

**IF2211 Strategi Algoritma**  
**Semester II Tahun Akademik 2024/2025**

**Laporan Tugas Besar 1**



*Disusun Oleh:*

Benedict Presley - 13523067

Mochammad Rifqi Rizqulloh - 13523069

Muhammad Jibril Ibrahim - 13523085

K-02

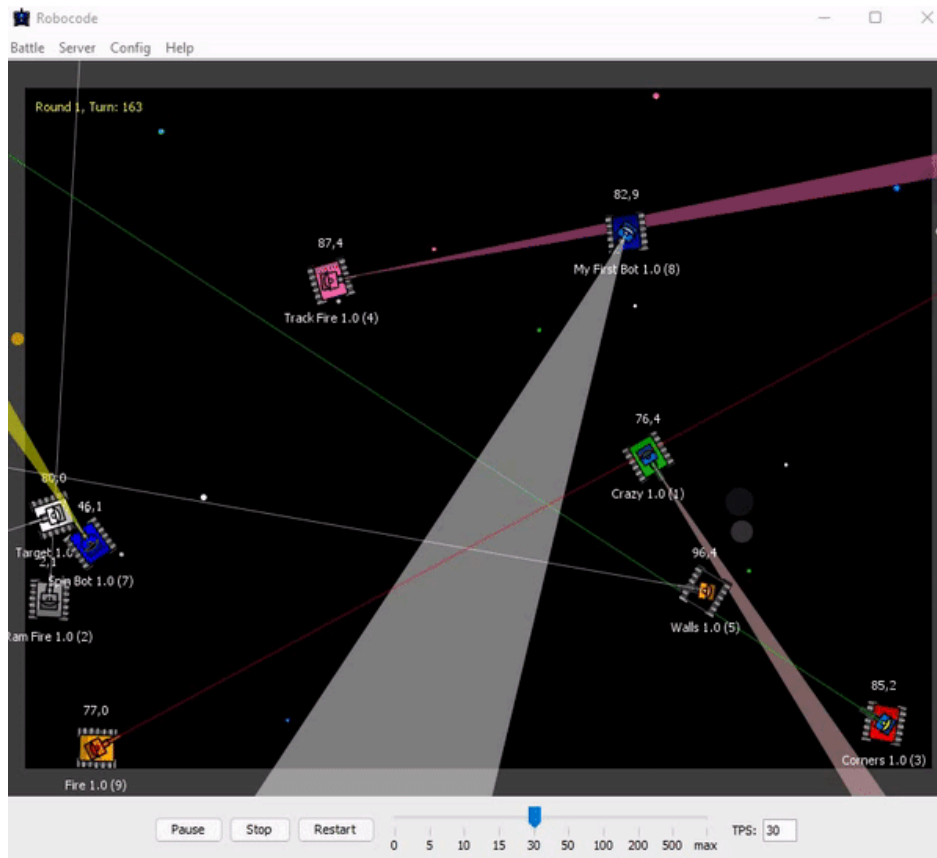
**Sekolah Teknik Elektro dan Informatika**  
**Institut Teknologi Bandung**  
**2025**

# DAFTAR ISI

<b>DAFTAR ISI</b>	<b>2</b>
<b>BAB 1: Deskripsi Tugas</b>	<b>3</b>
<b>BAB 2: Landasan Teori</b>	<b>5</b>
2.1. Algoritma Greedy	5
2.2. Robocode Bot	5
2.2.1. Cara Kerja Bot	5
2.2.2. Implementasi Algoritma Greedy ke Robocode	8
2.2.3. Menjalankan Robocode	8
<b>BAB 3: Aplikasi Strategi Greedy</b>	<b>16</b>
3.1. Elemen-Elemen Greedy pada Robocode	16
3.2. Eksplorasi Solusi	20
3.2.1. Pergerakan Radar	20
3.2.2. Pergerakan Bidikan	21
3.2.3. Pergerakan Tank	23
3.3. Analisis Efisiensi dan Efektivitas Solusi	26
3.3.1. Linear Tracker Bot	26
3.3.2. Adaptive Tracker Bot	28
3.3.3. Spin Shift Bot	29
3.3.4. Snake Bot	29
3.4. Pemilihan Strategi Greedy	30
<b>BAB 4: Implementasi dan Pengujian</b>	<b>31</b>
4.1. Pseudocode	31
4.1.1. Linear Tracker Bot	31
4.1.2. Adaptive Tracker Bot	34
4.1.3. Spin Shift Bot	38

4.1.4. Snake Bot	39
4.2. Source Code	43
4.3. Testing (Pengujian)	50
4.4. Analisis Pengujian	51
<b>BAB 5: Kesimpulan dan Saran</b>	<b>53</b>
5.1. Kesimpulan	53
5.2. Saran	53
5.3. Komentar	53
<b>LAMPIRAN</b>	<b>54</b>
<b>DAFTAR PUSTAKA</b>	<b>54</b>

## BAB 1: Deskripsi Tugas



Robocode adalah permainan pemrograman yang bertujuan untuk membuat kode bot dalam bentuk tank virtual untuk berkompetisi melawan bot lain di arena. Pertempuran Robocode berlangsung hingga bot-bot bertarung hanya tersisa satu seperti permainan Battle Royale, karena itulah permainan ini dinamakan Tank Royale. 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.

## **BAB 2: Landasan Teori**

### **2.1. Algoritma Greedy**

Algoritma greedy adalah algoritma yang memecahkan persoalan secara langkah per langkah (step by step) sedemikian sehingga pada setiap langkah selalu mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan dan “berharap” bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global. Namun, perlu diperhatikan bahwa algoritma greedy tidak selalu menghasilkan solusi yang optimal untuk semua persoalan.

Algoritma greedy terdiri dari beberapa elemen:

1. Himpunan kandidat yang berisi kandidat solusi yang akan dipilih pada setiap langkah
2. Himpunan solusi yang berisi kandidat solusi yang sudah dipilih
3. Fungsi solusi yang menentukan apakah himpunan kandidat yang telah dipilih memberikan solusi persoalan
4. Fungsi seleksi yang memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.
5. Fungsi kelayakan yang memeriksa apakah kandidat yang dipilih layak untuk dimasukkan ke dalam himpunan solusi
6. Fungsi objektif yang memaksimumkan atau meminimumkan solusi

Berdasarkan elemen-elemen tersebut, algoritma greedy dapat didefinisikan sebagai Algoritma yang melibatkan pencarian sebuah himpunan bagian,  $S$ , dari himpunan kandidat,  $C$ , yang dalam hal ini,  $S$  harus memenuhi beberapa kriteria yang ditentukan, yaitu  $S$  menyatakan suatu solusi dan  $S$  di optimisasi oleh suatu fungsi objektif.

## **2.2. Robocode Bot**

### **2.2.1. Cara Kerja Bot**

Robocode adalah sebuah permainan dimana pemain membuat program bot untuk berkompetisi dengan bot-bot pemain lain pada sebuah arena dengan tujuan mengalahkan bot musuh dan bertahan hidup hingga akhir permainan battle royale ini.

Komponen dari permainan ini antara lain:

#### **1. Rounds dan Turns**

Setiap permainan terdiri dari beberapa ronde. Setiap ronde dibagi menjadi beberapa giliran, yang merupakan unit waktu terkecil pada permainan ini. Jumlah giliran bergantung pada waktu untuk tersisa satu bot terakhir pada ronde ini. Pada setiap turn, bot akan menerima informasi terbaru tentang posisinya dan orientasinya di arena. Bot juga akan mendapatkan informasi tentang bot musuh ketika terdeteksi oleh pemindai.

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.

#### **2. Batas Waktu Giliran**

Setiap bot memiliki batas waktu untuk setiap giliran yang disebut turn timeout, biasanya antara 30-50 ms. dampaknya, bot tidak dapat memiliki algoritma lambat untuk menentukan tindakannya. Jika batas waktu ini dilewati, maka server tidak menerima perintah tindakan bot sehingga bot tidak bergerak.

