuk menghindari dari *bullet*, dan yang kedua adalah untuk menentukan target tembakan. Untuk membantu penentuan solusi, bot akan terus memutar *radar* sehingga informasi terkait medan perang dapat diketahui. Semua data penting terkait bot disimpan dan diperbarui.

Untuk algoritma menghindari, data bot tersimpan akan dibaca dan akan dipilih yang terdekat dan yang baru saja menghabiskan energi. Tidak ada cara konkret untuk menentukan apakah bot musuh baru saja menembak, tetapi kita bisa menyimpulkan bahwa jika bot tersebut kekurangan energi, maka ada kemungkinan bot tersebut menembak. Untuk menghindari dari kemungkinan tembakan *bullet*, maka bot akan belok 90° menjauhi dari titik bot saat ini. Elemen *greedy*-nya adalah:

- **Himpunan Kandidat (  $C$  )**: Semua data bot yang ada tersimpan di memori bot. Data bot mencakup jumlah energi di pembacaan terbaru dan jumlah energi di pembacaan sebelumnya.

- **Himpunan Solusi ( S )**: Bot dengan jarak terdekat dan energi yang menurun dari pembacaan sebelumnya (diasumsikan telah menembak ke arah bot).
- **Fungsi Solusi**: Bot terpilih valid.
- **Fungsi Seleksi**: Bot pertama yang terdeteksi *radar*. Jika dalam satu turn sapuan radar mencakup lebih dari 1 bot, maka hasil *event* scan terakhir di dalam turn tersebut yang akan memilih bot.
- **Fungsi Kelayakan**: Semua data bot di memori dianggap layak, karena data dari bot yang telah mati akan dihapus dari memori.
- **Fungsi objektif**: Memaksimalkan poin dari **survivor score** dengan cara menghindari.

Sedangkan untuk algoritma menembak, bot akan memilih dari probabilitas kemungkinan *bullet* mengenai musuh. Probabilitas *bullet* mengenai bot  $x$  dihitung dengan cara:

$$P(x) = \frac{\frac{x.hitCount}{x.shotCount} \times firepower}{distance^2}, \quad x.shotCount > 0$$

Dimana *firepower* adalah kekuatan tembakan yang dikalkulasi berdasarkan jarak bot  $x$ . Perlu diperhatikan bahwa *shotCount* haruslah lebih besar dari 0 untuk mencegah pembagian terhadap 0. Jika untuk kasus bot  $x$  belum pernah ditembak sebelumnya (yang berarti *shotCount* bernilai 0), maka diasumsikan  $P(x) = 1$ .

Terdapat kuadrat dari jarak sebagai pembagi karena hukum *inverse square law* menyatakan bahwa suatu kuantitas akan berkurang sebanding dengan kuadrat dari jarak. Dapat dipastikan bahwa jarak tidak mungkin 0 karena pada dasarnya, bot tidak bisa berada di tempat yang sama dengan bot lainnya. Dengan begitu, dapat didefinisikan elemen-elemen *greedy* dalam menentukan target bot sebagai berikut:

- **Himpunan Kandidat ( C )**: Semua data bot yang ada tersimpan di memori bot. Data bot mencakup jumlah tembakan terarah pada bot tersebut dan jumlah tembakan yang berhasil mengenai bot tersebut.
- **Himpunan Solusi ( S )**: Bot di dalam memori dengan probabilitas *hit* tertinggi.
- **Fungsi Solusi**: Bot memiliki probabilitas *hit* tertinggi.

- **Fungsi Seleksi:** Menghitung probabilitas *hit* dari himpunan kandidat bot di dalam memori menggunakan rumus yang telah terdefinisi.
- **Fungsi Kelayakan:** Semua data bot di memori dianggap layak, karena data dari bot yang telah mati akan dihapus dari memori.
- **Fungsi objektif:** Memaksimalkan poin dari **bullet damage** dengan mencari target bot dengan probabilitas *hit* tertinggi, atau dalam bentuk matematis:

$$F = \max(P(x))$$

#### 3.2.2.2. BATTLE

Di *state* ini, bot akan berusaha untuk memaksimalkan **bullet damage** dengan mengikuti bot target dengan mengitarinya dan menembaknya. Bot target adalah bot dari hasil sapuan *radar* yang jaraknya kurang dari suatu ambang batas. Selama masih di satu sesi *state BATTLE*, bot tidak akan mengganti targetnya walaupun ada bot lain yang masuk ke sapuan *radar*. Bot akan keluar dari *state* ini jika bot target menjauh dari ambang batas atau bot tertabrak oleh bot lain.

Karena target bot adalah yang terdekat dibawah ambang batas tertentu, maka dapat didefinisikan elemen-elemen *greedy* sebagai berikut:

- **Himpunan Kandidat ( C ):** Semua bot yang terdeteksi di *radar*.
- **Himpunan Solusi ( S ):** Bot terdeteksi *radar* yang lebih dekat dari suatu ambang batas.
- **Fungsi Solusi:** Bot memiliki jarak lebih dekat dari suatu ambang batas.
- **Fungsi Seleksi:** Menghitung jarak bot di dalam sapuan *radar* untuk menentukan apakah bot melewati batas yang telah ditentukan.
- **Fungsi Kelayakan:** Semua data bot hasil sapuan *radar* dianggap valid.
- **Fungsi objektif:** Memaksimalkan poin dari **bullet damage** dengan mencari target bot dengan jarak terdekat dan menembaknya.