### 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

Menembakkan peluru membutuhkan energi. Semakin besar energi yang digunakan, semakin besar kerusakan yang dihasilkan namun semakin lambat peluru tersebut. Sebaliknya, peluru ringan memiliki kerusakan kecil namun bergerak lebih cepat.

### 5. Panas Meriam

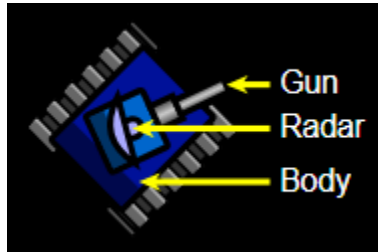
Bot menghasilkan panas saat menembakkan peluru. Peluru dengan energi lebih besar menghasilkan lebih banyak panas dibandingkan peluru dengan energi kecil. Ketika meriam terlalu panas, bot tidak dapat menembakkan peluru hingga suhu meriam turun ke nol. Pada awal ronde, semua bot memiliki panas sehingga tidak bisa langsung menembakkan peluru.

### 6. Tabrakan

Bot akan mendapat kerusakan jika bersentuhan dengan dinding dan bot lain. Tindakan bot menabrak bot musuh dengan bergerak maju disebut ramming yang memberikan skor bagi bot penyerang.

## 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. Perputaran

Setiap bagian tubuh bot bergerak independen satu sama lain. Setiap bagian tubuh memiliki kecepatan putar yang berbeda. Radar adalah bagian tercepat dan dapat berputar hingga 45 derajat per giliran, meriam dapat berputar hingga 20 derajat per giliran, dan tubuh tank dapat berputar hingga 10 derajat per giliran. Kecepatan putar tubuh tank bergantung pada kecepatan bot ini, semakin cepat bot semakin lambat berputar.



## 10. Pemindaian

Bot dapat melakukan pemindaian bot musuh menggunakan radar. Radar dapat mendeteksi bot dengan jarak hingga 1200 px. Bot hanya dapat memindai musuh yang berada dalam jangkauan sudut pemindaian-nya. Sudut pemindaian ini merupakan "sapuan radar" dari arah radar sebelumnya ke arah radar saat ini dalam satu giliran. Jika radar pemindai tidak bergerak maka sudut pemindaian sama dengan nol.

### 2.2.2. Implementasi Algoritma Greedy ke Robocode

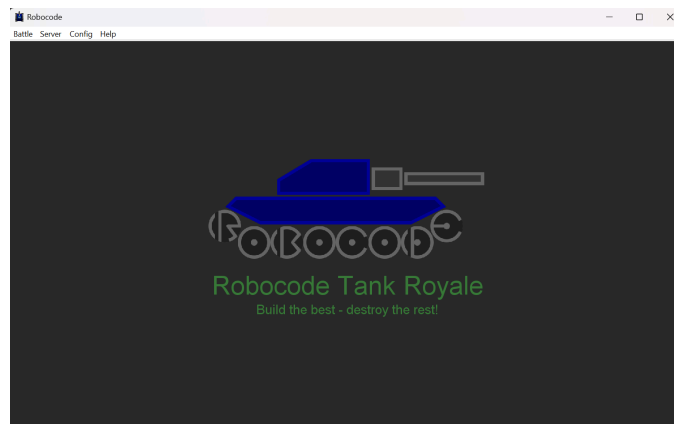
Algoritma Greedy adalah sebuah metode optimasi yang mengambil keputusan secara lokal pada setiap langkah dengan memilih opsi terbaik saat itu tanpa mempertimbangkan konsekuensi jangka panjang. Dalam konteks Robocode, algoritma ini dapat diterapkan dalam berbagai aspek bergantung dengan pendekatan penyelesaian masalah yang digunakan, Contohnya:

1. Menargetkan Musuh Terdekat – Bot akan selalu menyerang musuh yang berada dalam jarak terdekat untuk meningkatkan peluang mengenai sasaran.
2. Menghindari Tembakan – Bot akan bergerak ke posisi yang paling jauh dari titik tembakan terakhir lawan agar meminimalkan risiko terkena serangan.
3. Menentukan Arah Gerakan – Bot akan memilih arah gerakan yang memberikan keuntungan terbesar dalam hal pertahanan atau penyerangan

### 2.2.3. Menjalankan Robocode

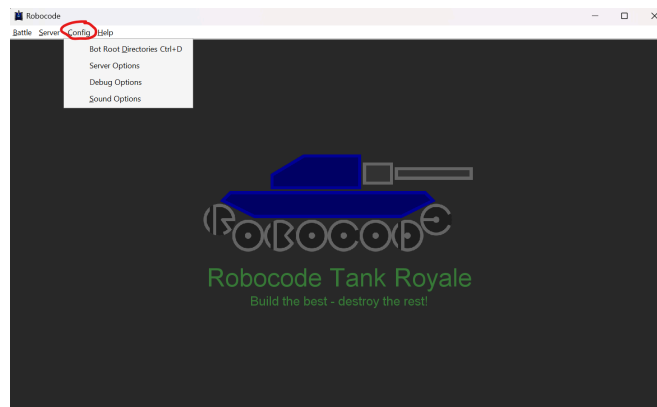
#### 2.2.3.1. Jalankan file .jar aplikasi GUI

```
java -jar robocode-tankroyale-gui-0.30.0.jar
```

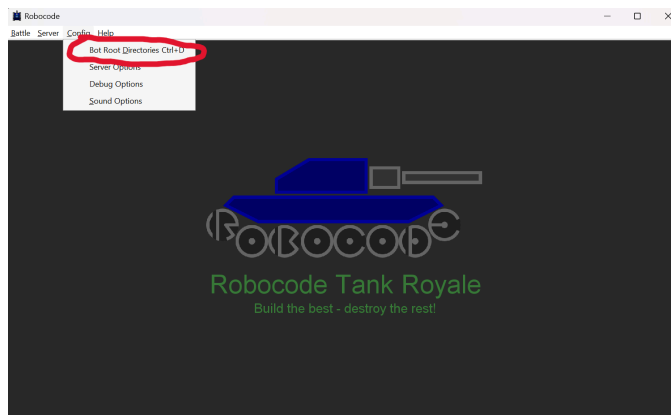


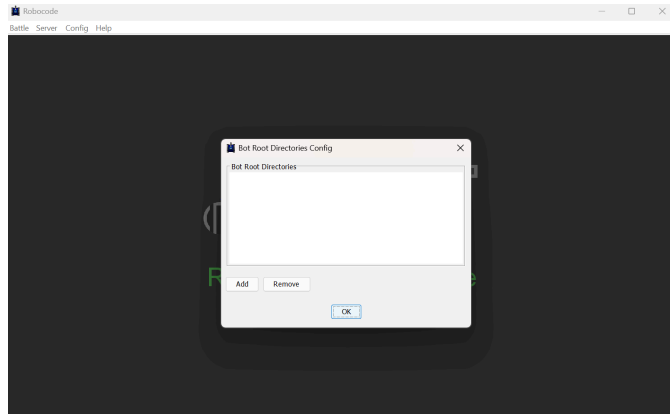
## 2.2.3.2. Setup Konfigurasi Booter

1) Klik tombol "Config"