#### 3.2.2.3. RUN

Bot akan masuk ke dalam *state* ini jika bot tertabrak (*rammed*) oleh bot musuh. Di *state* ini, bot akan berusaha untuk keluar dari

situasi bahaya dengan berlari ke arah tengah medan perang. Titik tengah medan perang dipilih karena jika bot berlari ke arah tembok, maka ada kemungkinan bot akan terjebak dan tidak bisa kabur. Jadi dapat dikatakan bahwa pada *state* ini, bot akan memaksimalkan **survival score** dengan cara berlari menjauhi bahaya

Berdasarkan rancangan algoritma tersebut, bot ini diasumsikan mampu mendapatkan skor lebih tinggi dari bot lainnya karena bot ini memiliki tingkat keberlangsungan hidup lebih tinggi dari bot lainnya. Dengan bertahan hidup lebih lama, bot memiliki kesempatan untuk mendapatkan skor lebih banyak, ditambah juga *survival score* dan *survival bonus*.

### 3.2.3. *Bot Pemburu Alatreon*

Untuk alternatif ketiga, bot menggunakan pendekatan greedy dengan heuristik mengincar **target terdekat** dan memaksimalkan **fire power** untuk jarak paling dekat. Alternatif ini muncul karena dengan mengincar bot terdekat, seharusnya bot tersebut dapat lebih mudah mengenai bot musuh.

Bot pertama kali akan memutar radar agar mendapatkan informasi jarak antara bot alatreon dengan bot musuh. Lalu dari informasi tersebut akan didapatkan bot dengan jarak terdekat. Bot tersebut akan di-*lock* lalu ditembak dengan *fire power* sesuai jarak bot musuh dengan bot alatreon. Jika terdapat bot musuh kedua yang lebih dekat dengan bot alatreon dibandingkan bot yang sedang ditarget dan bot kedua tersebut tertangkap radar bot alatreon, maka bot alatreon akan mengganti targetnya ke bot kedua.

- **Himpunan Kandidat (  $C$  )**: Semua Bot musuh yang terdeteksi dengan radar (untuk heuristik target terdekat) dan nilai - nilai tembakan yang dipilih (Untuk heuristik fire power nilai - nilai tersebut adalah : 1, 2, 3)
- **Himpunan Solusi (  $S$  )**: Bot yang memiliki jarak terdekat (untuk heuristik target terdekat) dan nilai tembakan sesuai kondisi jarak bot target dengan bot alatreon (untuk heuristik fire power).
- **Fungsi Solusi**: Menentukan dari semua jarak antara bot musuh dengan bot alatreon, yang paling dekat akan dipilih sebagai target (untuk heuristik jarak terdekat). Mengambil informasi jarak target ke bot alatreon. Lalu tentukan nilai tembakan yang digunakan berdasarkan jarak target ke bot alatreon (untuk heuristik fire power).
- **Fungsi Seleksi**: Pilih bot untuk dijadikan target berdasarkan jarak yang paling dekat (untuk heuristik jarak terdekat). Pilih nilai tembakan berdasarkan jarak target ke bot alatreon (untuk heuristik fire power).
- **Fungsi Kelayakan**: Target dianggap layak jika datanya masih “*fresh*”, yakni musuh telah *discan* dalam periode tertentu. Untuk bot alatreon sendiri, target dianggap tidak layak jika target tersebut tidak *terscan* lebih dari 30 *turn*. Lalu sebelum menembak bot alatreon akan



mengecek apakah energi yang digunakan itu cukup untuk menembak (jika tidak dilakukan pengecekan maka bot dapat mati karena menembak).

- **Fungsi Objektif:** Memaksimalkan damage terhadap musuh yang paling dekat.

Untuk algoritma greedy bot Pemburu Alatreon, heuristik yang digunakan adalah target terdekat dan fire power maksimum untuk target jarak terdekat. Hal ini cukup efisien karena untuk implementasinya sendiri hanya menggunakan perhitungan sederhana seperti menghitung jarak terdekat, memeriksa jarak, dan memilih nilai tembakan berdasarkan *threshold* jarak. Karena hal tersebut, algoritma ini dapat mengambil keputusan secara instan untuk setiap turn-nya.

Untuk efektivitasnya sendiri, algoritma greedy dengan heuristik yang digunakan bot Pemburu Alatreon cukup efektif. Hal ini disebabkan karena heuristik yang digunakan bertujuan untuk memaksimalkan damage untuk bot terdekat. Hal ini efektif dikarenakan target yang paling dekat cenderung memberikan peluang yang besar untuk tembakan dari bot Pemburu Alatreon mengenai bot tersebut yang jaraknya paling dekat.

#### 3.2.4. *Bot DrinkAndDrive*

Untuk alternatif keempat, bot dibuat dengan motif mengincar musuh dengan **energy** atau **nyawa terendah** yang tergantung pada jaraknya juga akan memaksimalkan serangan tembakan. Jangkauan pemindaian bot ini memiliki batas karena asumsinya jika apabila bot dengan *hp* terendah memiliki jarak yang cukup jauh, maka kurang optimal untuk mendekati baru melakukan penyerangan. Motif dari bot ini adalah bisa fokus mengeliminasi musuh dengan *hp* terendah terlebih dahulu yang kemungkinan dibantu dengan bot lain agar mempercepat pertandingan.

Awalnya bot akan melakukan pemindaian menggunakan radar ke seluruh penjuru dengan batas jarak tertentu sebelum mengunci target musuh yang akan diincar jika musuh berhasil keluar dari jangkauan pemindaian atau berhasil dieliminasi, maka proses pemindaian akan dilakukan kembali.

Secara garis besar, mekanisme bot ini terbagi dalam 3 *state* atau kondisi yaitu MOVEMENT, BATTLE, dan ESCAPE untuk mempermudah kondisi, tetapi patokan *greedy* yang digunakan tetap sama yaitu *greedy by lowest hp*.

- **Himpunan Kandidat ( C ):** Semua bot yang terdeteksi dalam batasan jangkauan radar.
- **Himpunan Solusi ( S ):** Bot yang memiliki *energy* / *hp* terendah diantara yang lain.