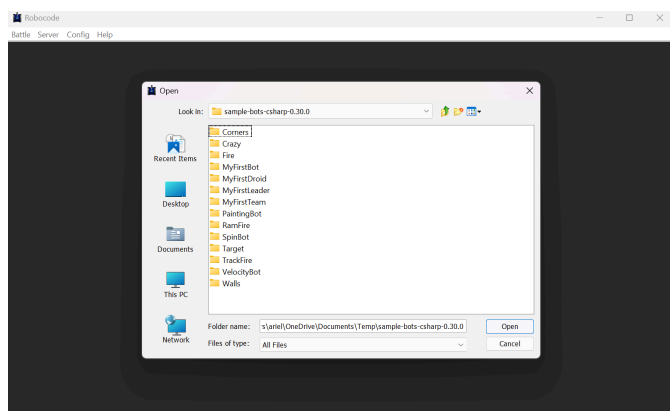
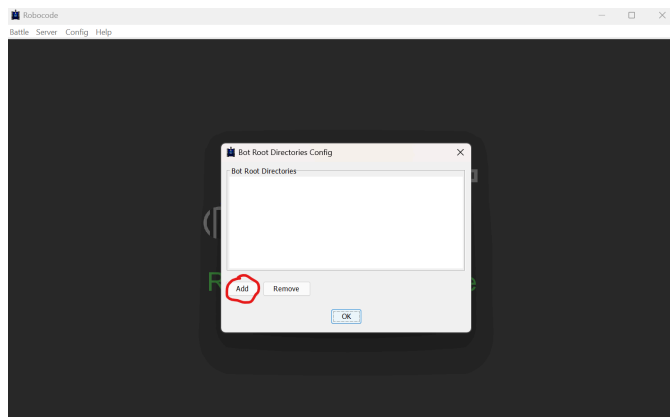


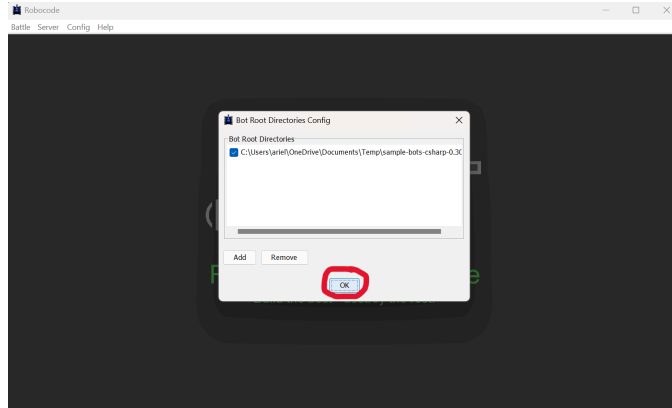
2) Klik tombol "Bot Root Directories"





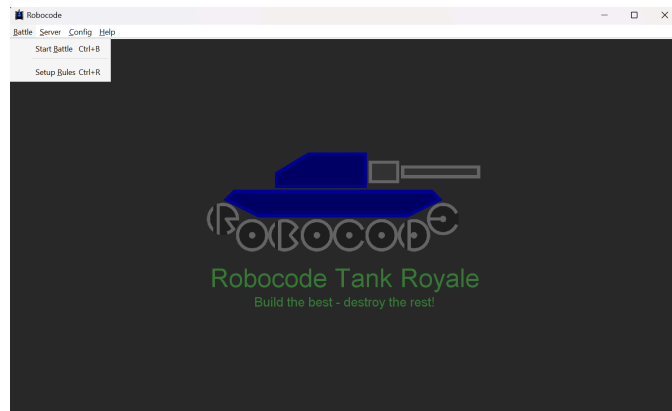
3) Masukkan *directory* yang berisi folder-folder bot kalian



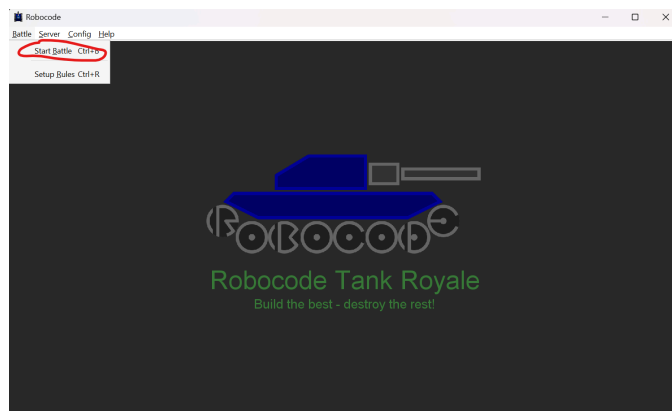


### 2.2.3.3. Jalankan Sebuah Battle

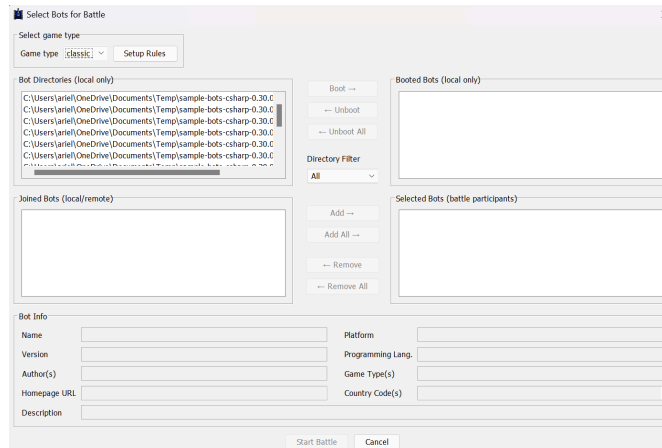
1) Klik tombol "Battle"



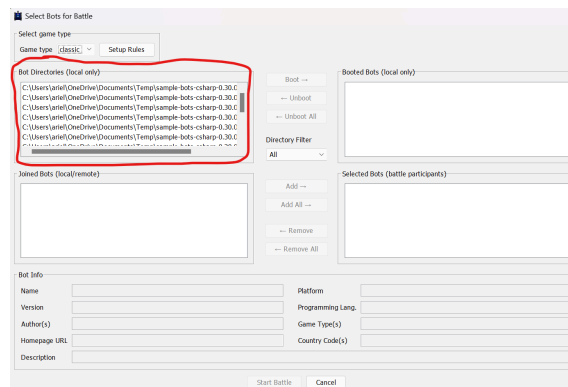
2) Klik tombol "Start Battle"



3) Akan muncul panel konfigurasi permainan

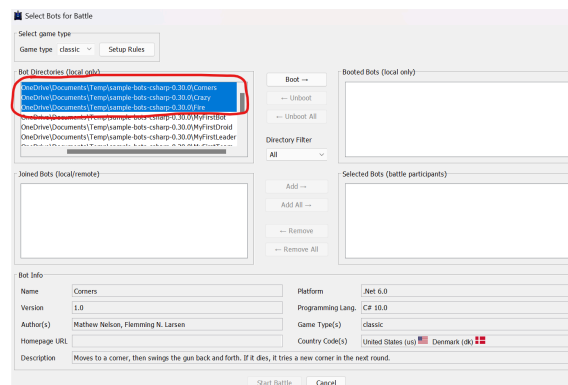


- a) Bot-bot di dalam directory yang telah disetup pada proses konfigurasi akan otomatis muncul pada kotak kiri-atas

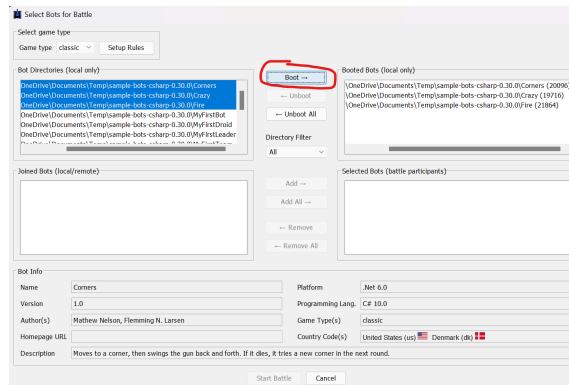


- 4) Boot bot yang ingin Anda mainkan

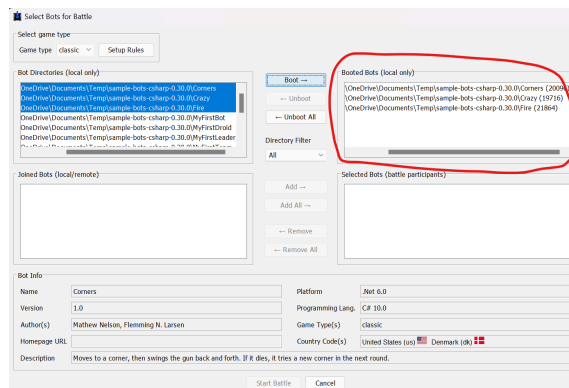
- a) Select bot yang ingin dimainkan pada kotak kiri-atas



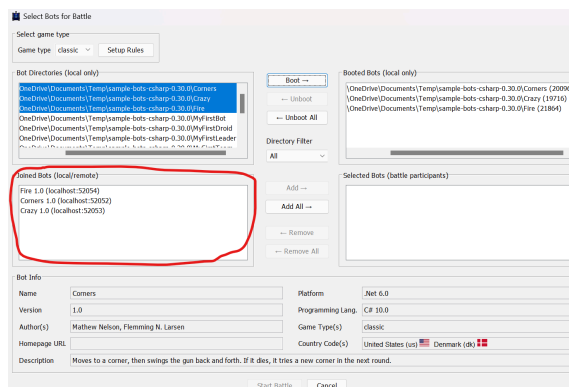
- b) Klik tombol “Boot →”



c) Bot yang telah diselect akan muncul pada kotak kanan-atas

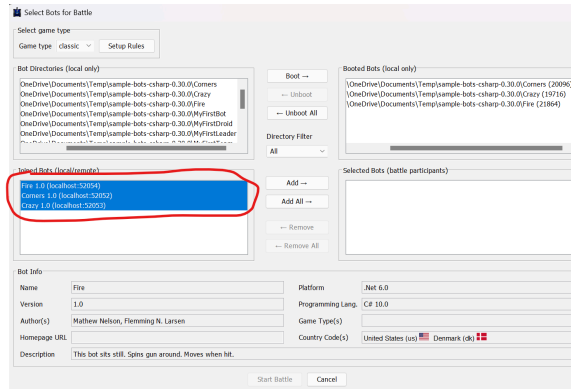


d) Bot yang berhasil di-boot akan muncul pada kotak kiri-bawah

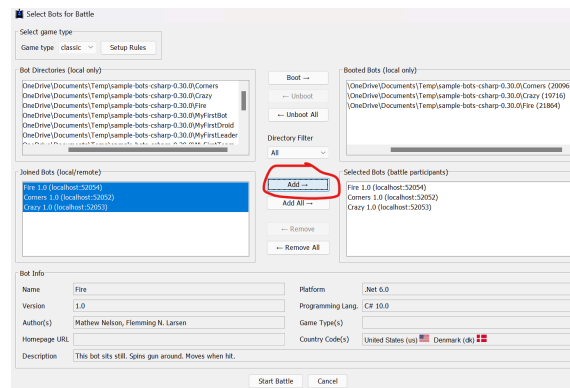


5) Tambahkan bot ke dalam permainan

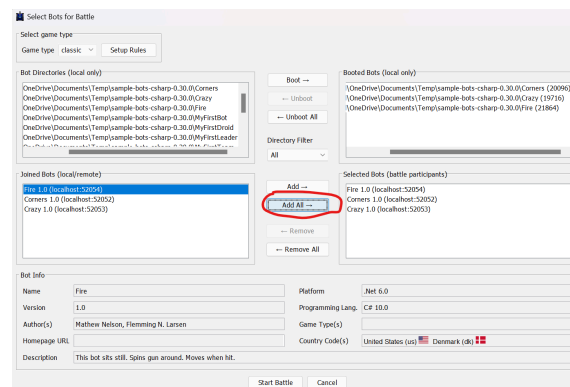
a) Select bot yang ingin ditambahkan ke dalam permainan pada kotak kiri-bawah



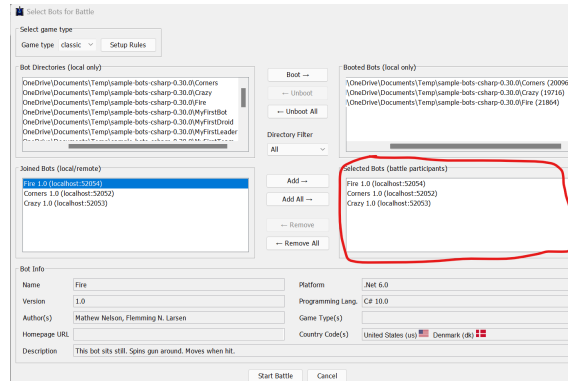
b) Klik tombol “Add →”



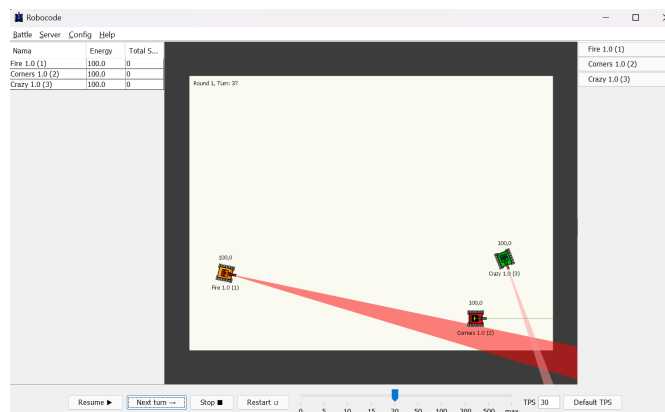
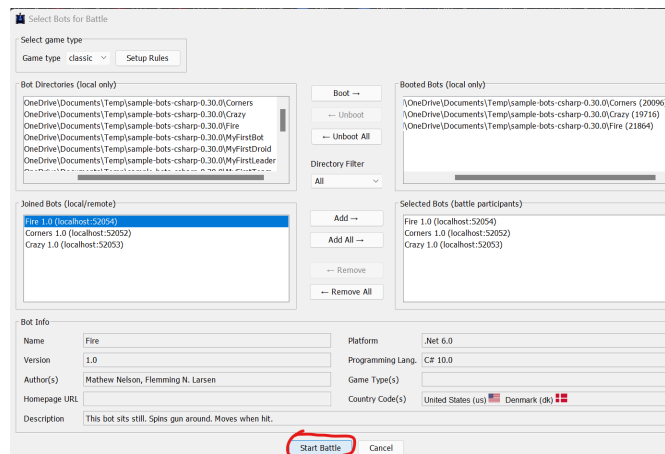
c) Apabila Anda ingin menambahkan semua bot, klik tombol “Add All →”



d) Bot yang telah ditambahkan akan otomatis muncul pada kotak kanan-bawah



6) Mulai permainan dengan menekan tombol “Start Battle”





## **BAB 3: Aplikasi Strategi Greedy**

### **3.1. Elemen-Elemen Greedy pada Robocode**

Dengan menggunakan elemen-elemen yang terdapat dalam algoritma greedy, maka dapat dikatakan bahwa Algoritma greedy melibatkan pencarian terhadap sebuah himpunan bagian,  $S$ , dari himpunan kandidat,  $C$ ; yang dalam hal ini,  $S$  harus memenuhi salah satu atau beberapa kriteria yang telah ditentukan, yaitu  $S$  menyatakan sebuah solusi dan  $S$  dioptimisasi dengan fungsi objektif.