- **Fungsi Solusi:** Menentukan bot yang sesuai dengan kriteria kemudian dikunci hingga berhasil dieliminasi atau apabila target berhasil keluar dari jangkauan terkunci.
- **Fungsi Seleksi:** Memungkinkan bot untuk mencari musuh yang memenuhi kriteria berdasarkan lokasi dan id yang tersimpan sebagai data.
- **Fungsi Kelayakan:** Semua data bot yang berhasil dipindai dan disimpan selalu dianggap valid dan dapat dipertimbangkan.
- **Fungsi Objektif:** Memfokuskan pada mengeliminasi bot musuh dengan *energy* / *hp* terendah.

Berdasarkan motif algoritma yang diterapkan, bot ini diasumsikan bisa mendapat poin terbanyak karena bot mengincar musuh dengan hp terendah sehingga bisa mendapat banyak poin bonus (*bullet damage bonus*) dari eliminasi / *last hit*.

### 3.3. *Alternatif Solusi Pilihan*

Setelah mempertimbangkan hasil analisis teoritis dari semua alternatif solusi, ditentukan bahwa bot Chh dipilih menjadi solusi pilihan.

## 4. Implementasi dan Pengujian

### 4.1. Implementasi

Untuk bisa mendesain dan merumuskan algoritma greedy yang efisien, maka elemen-elemen greedy pada robocode perlu diidentifikasi terlebih dahulu.

#### 4.1.1. Bot Bhh

Berikut adalah *pseudocode* dari loop utama bot Bhh:

```
while (IsRunning) do

    if (not foundBot) then

        WaitFor(new RadarTurnCompleteCondition(this))
        SetTurnRadarRight(360 * TurnDir)

    else

        foundBot ← false
        Rescan()

Go()
```

Komponen *greedy* bot Bhh ada di fungsi OnScannedBot, yang terpanggil jika *radar* mendeteksi bot musuh. Berikut adalah isi fungsinya:

```
OnScannedBot(evt: ScannedBotEvent)

bearing ← BearingTo(evt.X, evt.Y)
gunBearing ← GunBearingTo(evt.X, evt.Y)
distance ← DistanceTo(evt.X, evt.Y)
speed ← Math.Abs(bearing) < 20 ? 50 : 20

SetForward(speed)
SetTurnLeft(bearing)
SetTurnGunLeft(gunBearing)
SetTurnRadarRight(0)
SetTurnRadarLeft(RadarTurnRemaining * TurnDir)

TurnDir ← TurnDir * -1

if (gunBearing < 10) then

    SmartFire(distance)
    Go()

    foundBot ← true
```

Juga terdapat fungsi untuk menentukan *powerfire* yang sesuai dengan jarak ke bot musuh:

SmartFire(distance: double)
<pre> powerFire ← 0  <b>if</b> (distance &lt; 200) <b>then</b>     powerFire ← 3 <b>else if</b> (distance &lt; 400) <b>then</b>     powerFire ← 2 <b>else</b>     powerFire ← 1  { Pastikan tembakan tidak membahayakan bot sendiri } <b>if</b> (Energy &gt; powerFire + 1) <u><b>then</b></u>     SetFire(powerFire) </pre>

#### 4.1.2. Bot Chh

Bot ini memerlukan beberapa hal khusus dalam implementasinya, seperti struktur data untuk menyimpan data bot dan sistem *state machine*.

class Chh : Bot
<pre> <b>enum</b> BotState:     EVADE,     BATTLE,     HIT  botState ← BotState.EVADE { <i>state</i> awal bot }  rand ← Random() { Random module }  <b>class</b> BotData:     int ID;     int hitCount;     int shotCount;     double lastEnergy;     double currentEnergy;     double X;     double Y;  scannedBots ← Dictionary&lt;int, BotData&gt;() targetId = -1 { ID bot target saat mode BATTLE } stuckCooldown = 0 { Cooldown untuk mencegah wall / corner stuck } hitCooldown = 0 </pre>

```
isMovingForward ← true  
  
const NEAR_WALL_OFFSET ← 60
```

Di dalam loop utama bot, keputusan diambil berdasarkan *state* bot saat ini:

```
while IsRunning do  
  
    { Kurangi cooldown tiap turn }  
    if (stuckCooldown > 0) then  
        stuckCooldown ← stuckCooldown - 1  
    if (hitCooldown > 0) then  
        hitCooldown ← hitCooldown - 1  
  
    isNearWall ← (X < NEAR_WALL_OFFSET or X > ArenaWidth -  
    NEAR_WALL_OFFSET or Y < NEAR_WALL_OFFSET or Y >  
    ArenaHeight - NEAR_WALL_OFFSET);  
  
    if EnemyCount = 1 and TurnNumber mod 30 ≤ 1 then  
        ReverseDirection()  
        SetForward(0)  
        SetBack(0)  
  
    SetTurnRadarRight(45)  
    depend on (botState)  
  
    BotState.BATTLE:  
  
        if isNearWall then  
            if stuckCooldown > 0 then  
                Move(centerX, centerY)  
            else  
                ReverseDirection(rand.Next(12, 24))  
  
        Rescan()  
  
        if scannedBot.TryGetValue(targetId, out BotData botData) then  
            SmartFire(botData)  
  
        if isMovingForward then  
            SetBack(0)  
            SetForward(50)  
        else  
            SetForward(0)  
            SetBack(50)  
  
    BotState.EVADE:
```

```

        Move(centerX, centerY)

        if not isNearWall then
            GreedyEvade()

        BotState.HIT:

            Move(centerX, centerY)

        }
        Go()
    }