Setelah meninjau persoalan diamond pada permainan ini, kami mencoba untuk melakukan mapping terhadap persoalan tersebut menjadi elemen-elemen algoritma greedy agar mempermudah implementasi algoritma greedy pada program bot permainan ini. Mapping yang dihasilkan dari persoalan tersebut dapat dilihat pada penjelasan di bawah ini

1. Himpunan kandidat ( $C$ ):  
Himpunan kandidat adalah semua gerakan yang mungkin dilakukan oleh bot
2. Himpunan solusi ( $S$ ) :  
Himpunan solusi adalah himpunan-himpunan gerakan yang memberikan poin semaksimal mungkin
3. Fungsi solusi:  
Fungsi untuk menemukan gerakan-gerakan yang memaksimalkan objektif
4. Fungsi seleksi  
Fungsi untuk memutuskan gerakan yang akan dilakukan untuk mendapatkan objektif
5. Fungsi kelayakan  
Semua gerakan pada mungkin adalah gerakan yang layak
6. Fungsi objektif  
Objektif dari algoritma greedy adalah untuk memaksimalkan poin yang didapatkan. Poin dapat didapatkan melalui beberapa cara, yaitu memberi damage kepada bot lain, membunuh bot lain, serta bertahan hidup.

Pada permainan Robocode Tank Royale, terdapat berbagai elemen aksi dan kejadian permainan untuk berjalannya Robocode tersebut. Elemen elemen tersebut kami rangkum ke dalam tabel berikut :

No.	Command Aksi / Event	Kegunaan
1.	<code>SetFire(double p) Fire(double p)</code>	Untuk melancarkan tembakan dengan kekuatan sebesar p
2.	<code>CalcBearing(double dir)</code>	Untuk mengetahui sudut antara <b>direction</b> bot dengan suatu sudut dir
3.	<code>CalcGunBearing(double dir)</code>	Untuk mengetahui sudut antara <b>gun direction</b> bot dengan suatu sudut dir
4.	<code>CalcRadarBearing(double dir)</code>	Untuk mengetahui sudut antara <b>radar direction</b> bot dengan suatu sudut dir
5.	<code>DirectionTo(double x, double y)</code>	Untuk mengetahui sudut dari suatu koordinat (x, y )
6.	<code>BearingTo(double x, double y)</code>	Untuk mengetahui sudut antara <b>direction</b> bot dengan suatu koordinat (x, y)
7.	<code>GunBearingTo(double x, double y)</code>	Untuk mengetahui sudut antara <b>gun direction</b> bot dengan suatu koordinat (x, y)
8.	<code>RadarBearingTo(double x, double y)</code>	Untuk mengetahui sudut antara <b>radar direction</b> bot dengan suatu koordinat (x, y)

9.	<code>DistanceTo(double x, double y)</code>	Untuk mengetahui jarak antara posisi bot saat ini terhadap koordinat (x, y)
10.	<code>OnGameStarted(GameStartedEvent gameStatedEvent)</code>	Penanda event. Digunakan apabila bot perlu untuk melakukan suatu hal pada awal permainan
11.	<code>OnGameEnded(GameEndedEvent gameEndedEvent)</code>	Penanda event. Digunakan apabila bot perlu untuk melakukan suatu hal pada akhir permainan
12.	<code>OnRoundStarted(RoundStartedEvent roundStatedEvent)</code>	Penanda event. Digunakan apabila bot perlu untuk melakukan suatu hal pada awal ronde
13.	<code>OnRoundEnded(RoundEndedEvent roundEndedEvent)</code>	Penanda event. Digunakan apabila bot perlu untuk melakukan suatu hal pada akhir ronde
14.	<code>OnTick(TickEvent tickEvent)</code>	Penanda event. Digunakan apabila bot perlu untuk melakukan suatu hal pada suatu tick tertentu
15.	<code>OnBotDeath(BotDeathEvent botDeathEvent)</code>	Penanda event. Digunakan apabila bot perlu untuk melakukan suatu hal apabila ada bot yang telah hancur

16.	OnDeath(DeathEvent botDeathEvent)	Penanda event. Digunakan apabila bot perlu untuk melakukan suatu hal saat bot tersebut hancur
17.	OnHitBot(HitBotEvent botHitBotEvent)	Penanda event. Digunakan apabila bot perlu untuk melakukan suatu hal saat terbentur dengan bot lain.
18.	OnHitWall(HitWallEvent botHitWallEvent){	Penanda event. Digunakan apabila bot perlu untuk melakukan suatu hal saat terbentur dengan tembok.
19.	OnBulletFired(BulletFiredEvent bulletFiredEvent)	Penanda event. Digunakan apabila bot perlu untuk melakukan suatu hal saat menembakkan peluru. (Suatu bot hanya dapat mendapatkan informasi dari peluru yang ditembakkan oleh bot itu sendiri)
20.	OnHitByBullet(HitByBulletEvent bulletHitBotEvent)	Penanda event. Digunakan apabila bot perlu untuk melakukan suatu hal saat terkena tembakan peluru
21.	OnBulletHit(BulletHitBotEvent bulletHitBotEvent)	Penanda event. Digunakan apabila bot perlu untuk melakukan suatu hal saat peluru yang ditembakkan oleh bot tersebut mengenai bot lain (Suatu bot hanya

		dapat mendapatkan informasi dari peluru yang ditembakkan oleh bot itu sendiri)
22.	OnBulletHitBullet(BulletHitBulletEvent bulletHitBulletEvent)	<p>Penanda event.</p> <p>Digunakan apabila bot perlu untuk melakukan suatu hal saat peluru yang ditembakkan mengenai peluru lain yang ditembakkan oleh bot lain</p>
23.	OnBulletHitWall(BulletHitWallEvent bulletHitWallEvent)	<p>Penanda event.</p> <p>Digunakan apabila bot perlu untuk melakukan suatu hal saat peluru yang ditembakkan tidak mengenai bot apapun dan hancur saat mengenai tembok.</p>
24.	OnScannedBot(ScannedBotEvent scannedBotEvent)	<p>Penanda event.</p> <p>Digunakan apabila bot perlu untuk melakukan suatu hal saat radar mendeteksi adanya bot. Event ini akan mengembalikan informasi-informasi mengenai bot yang dideteksi, yaitu :</p> <ul style="list-style-type: none"> <li>• ID Bot pendeteksi</li> <li>• ID Bot yang dideteksi</li> <li>• Energy bot yang dideteksi</li> <li>• Posisi X bot yang dideteksi</li> <li>• Posisi Y bot yang dideteksi</li> <li>• Direction bot yang dideteksi</li> <li>• Speed bot yang dideteksi</li> </ul>

25.	OnSkippedTurn(SkippedTurnEvent skippedTurnEvent)	<p>Penanda event.</p> <p>Digunakan apabila bot perlu untuk melakukan suatu hal saat terdapat turn yang terskip (bot tidak melakukan apapun pada turn tersebut).</p> <p>Turn skip dapat terjadi apabila proses bot terlalu memakan waktu, atau tidak ada blocking action yang dikirimkan</p>
26.	OnWonRound(WonRoundEvent wonRoundEvent)	<p>Penanda event.</p> <p>Digunakan apabila bot perlu untuk melakukan suatu hal saat bot memenangkan suatu ronde</p>
27.	OnCustomEvent(CustomEvent customEvent)	<p>Penanda event custom</p>

### 3.2. Eksplorasi Solusi

Secara umum, dalam pengembangan bot Robocode, kita dapat memecah logika menjadi tiga komponen utama untuk memudahkan pengelolaan dan pengoptimalan strategi. Pertama, pergerakan radar bertugas untuk mendeteksi musuh secara terus-menerus, biasanya dengan cara memutar radar secara independen agar tidak kehilangan jejak lawan. Kedua, pergerakan bidikan (gun) difokuskan pada mengarahkan meriam ke posisi musuh dan menyesuaikan akurasi tembakan berdasarkan prediksi gerak lawan. Ketiga, pergerakan tank (body) mengatur posisi dan pergerakan keseluruhan robot, baik untuk menghindari tembakan musuh, menjaga jarak ideal, atau mengambil posisi strategis. Dengan memisahkan ketiga

aspek ini, bot dapat dikembangkan secara modular dan lebih responsif terhadap situasi pertempuran.