```

Salah satu fungsi yang penting adalah di fungsi OnScannedBot, dimana bot bisa memperbarui data-data bot di medan perang melalui hasil sapuan *radar*:

```

if hitCooldown == 0 then
    dist ← DistanceTo(evt.X, evt.Y)

    if dist < 200 or EnemyCount == 1 then
        { Masuki state BATTLE }
        botState ← BotState.BATTLE

        bearing ← BearingTo(evt.X, evt.Y)
        gunBearing ← GunBearingTo(evt.X, evt.Y)

        { Arahkan tank dan tembakan ke musuh }
        SetTurnLeft(bearing + 90)
        SetTurnGunLeft(gunBearing)

        RotateRadar(evt.X, evt.Y)
    else
        SetTurnRight(0)
        botState ← BotState.EVADE

    { Update bot data atau buat entry bot data }
    if scannedBots.TryGetValue(evt.ScannedBotId, out BotData bd) then
        bd.X = evt.X
        bd.Y = evt.Y
        bd.lastEnergy = bd.currentEnergy
        bd.currentEnergy = evt.Energy
    else
        newBd ← BotData(evt.ScannedBotId, -1, evt.Energy, evt.X, evt.Y, 0,
0)
        scannedBots[evt.ScannedBotId] ← newBd

```

```
targetId ← evt.ScannedBotId;
```

Sekarang kita masuk ke fungsi GreedyEvade, dimana bot akan berusaha menghindari dari tembakan musuh:

```
minDistance ← Double.PositiveInfinity
maxProbability ← -1.0
gunAngle ← 0.0
nearestBot ← null
probableTarget ← null

firePower ← 0
foreach KeyValuePair<int, BotData> item in scannedBots do
    bd ← item.Value;

    dist ← DistanceTo(bd.X, bd.Y)

    { Deteksi penurunan energy → bot menembak }
    if bd.lastEnergy ≠ -1 and bd.currentEnergy > 3 and bd.currentEnergy
    < bd.lastEnergy then

        { Simpan data bot terdekat }
        if dist < minDistance then
            minDistance ← dist
            nearestBot ← bd

    _firePower ← CalculatePowerFire(bd, out hitProbability);

    if hitProbability > maxProbability then
        firePower ← _firePower
        maxProbability ← hitProbability
        probableTarget ← bd
        gunAngle ← GunBearingTo(bd.X, bd.Y)

{ terminasi foreach }

if probableTarget ≠ null and Math.Abs(gunAngle) < 12 then
    SmartFire(probableTarget, firePower)

Dodge(nearestBot)

SetTurnGunLeft(gunAngle)
```

Terlihat di fungsi tersebut, pemanggilan terhadap fungsi Dodge. Fungsi Dodge bertujuan untuk menggerakkan bot agar menjauh dari titik bot saat ini:

Dodge(nearestBot: BotData)
<u>if</u> nearestBot $\neq$ <u>null</u> <u>then</u> bearingTo $\leftarrow$ BearingTo(nearestBot.X, nearestBot.Y) enemyBearing $\leftarrow$ Direction + bearingTo  rnd $\leftarrow$ Math.Sign(rand.NextDouble()) SetTurnLeft(NormalizeRelativeAngle(enemyBearing + 90 * rnd - Direction))  SetForward(rand.NextDouble() > 0.25 ? 100 : -100)

Terlihat juga fungsi SmartFire untuk menginisiasi tembakan:

SmartFire(bd: BotData, firePower: <u>integer</u> $\leftarrow$ -1.0)
<u>if</u> firePower < 0 <u>then</u> firePower $\leftarrow$ CalculatePowerFire(bd)  <u>if</u> Energy > firePower + 1 <u>then</u> SetFire(firePower)

Berikut adalah fungsi CalculatePowerFire, dimana probabilitas hit suatu bot x juga dihitung:

CalculatePowerFire(bd: BotData, <u>out</u> hitProbability : <u>double</u> ) $\rightarrow$ <u>double</u>
distance $\leftarrow$ DistanceTo(bd.X, bd.Y) <u>if</u> bd.shotCount > 0 <u>then</u> hitProbability $\leftarrow$ bd.hitCount / bd.shotCount <u>else</u> hitProbability $\leftarrow$ 1  distanceFactor $\leftarrow$ (1.0 - hitProbability) * 200  powerFire $\leftarrow$ 0 <u>if</u> distance < 200 - distanceFactor <u>then</u> powerFire $\leftarrow$ 3 <u>else if</u> distance < 400 - distanceFactor <u>then</u> powerFire $\leftarrow$ 2 <u>else if</u> ((distance < 800 <u>and</u> Energy > bd.currentEnergy + 1) <u>or</u> rand.NextDouble() < hitProbability) <u>then</u> powerFire $\leftarrow$ 1  hitProbability $\leftarrow$ (hitProbability * powerFire) / (distance * distance)



```
powerFire →
```

#### 4.1.3. Bot PemburuAlatreon

Berikut adalah *pseudocode* dari loop run bot Pemburu Alatreon:

```
public override void Run()
while (isRunning) do
    var target ← closestEnemy()

    if (target ≠ null) then
        double enemyAngle ← DirectionTo(target.x, target.y)
        double distToEnemy ← DistanceTo(target.x, target.y)

        if (distToEnemy < 60) then
            double dodgeAngle ← normalizeAbsoluteAngle(enemyAngle +
(randomMovement.Next(90, 181) * movementDirection))
            double turnAngle ← normalizeRelativeAngle(dodgeAngle -
Direction)

            if turnAngle > 0 then
                SetTurnLeft(turnAngle)
            else then
                SetTurnRight(-turnAngle)

            SetBack(200)
        else
            double forwardDistance ← 50

            if (target.distance < 150) then
                forwardDistance ← -150
            else if (target.distance < 400) then
                forwardDistance ← 150
            else
                forwardDistance ← 50

            double headingTo ← normalizeAbsoluteAngle(enemyAngle +
(90 * movementDirection))
            double turnAngle ← normalizeRelativeAngle(headingTo -
Direction)

            if turnAngle > 0 then
                SetTurnLeft(turnAngle)
            else
                SetTurnRight(-turnAngle)
```

```

    if forwardDistance > 0 then
        SetForward(forwardDistance)
    else
        SetBack(-forwardDistance)
    else
        int randomTurn ← randomMovement.Next(-30, 31)
        if randomTurn > 0 then
            SetTurnLeft(randomTurn)
        else
            SetTurnRight(-randomTurn)

    if TurnNumber - lastRadarScanTurn > 2 then
        SetTurnRadarLeft(360)
    else
        SetTurnRadarLeft(360)

    Go()

```

Berikut adalah *pseudocode* dari method onScannedBot:

```

public override void onScannedBot(ScannedBotEvent e)

    lastRadarScanTurn ← TurnNumber
    UpdateEnemyInfo(e)

    var targetBot ← closestEnemy()

    if targetBot = null then
        →

    aimRadarandGun(targetBot)

    if lastEnergyMap.TryGetValue(e.ScannedBotId, out double oldEnergy)
    then
        if oldEnergy - e.Energy > 0.1 then
            double dodgeAngle ← randomMovement.Next(-60, 61)

            if dodgeAngle > 0 then
                SetTurnLeft(dodgeAngle)
            else
                SetTurnRight(-dodgeAngle)
            SetForward(80)

    lastEnergyMap[e.ScannedBotId] ← e.Energy

    double gunbearing ← GunBearingTo(targetBot.x, targetBot.y)

```

```
if Math.Abs(gunbearing) < 5 then  
    distanceFireGun(targetBot.distance)
```

Berikut adalah *pseudocode* dari method `closestEnemy` untuk menentukan bot terdekat:

```
private EnemyInfo closestEnemy()  
  
if enemies.Count = 0 then  
    → null  
  
double maxTurn ← 30  
  
var validEnemy ← enemies.Values.Where(enemy => TurnNumber -  
    enemy.lastScan  
    <= maxTurn).ToList()  
  
if validEnemy.Count = 0 then  
    return null  
  
var Closest ← validEnemy.OrderBy(enemy =>  
    enemy.distance).FirstOrDefault()  
    → Closest
```

Berikut adalah *pseudocode* dari method `distanceFireGun`

```
private void distanceFireGun(double distance)  
  
double powerFire  
  
if distance < 200 then  
    powerFire ← 3  
else if distance < 400 then  
    powerFire ← 2  
else then  
    powerFire ← 1  
  
if Energy > powerFire + 1 then  
    SetFire(powerFire)
```

#### 4.1.4. Bot DrinkAndDrive

Bot ini juga memerlukan beberapa hal khusus dalam implementasinya, seperti struktur data untuk menyimpan data bot dan sistem *state machine*:

```
public class DrinkAndDrive : Bot
```

```
public enum BotState:  
    MOVEMENT,  
    BATTLE,  
    ESCAPE
```

```
BotState botState = BotState.MOVEMENT;
```

```
Random rand ← new Random();  
private Dictionary<int, BotData> scannedBots ← new();  
private int targetId ← -1;  
private int stuckCooldown ← 0;  
private int hitCooldown ← 0;  
private bool isMovingForward ← true;
```

```
private class BotData:  
    public int ID;  
    public int hitCount;  
    public int shotCount;  
    public double lastEnergy;  
    public double currentEnergy;  
    public double X;  
    public double Y;
```

```
public BotData()
```

```
public BotData(int: ID, double: lastEnergy, double: currentEnergy,  
double: X, double: Y, int: hitCount, int: shotCount):  
    this.ID ← ID;  
    this.lastEnergy ← lastEnergy;  
    this.currentEnergy ← currentEnergy;  
    this.X ← X;  
    this.Y ← Y;  
    this.hitCount ← hitCount;  
    this.shotCount ← shotCount;
```

```
private int centerX;  
private int centerY;
```

```
const int NEAR_WALL_OFFSET ← 60;
```

Berikut adalah *pseudocode* loop run dari DrinkAndDrive:

```
public override procedure Run():
```

```
    AdjustRadarForBodyTurn ← false;  
    AdjustGunForBodyTurn ← false;  
    AdjustRadarForGunTurn ← false;
```

```

centerX ← ArenaWidth / 2;
centerY ← ArenaHeight / 2;

BodyColor ← Color.FromArgb(20, 61, 96);
TurretColor ← Color.FromArgb(235, 91, 0);
RadarColor ← Color.FromArgb(38, 31, 79);
BulletColor ← Color.FromArgb(235, 91, 0);
ScanColor ← Color.FromArgb(229, 32, 32);
TracksColor ← Color.FromArgb(25, 91, 0);
GunColor ← Color.FromArgb(20, 61, 96);

while (IsRunning) do
    if (stuckCooldown > 0) then stuckCooldown--;
    if (hitCooldown > 0) then hitCooldown--;

    bool isNearWall ← (X < NEAR_WALL_OFFSET or X >
ArenaWidth - NEAR_WALL_OFFSET or Y <
NEAR_WALL_OFFSET or Y > ArenaHeight -
NEAR_WALL_OFFSET);

    SetTurnRadarRight(45);
    switch (botState):
        case BotState.BATTLE:
            if (isNearWall) then
                if (stuckCooldown > 8) then Move(centerX, centerY);
                else RunAway(rand.Next(6, 12));
            else stuckCooldown ← 0;
            Rescan();
            if (scannedBots.TryGetValue(targetId, out BotData
botData)) then SmartFire(botData);
            if (isMovingForward) then
                SetBack(0);
                SetForward(50);
            else
                SetForward(0);
                SetBack(50);
            break;

        case BotState.MOVEMENT:
            Move(centerX, centerY);
            if (!isNearWall) then LockTarget();
            else stuckCooldown ← 0;
            break;

        case BotState.ESCAPE:
            SetTurnRight(45);
            SetForward(0);
            SetBack(300);