### **3.2.1. Pergerakan Radar**

Strategi radar dalam Robocode sangat penting karena radar berfungsi sebagai "mata" bot untuk mendeteksi dan melacak musuh. Kelompok kami membuat 2 pilihan strategi untuk pergerakan radar, yaitu :

- *360 Scan*

Pergerakan dari strategi ini adalah radar diputar 360 derajat tanpa henti untuk menjamin bot selalu mendapatkan informasi musuh, terutama saat musuh bergerak cepat atau berpindah arah. Karena ada batasan limitasi dari sistem, radar dapat berputar 1 putaran dalam 8 turn (1 turn hanya dapat berputar maksimal 45 derajat). Selain itu, batasan lain adalah dimana hanya ada 1 bot yang dapat discan di satu waktu.

Bagaimanapun juga, strategi ini memiliki kelemahan yang fatal. Strategi ini memaksa tank untuk menggunakan strategi pembidikan yang sama, yaitu berputar bersama dengan radar. Apabila strategi penembakan yang dipakai tidak sama dengan strategi radar, sangat mungkin terjadi adanya “late reaction” dari bidikan. Sebagai contoh, radar mendeteksi adanya lawan di arah utara, akan tetapi bidikan masih berada di arah barat. Sehingga perlu adanya pergerakan dari arah bidikan terlebih dahulu sebelum peluru ditembak, mengakibatkan adanya “late reaction” tersebut.

- *Tracking Scan*

Gerakan dari strategi ini adalah radar akan dikunci dan mengikuti posisi musuh setelah terdeteksi. Teknik ini memastikan radar terus memperbarui data musuh secara real-time tanpa kehilangan jejak. Untuk menjaga radar tetap terkunci, bot dapat memutar radar sedikit lebih cepat dari pergerakan musuh. Implementasi metode ini dimungkinkan melalui fungsi `RadarBearingTo(double x,`

double  $y$ ). Agar radar tetap konsisten mengunci pergerakan lawan, perlu adanya offset. Offset yang kelompok kami gunakan pada strategi ini adalah 5 derajat.

### 3.2.2. Pergerakan Bidikan

Strategi bidikan merupakan salah satu aspek krusial dalam pengembangan bot Robocode karena seakurat apa pun radar dan sebaik apa pun pergerakan tank, kemenangan tetap bergantung pada seberapa sering tembakan mengenai musuh. Tanpa strategi bidikan yang baik, energi akan terbuang sia-sia dan peluang menang akan menurun. Kelompok kami menawarkan tiga macam strategi utama yang bisa digunakan dalam bidikan, yaitu :

- *Just Shot Directly*

Strategi ini adalah strategi paling *naive*. Dimana, ketika suatu bot dideteksi oleh radar, kita langsung menembakkan peluru langsung ke posisi musuh saat terakhir terdeteksi. Tentunya strategi ini memiliki banyak sekali kekurangan karena tidak memperhitungkan pergerakan musuh.

- *Linear Movement Prediction*

Strategi ini melakukan tembakan dengan asumsi bahwa bot yang sedang dibidik bergerak dengan linear, tidak memperdulikan apakah bot tersebut sedang berbelok. Prediksi pergerakan bot lawan dihitung dengan perhitungan matematika yang melibatkan posisi penembakan peluru, posisi musuh, kekuatan peluru, kecepatan peluru, arah gerak dari bot, serta kecepatan dari bot. Detail dari implementasi tersebut adalah sebagai berikut :

**Misal :**

Posisi tank kita (Posisi penembakan peluru) :  $(x_0, y_0)$

Posisi musuh saat terdeteksi radar :  $(x_e, y_e)$

Arah gerak musuh :  $\theta$

Kecepatan gerak musuh :  $v$

Kekuatan peluru =  $P$



**Jarak ke Musuh :**

$$d = \sqrt{(x_e - x_0)^2 + (y_e - y_0)^2}$$

**Kecepatan Peluru :**

$$v_b = 20 - 3P$$

**Pemodelan Matematis :**

$$\text{Misal } \Delta x = x_e - x_0, \quad \Delta y = y_e - y_0$$

$$(v_x t + \Delta x)^2 + (v_y t + \Delta y)^2 = (v_b t)^2$$

$$\Rightarrow (v_x^2 + v_y^2 - v_b^2)t^2 + 2(\Delta x v_x + \Delta y v_y)t + (\Delta x^2 + \Delta y^2) = 0$$

$$\text{Bentuk umum: } at^2 + bt + c = 0$$

$$\text{dengan: } \begin{cases} a = v_x^2 + v_y^2 - v_b^2 \\ b = 2(\Delta x v_x + \Delta y v_y) \\ c = \Delta x^2 + \Delta y^2 \end{cases}$$

**Menyelesaikan Persamaan :**

Jika diskriminan  $D = b^2 - 4ac \geq 0$ , maka:

$$t = \min(t_1, t_2), \quad t_1 = \frac{-b + \sqrt{D}}{2a}, \quad t_2 = \frac{-b - \sqrt{D}}{2a}$$

Gunakan waktu  $t$  positif terkecil. Jika tidak ada solusi valid, gunakan  $t = 0$  (langsung tembak ke posisi saat ini).

**Prediksi Posisi Musuh :**

$$x_p = x_e + v_x t, \quad y_p = y_e + v_y t$$

### *Arc Movement Prediction*

Strategi ini merupakan percobaan untuk meningkatkan akurasi dari strategi sebelumnya yang hanya berasumsi bahwa gerakan musuh adalah linear. Akan tetapi, karena pada kenyataannya suatu tank bisa belok, akan membentuk pergerakan yang melengkung seperti busur lingkaran (*arc*). Dalam implementasinya, terdapat *unkown variable*, yaitu seberapa cepat suatu tank bergerak (*turn rate*), sehingga terdapat asumsi bahwa *turn rate* dari suatu tank adalah konstan. *Turn rate* bisa didapatkan dengan melakukan perhitungan matematis sederhana pada perbedaan *direction* dari suatu tank pada 2 tick berurutan.

**Misal :**

$$\theta_{\text{prev}} = \text{arah musuh pada waktu } t - 1$$
$$\theta_{\text{curr}} = \text{arah musuh pada waktu } t$$

**Maka, Turn Rate :**

$$\Delta\theta = \theta_{\text{curr}} - \theta_{\text{prev}}$$

- *Gabungan : Linear + Arc Movement Prediction*

Strategi ini merupakan gabungan dari 2 strategi sebelumnya, dimana akan dilakukan check terlebih dahulu apakah gerakan musuh yang saat ini di scan adalah gerakan linear atau gerakan melengkung. Dengan demikian, diharapkan strategi ini dapat dengan lebih baik memprediksi gerakan musuh. Metode pengecekan dilakukan dengan statistika sederhana memanfaatkan data gerakan di turn sebelumnya. Apabila *direction* pada turn sebelumnya sama dengan *direction* pada turn saat ini, artinya bot tersebut sedang melakukan gerakan linear.

### 3.2.3. Pergerakan Tank

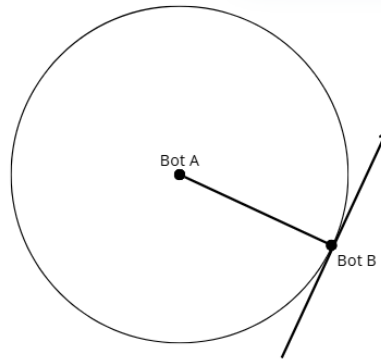
Strategi pergerakan merupakan salah satu aspek terpenting dalam pengembangan bot Robocode karena menentukan seberapa baik bot dapat menghindari tembakan musuh dan mempertahankan posisi yang menguntungkan selama pertempuran. Bot yang hanya diam di tempat akan sangat mudah diprediksi dan menjadi sasaran empuk, sedangkan bot yang terus bergerak dengan pola yang tepat dapat mempersulit musuh untuk membidik secara akurat sekaligus mencari celah untuk menyerang. Dengan pergerakan yang cerdas, sebuah bot tidak hanya bisa bertahan lebih lama, tetapi juga memaksa musuh melakukan kesalahan.