```

```

        break;

    default:
        break;
Go();

```

berikut adalah *pseudocode* dari prosedur LockTarget:

```

private procedure LockTarget():

    double minEnergy ← Double.PositiveInfinity;
    BotData lowestHPBot ← null;
    foreach (KeyValuePair<int, BotData> item in scannedBots) then
        BotData bd = item.Value;
        if (bd.currentEnergy < minEnergy) then
            minEnergy ← bd.currentEnergy;
            lowestHPBot ← bd;

    if (lowestHPBot ≠ null) then
        targetId = lowestHPBot.ID;
        Dodge(lowestHPBot);

```

Berikut adalah *pseudocode* dari prosedur Dodge:

```

private procedure Dodge(lowestHPBot: BotData):

    if (lowestHPBot ≠ null and lowestHPBot.lastEnergy ≠ -1 and
        lowestHPBot.currentEnergy > 3 and lowestHPBot.currentEnergy <
        lowestHPBot.lastEnergy) then
        double bearingTo ← BearingTo(lowestHPBot.X,
            lowestHPBot.Y);
        var enemyBearing ← Direction + bearingTo;
        double rnd ← Math.Sign(rand.NextDouble());
        SetTurnLeft(NormalizeRelativeAngle(enemyBearing + 90 * rnd
            - Direction));
        SetForward(rand.NextDouble() > 0.25 ? 100 : -100);

```

Berikut adalah *pseudocode* dari prosedur Move:

```

private procedure Move(X: double, Y: double):

    var angle ← BearingTo(X, Y);
    SetTurnLeft(angle);
    if (Math.Abs(angle) < 90) then SetForward(50);
    else SetBack(50);

```

Berikut adalah *pseudocode* dari prosedur RunAway:

<u>private procedure</u> RunAway(duration: <u>integer</u> ):
<u>if</u> (stuckCooldown = 0) <u>then</u> isMovingForward $\leftarrow$ <u>not</u> isMovingForward; stuckCooldown $\leftarrow$ duration;

Berikut adalah *pseudocode* dari fungsi CalculatePowerFire:

<u>private function</u> CalculatePowerFire(bd: BotData, hitProbability: <u>double</u> ) $\rightarrow$ <u>double</u>
<u>double</u> distance $\leftarrow$ DistanceTo(bd.X, bd.Y); hitProbability $\leftarrow$ bd.shotCount > 0 ? ( <u>double</u> )bd.hitCount / bd.shotCount : 1; <u>double</u> distanceFactor $\leftarrow$ (1.0 - hitProbability) * 200; <u>double</u> powerFire $\leftarrow$ 0; <u>if</u> (distance < 200 - distanceFactor) <u>then</u> powerFire $\leftarrow$ 3; <u>else if</u> (distance < 400 - distanceFactor) <u>then</u> powerFire $\leftarrow$ 2; <u>else if</u> ((distance < 800 <u>and</u> Energy > bd.currentEnergy + 1)) <u>then</u> powerFire $\leftarrow$ 1; hitProbability $\leftarrow$ (hitProbability * powerFire) / (distance * distance); $\rightarrow$ powerFire;

Berikut adalah *pseudocode* dari prosedur SmartFire:

<u>private procedure</u> SmartFire(bd: BotData, firePower: <u>double</u> ):
<u>if</u> (firePower < 0) <u>then</u> firePower $\leftarrow$ CalculatePowerFire(bd, out _); <u>if</u> (Energy > firePower + 1) <u>then</u> SetFire(firePower);

Berikut adalah *pseudocode* dari prosedur RotateRadar:

<u>private procedure</u> RotateRadar(X: <u>double</u> , Y: <u>double</u> ):
<u>double</u> radarbearing $\leftarrow$ RadarBearingTo(x, y); <u>double</u> margin $\leftarrow$ 5; <u>if</u> (radarbearing > 0) <u>then</u> SetTurnRadarLeft(radarbearing + margin); <u>else</u> SetTurnRadarRight(-radarbearing + margin);

Berikut adalah *pseudocode* dari prosedur OnBulletHit:

<u>public override procedure</u> OnBulletHit(evt: BulletHitBotEvent)
<u>if</u> (scannedBots.TryGetValue(evt.VictimId, <u>out var</u> bdt)) <u>then</u> bdt.hitCount++; bdt.lastEnergy ← bdt.currentEnergy; bdt.currentEnergy ← evt.Energy;

Berikut adalah *pseudocode* dari prosedur OnBulletFired:

<u>public override procedure</u> OnBulletFired(evt: BulletHitBotEvent)
<u>if</u> (scannedBots.TryGetValue(targetId, <u>out var</u> bd)) <u>then</u> bd.shotCount++;

Berikut adalah *pseudocode* dari prosedur OnBotDeath:

<u>public override procedure</u> OnBotDeath(evt: BulletHitBotEvent)
scannedBots.Remove(evt.VictimId);

Berikut adalah *pseudocode* dari prosedur OnScannedBot:

<u>public override procedure</u> OnScannedBot(evt: BulletHitBotEvent)
<u>if</u> (hitCooldown == 0) <u>then</u> <u>var</u> dist ← DistanceTo(evt.X, evt.Y); <u>if</u> (dist < 200 <u>or</u> EnemyCount = 1) <u>then</u> botState ← BotState.BATTLE; <u>var</u> bearing ← BearingTo(evt.X, evt.Y); <u>var</u> gunBearing ← GunBearingTo(evt.X, evt.Y); SetTurnLeft(bearing + 90); SetTurnGunLeft(gunBearing); RotateRadar(evt.X, evt.Y); <u>else then</u> SetTurnRight(0); botState ← BotState.MOVEMENT; <u>if</u> (scannedBots.TryGetValue(evt.ScannedBotId, <u>out</u> BotData bd)) <u>then</u> bd.X ← evt.X; bd.Y ← evt.Y; bd.lastEnergy ← bd.currentEnergy; bd.currentEnergy ← evt.Energy; <u>else then</u> BotData newBd ← new BotData(evt.ScannedBotId, -1, evt.Energy, evt.X, evt.Y, 0, 0); scannedBots[evt.ScannedBotId] ← newBd; targetId ← evt.ScannedBotId;



Berikut adalah *pseudocode* dari prosedur OnHitBot:

```

public override procedure OnHitBot(evt: BulletHitBotEvent)

    if (evt.VictimId  $\neq$  targetId and EnemyCount > 1) then
        botState  $\leftarrow$  BotState.ESCAPE;
        hitCooldown  $\leftarrow$  6;
        RotateRadar(evt.X, evt.Y);
  
```

Berikut adalah *pseudocode* dari prosedur OnHitWall:

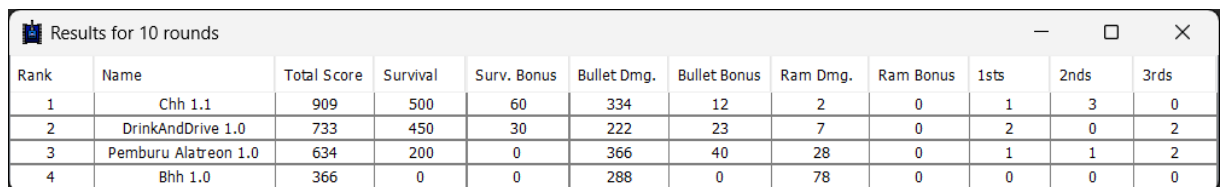
```

public override procedure OnHitWall(evt: BulletHitBotEvent)

    RunAway();
  
```

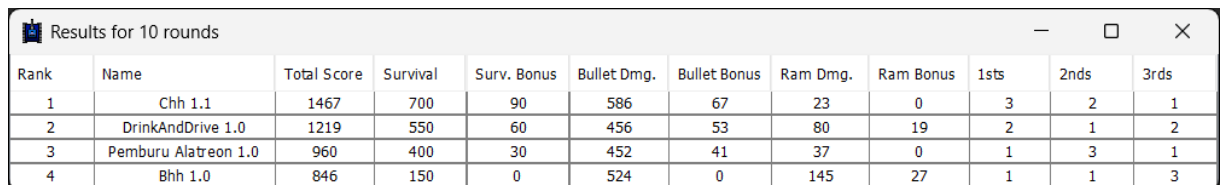
## 4.2. Pengujian dan Analisis

Untuk menguji seberapa efektif dan efisien dari alternatif solusi yang dibuat, dilakukan pengujian dengan melakukan tank royale 1v1v1v1 antar semua bot yang dibuat.



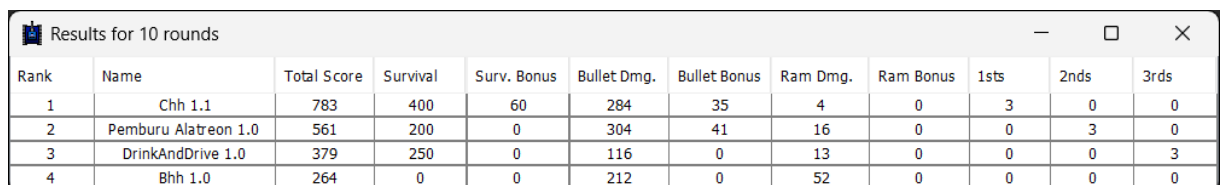
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Chh 1.1	909	500	60	334	12	2	0	1	3	0
2	DrinkAndDrive 1.0	733	450	30	222	23	7	0	2	0	2
3	Pemburu Alatreon 1.0	634	200	0	366	40	28	0	1	1	2
4	Bhh 1.0	366	0	0	288	0	78	0	0	0	0

Gambar 4.2.1. Hasil pengujian ke-1



Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Chh 1.1	1467	700	90	586	67	23	0	3	2	1
2	DrinkAndDrive 1.0	1219	550	60	456	53	80	19	2	1	2
3	Pemburu Alatreon 1.0	960	400	30	452	41	37	0	1	3	1
4	Bhh 1.0	846	150	0	524	0	145	27	1	1	3

Gambar 4.2.2. Hasil pengujian ke-2



Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Chh 1.1	783	400	60	284	35	4	0	3	0	0
2	Pemburu Alatreon 1.0	561	200	0	304	41	16	0	0	3	0
3	DrinkAndDrive 1.0	379	250	0	116	0	13	0	0	0	3
4	Bhh 1.0	264	0	0	212	0	52	0	0	0	0

Gambar 4.2.3. Hasil pengujian ke-3

Berikut adalah hasil analisis setiap alternatif solusi:

#### 4.2.1. *Bot Bhh*

Berdasarkan data hasil pengujian tank royale, terlihat bahwa bot Bhh selalu menempati peringkat terakhir dalam pengurutan skor. Namun, jika kita lihat dari kolom **ram damage** dan **ram bonus**, bot Bhh berhasil mendapatkan skor tertinggi di kolom tersebut dibandingkan dengan bot lainnya. Bot juga bisa dibilang berhasil mendapatkan skor **bullet damage** karena terlihat skor di kolom tersebut bukanlah yang terendah dibanding bot lainnya.

Walaupun begitu, bot Bhh terus mendapatkan skor *survival* terendah karena dengan cara bot ini berperang, bot tidak memedulikan nyawa atau energi nya sendiri, sehingga bot terus menjadi bot yang pertama kali mati di hampir setiap ronde.

#### 4.2.2. *Bot Chh*

Bot Chh berhasil menduduki peringkat pertama di setiap pengujian. Berkat algoritma *greedy*-nya yang berusaha memaksimalkan skor **survival**, bot Chh juga memiliki skor *survival* dan *survival bonus* terbesar dibanding bot lainnya di setiap ronde.

Algoritma *greedy* yang digunakan untuk menembakkan peluru juga dapat dianggap berhasil, dilihat dari kolom **bullet damage** dan **bullet bonus** bahwa bot Chh tidak kalah jauh dari bot-bot lain (bahkan kadang yang tertinggi).

Salah satu masalah yang dialami bot di pengujian adalah ketika bot tersudut di pinggiran atau sudut tembok, bot tidak bisa menghindari dari serangan musuh dengan leluasa, sehingga bot mudah mati.

#### 4.2.3. *Bot Pemburu Alatreon*

Berdasarkan hasil pengujian, Bot Pemburu Alatreon cenderung menempati posisi 2 dan 3. Walaupun begitu jika dilihat dari score bullet damagenya, bot Pemburu Alatreon masih cukup berhasil dalam memaksimalkan damage dengan implementasi algoritma *greedy* dengan heuristik target terdekat dan fire power maksimum untuk target dengan jarak terdekat.

Pada pertarungan pun dapat terlihat bot Pemburu Alatreon konsisten dalam menarget bot dengan jarak terdekat. Kondisi seperti terdapat lebih dari 1 bot yang mana ketika bot Pemburu Alatreon sedang menarget 1 bot lalu terdapat bot lain yang jarak antara bot tersebut dengan bot Pemburu Alatreon lebih dekat dibandingkan jarak antara bot yang sedang ditarget dengan bot Pemburu Alatreon maka, bot Pemburu Alatreon akan menarget bot yang lebih dekat jaraknya.

Untuk heuristik fire power maksimum untuk target dengan jarak terdekat sendiri berhasil diimplementasikan. Hal ini dapat dilihat ketika pertarungan, peluru yang ditembakkan oleh bot Pemburu Alatreon berukuran besar jika jaraknya semakin dekat.

#### 4.2.4. *Bot DrinkAndDrive*

Berdasarkan hasil pengujian, sama seperti Bot Pemburu Alatreon, Bot DrinkAndDrive juga cenderung menempati posisi 2 dan 3. Dilihat juga berdasarkan data hasil pertandingan, muncul inkonsistensi dimana bisa ada kondisi bot ini dapat memperoleh banyak poin ada juga yang tidak, serta perolehan *bullet damage bonus* jugalah bukan yang paling optimal, sehingga penerapan algoritma *greedy by lowest energy* cukup bisa dianggap optimal karena algoritma dimanfaatkan berdasarkan aspek offensif maupun defensif.

Pada pertandingan 1, hal yang paling menonjol adalah aspek defensif karena algoritma terfokus pada menghindari bot-bot yang mendekat sembari menyerang walaupun masih kurang optimal, walaupun begitu bot ini jadi memiliki *survivability* yang lebih baik karena terfokus pada menghindari musuh-musuh yang mendekat.

Sedangkan pada pertandingan 2, momen terbaik dimana bot ini fokus dan seimbang dalam offensif maupun defensif karena serangan terus difokuskan kepada target selagi menghindar dan juga berhasil menabrak musuh lain dalam prosesnya sehingga poin yang didapat dari cara biasa maupun bonus cukup banyak.

Akhirnya, pertandingan 3 merupakan momen terburuk atau *worst case scenario* yang menjadi kondisi hal terburuk karena terlalu dikepung dalam jarak yang dekat menyebabkan kebingungan dalam fokus penentuan mode secara offensif maupun defensif, sehingga kurang optimal dalam bertahan hidup.

## 5. Kesimpulan dan Saran

### 5.1. Kesimpulan

Kesimpulan dari pengembangan algoritma *greedy* untuk robocode antara lain:

1. Algoritma *greedy* cukup efektif jika diimplementasikan pada robocode. Hal ini disebabkan karena dengan menggunakan algoritma *greedy* kita dapat membuat keputusan optimal secara lokal yang berakibat pada perhitungan yang digunakan untuk implementasinya cukup sederhana.
2. Tugas ini memberikan pemahaman mendalam terhadap penggunaan algoritma *greedy*. Tugas ini juga mengajarkan bagaimana heuristik algoritma *greedy* dapat diadaptasi dan dimodifikasi untuk membuat bot yang efektif dan efisien.
3. Algoritma *greedy* masih bisa dikembangkan lebih lanjut. Pengembangan ini dapat dilakukan pada bagian aspek survivalitas yang mana kebanyakan kami mengembangkan algoritma *greedy* untuk aspek menyerang.

### 5.2. Saran

Ada beberapa hal yang masih bisa dikembangkan untuk meningkatkan algoritma *greedy* agar bot yang kami kembangkan lebih optimal. Beberapa aspek lain bisa dikembangkan heuristiknya seperti aspek survivalitas. Heuristik dapat dikembangkan lebih kreatif dan optimal. Testing juga harus dilakukan lebih banyak dan bervariasi pada bot dan algoritma *greedy*-nya.

## Lampiran

- Tabel penyelesaian tugas besar

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.		✓

- Tautan repository

[https://github.com/filbertengyo/Tubes1\\_Pemburu-Tankjil/tree/main](https://github.com/filbertengyo/Tubes1_Pemburu-Tankjil/tree/main)

## Daftar Pustaka

1. [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)
2. [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)
3. [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)
4. <https://robocode-dev.github.io/tank-royale/>