- *Planetary Movement*

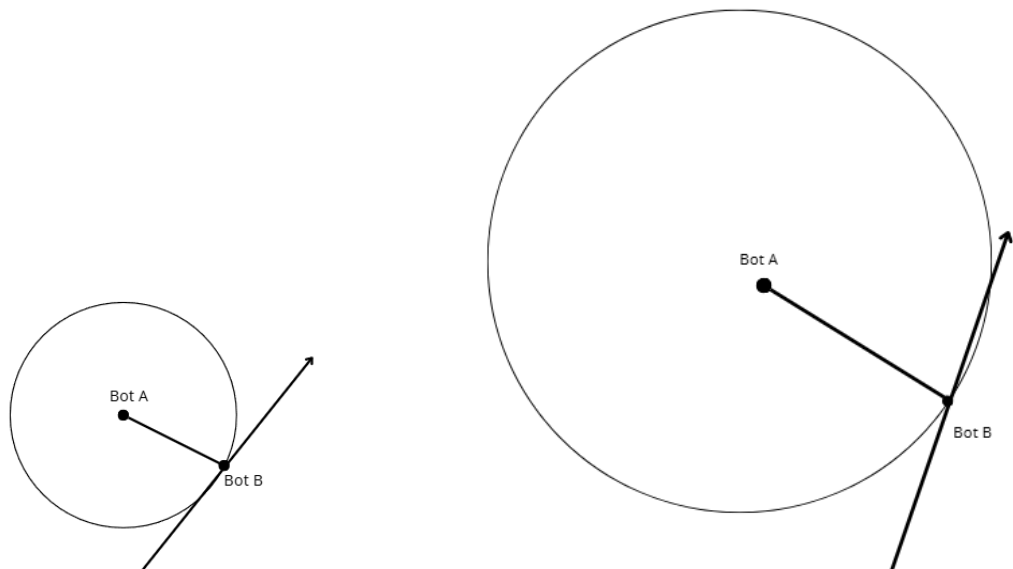
Strategi pergerakan ini merupakan strategi yang terinspirasi oleh pergerakan planet, yaitu suatu planet akan memiliki orbit mengitari matiharinya masing-masing. Dalam kasus robocode, strategi pergerakan ini akan mencoba untuk memilih satu target, dan bergerak mengitari bot tersebut sampai bot tersebut hancur. Setelah robot tersebut hancur, bot akan memilih 1 target baru.

Dalam implementasinya, movement dari bot ini akan selalu berusaha untuk menjaga jarak sejauh 100 - 120 dari bot target. Apabila jarak kurang dari 100, maka bot akan berusaha menjauh. Sebaliknya, apabila jarak bot lebih dari 120, maka bot akan berusaha mendekat. Hal tersebut ditujukan agar untuk meningkatkan akurasi dari tembakan bot tersebut, karena semakin dekat jarak, semakin mungkin peluru tepat sasaran.

Apabila jarak bot dengan targetnya sejauh 100 - 120, maka bot akan berbelok menuju arah 90 derajat + sudut antara 2 bot. Artinya, direction dari bot akan selalu tegak lurus terhadap jari-jari lingkaran lintasan yang terbentuk. Direction dari bot akan di update setiap tick atau turn, sehingga jalur pergerakan bot tersebut akan sangat menyerupai lingkaran.



Sedangkan apabila jarak antar bot kurang dari 100 atau 120, belokan bot akan diberikan offset agar bot kembali menuju orbit yang sesuai. Jika kurang dari 100, maka offset ditujukan agar arah dari pergerakan bot sedikit menjauh dari target. Sebaliknya, jika lebih dari 120, maka offset ditujukan agar arah dari pergerakan bot akan ditujukan agar arah dari pergerakan bot sedikit mendekati target. Offset yang digunakan menyesuaikan terhadap seberapa dekat / seberapa jauh bot tersebut.



- *Randomized Circular Movement*

Strategi ini memanfaatkan pergerakan melingkar yang lebih sulit dilacak dibandingkan dengan pergerakan linear. Bot akan mengitari suatu titik pusat yang

ditentukan secara acak. Setiap beberapa langkah, titik pusat akan bergeser ke titik acak lain sehingga membuat pergerakan bot lebih sulit dilacak.

Dalam kasus bot menabrak bot lain, maka bot akan menyerang bot tersebut kemudian kabur. Dalam kasus bot menabrak dinding, maka bot akan bergerak ke pusat untuk mendapat lebih banyak ruangan.

Bot akan menembakkan peluru setiap kali mendeteksi bot lain.

- *Dodging Strategy and Snake Movement*

Strategi ini adalah strategi untuk menjauh dari musuh yang memiliki jarak kurang dari 100 px. Arah kabur ditentukan dengan memperhitungkan letak dan jarak musuh serta dinding arena. Rumus yang digunakan untuk menghitung arah menghindari dari musuh adalah sebagai berikut:

$$rad = \frac{direction \times \pi}{180}$$

$$dodge_x = -\cos(rad) \times \frac{100.0}{distance + 1.0}$$

$$dodge_y = -\sin(rad) \times \frac{100.0}{distance + 1.0}$$

$$total\_weight += weight$$

Dilanjutkan dengan memperhitungkan jarak dengan dinding arena,

$$wall\_weight = boundary\_weight \times \frac{100 - dist\_to\_wall}{100.0}$$

$$dodge = dodge + wall\_weight$$

$$total\_weight = dodge + wall\_weight$$

\*dodge seharusnya dodge\_x atau dodge\_y bergantung untuk dinding horizontal atau vertical

Lalu untuk menghitung arah menghindari digunakan rumus berikut,

$$dodge_x = dodge_x / total\_weight$$

$$dodge_y = dodge_y / total\_weight$$

$$result\_angle = \tan^{-1} \left( \frac{dodge_y}{dodge_x} \right) \times \frac{180}{\pi}$$

Setelah mendapat arah menghindar, bot melakukan strategi pergerakan lain yaitu Snake Movement dimana bot bergerak seperti ular (zig-zag) dengan tujuan untuk menghindar dari bidikan musuh yang menggunakan linear tracker. Gerakan ini diimplementasikan dengan menggerakkan bot untuk belok 90 derajat ke kiri selagi bot bergerak maju lalu belok 90 derajat ke kanan selagi bergerak maju.

### 3.3. Analisis Efisiensi dan Efektivitas Solusi

#### 3.3.1. Adaptive Tracker Bot

Bot ini merupakan hasil perpaduan dari tiga strategi utama yang saling mendukung: pergerakan radar tracking, bidikan linear prediction, dan pergerakan tank *planetary movement*. Kombinasi ini dirancang untuk menciptakan bot yang tidak hanya agresif dan akurat, tetapi juga sulit untuk dikalahkan.

Strategi radar tracking memungkinkan bot untuk secara konsisten menjaga informasi posisi musuh dengan mengunci radar pada target utama. Dengan radar yang selalu mengikuti pergerakan lawan, bot dapat memperbarui data posisi dan arah musuh secara real-time, yang sangat penting untuk menjaga efektivitas strategi lainnya.

Pada sisi serangan, bot menggunakan strategi linear prediction untuk membidik. Dengan memanfaatkan kecepatan dan arah gerak musuh saat ini, bot dapat memperkirakan posisi musuh di masa depan saat peluru mencapai target. Hal ini jauh lebih akurat dibandingkan dengan menembak langsung ke posisi musuh saat ini, terutama ketika musuh bergerak dalam garis lurus atau kecepatan konstan.

Sementara itu, strategi planetary movement pada pergerakan tank berperan penting dalam pertahanan dan positioning. Dengan mengitari satu bot musuh secara melingkar dan menjaga jarak ideal antara 100 hingga 120 satuan, bot menjadi target yang sulit diprediksi dan sulit ditembak.

Solusi ini akan sangat efektif melawan bot dengan gerakan apapun, kecuali kamikaze bot.

Solusi ini tidak akan efektif apabila bertemu bot agresif yang akan langsung menargetkan bot ini dan menantang bot ini dalam pertarungan jarak yang sangat dekat, atau biasa disebut dengan kamikaze bot

Pada pengujian melawan beberapa Bot Template yang sudah disediakan, yaitu CrazyBot dan SpinBot. Ketika melawan CrazyBot, strategi ini mendapatkan Hit Rate sebesar 49.3%, dimana dari 286 peluru yang ditembakkan, 141 diantaranya tepat sasaran.

```
362 Bullet Shot : 286  
Bullet Hit : 141 Hit Rate : 0,493006993006993  
Game has ended
```

Sementara pada saat menghadapi SpinBot, strategi ini mendapatkan Hit Rate sebesar 72.3%, dimana dari 282 peluru yang ditembakkan, 204 diantaranya tepat sasaran.

```
404 Bullet Shot : 282  
Bullet Hit : 204 Hit Rate : 0,723404255319149  
Game has ended
```

### 3.3.2. Linear Tracker Bot

Bot ini merupakan hasil perpaduan dari tiga strategi utama yang saling mendukung: pergerakan radar tracking, bidikan linear prediction, dan pergerakan tank *planetary movement*. Kombinasi ini dirancang untuk menciptakan bot yang tidak hanya agresif dan akurat, tetapi juga sulit untuk dikalahkan.

Strategi radar tracking memungkinkan bot untuk secara konsisten menjaga informasi posisi musuh dengan mengunci radar pada target utama. Dengan radar yang selalu mengikuti pergerakan lawan, bot dapat memperbarui data posisi dan arah musuh secara real-time, yang sangat penting untuk menjaga efektivitas strategi lainnya.

Pada sisi serangan, bot menggunakan strategi linear prediction untuk membidik. Dengan memanfaatkan kecepatan dan arah gerak musuh saat ini, bot dapat memperkirakan posisi musuh di masa depan saat peluru mencapai target. Hal ini jauh lebih akurat dibandingkan dengan menembak langsung ke posisi musuh saat ini, terutama ketika musuh bergerak dalam garis lurus atau kecepatan konstan.

Sementara itu, strategi planetary movement pada pergerakan tank berperan penting dalam pertahanan dan positioning. Dengan mengitari satu bot musuh secara melingkar dan menjaga jarak ideal antara 100 hingga 120 satuan, bot menjadi target yang sulit diprediksi dan sulit ditembak.

Solusi ini akan sangat efektif melawan bot dengan gerakan linear. Akan tetapi juga masih efektif untuk melawan bot pada umumnya, termasuk bot dengan arah gerakan yang melingkar meskipun tidak seefektif pada saat melawan bot dengan gerakan linear

Solusi ini tidak akan efektif apabila bertemu bot agresif yang akan langsung menargetkan bot ini dan menantang bot ini dalam pertarungan jarak yang sangat dekat, atau biasa disebut dengan kamikaze bot

Pada pengujian melawan beberapa Bot Template yang sudah disediakan, yaitu CrazyBot dan SpinBot. Ketika melawan CrazyBot, strategi ini mendapatkan Hit Rate sebesar 37.5%, dimana dari 325 peluru yang ditembakkan, 122 diantaranya tepat sasaran.

```
485 Bullet Shot : 325  
Bullet Hit : 122 Hit Rate : 0,37538461538461537  
Game has ended
```



Sementara pada saat menghadapi SpinBot, strategi ini mendapatkan Hit Rate sebesar 46.2%, dimana dari 599 peluru yang ditembakkan, 277 diantaranya tepat sasaran.

```
504 Bullet Shot : 599
Bullet Hit : 277 Hit Rate : 0,46243739565943237
Game has ended
```

### 3.3.3. Spin Shift Bot

Bot ini mengutamakan pergerakan yang sulit diikuti dan sulit ditebak sehingga dapat bertahan lebih lama. Ide *greedy* dari bot ini adalah bertahan selama mungkin dengan pergerakan sirkuler dan acak. Bot akan menembak setiap kali melihat bot lain, pergerakan bot yang memiliki sifat acak membuat akurasi bot tidak terlalu bagus karena untuk membuat bot yang memiliki akurasi tinggi diperlukan gerakan yang mendukung juga.

Solusi ini akan efektif bila tidak banyak bot yang berada di arena. Bot ini sangat rawan terhadap ramming secara tidak sengaja ataupun menabrak dinding. Dalam keadaan arena cukup kosong, bot dapat bergerak lebih leluasa sehingga gerakan bot menjadi efektif. Selain itu, solusi ini juga efektif dalam menghadapi bot yang tidak didesain untuk 1v1.

Dalam pengujian melawan bot yang pasif di arena yang kosong, yaitu Corner Bot, dapat dilihat bahwa Spin Shift Bot bertahan dengan sangat baik.

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Spin Shift Bot 1.0	1238	350	70	672	135	11	0	7	0	0
2	Corners 1.0	56	0	0	50	0	6	0	0	7	0

Solusi ini tidak akan efektif bila arena padat. Setiap kali bot menabrak bot lain atau dinding, bot akan memperbaiki posisinya dan menuju lokasi yang lebih kosong, namun dalam proses ini, gerakan bot menjadi mudah ditebak dan bot tetap rawan terhadap menabrak bot lain atau dinding. Solusi ini tidak akan efektif bila melawan bot yang pandai menghindar.

Dalam pengujian di arena yang padat, yaitu melawan beberapa Crazy Bot, Spin Shift Bot tidak bertahan dengan baik karena sulit menembak dengan akurat dan terus menabrak bot lain.

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Crazy 1.0	647	450	40	124	5	28	0	1	1	2
2	Crazy 1.0	551	350	40	132	1	28	0	1	0	1
3	Spin Shift Bot 1.0	498	300	0	176	6	16	0	0	1	1
4	Crazy 1.0	390	200	40	124	0	26	0	2	0	0
5	Crazy 1.0	324	200	0	100	0	24	0	0	2	0

Dalam pengujian melawan bot yang pandai menghindar, yaitu Adaptive Tracker Bot yang merupakan main bot untuk kelompok ini, Spin Shift Bot kalah telak.

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	AdaptiveTracker 1.0	1708	500	100	916	184	8	0	10	0	0
2	Spin Shift Bot 1.0	26	0	0	16	0	10	0	0	10	0

### 3.3.4. Snake Bot

Bot ini mengutamakan tindakan menghindar dari bot lain dan mengutamakan keselamatan diri. Untuk itu, bot memperhitungkan lokasi-lokasi bot dan dinding arena untuk mencari lokasi yang jauh dari semua itu. Bot mencari sudut menghindar dengan memperhitungkan sudut tiap musuh dan jarak mereka. Selagi menghindar, Setiap kali bot mendeteksi musuh maka akan menembakkan peluru agar memberikan damage dan mendapatkan score.

Solusi ini akan efektif bila melawan bot yang menyerang dari jarak jauh atau bot yang tidak agresif mendekati bot lain. Karena Snake bot dapat terus menghindar dengan baik dan tidak terpojok ke corner.

Solusi ini tidak akan efektif bila melawan bot yang agresif mendekati dan dengan bot yang memiliki tracking terhadap gerakan arc seperti pada salah satu bot yang dibuat, adaptive tracker bot.

### **3.4. Pemilihan Strategi Greedy**

Setelah meninjau kelebihan dan kekurangan tiap bot yang telah dibuat, bot utama yang dipilih sebagai perwakilan dari kelompok Atmint Cabang DoaAyahRestulbu adalah Adaptive Tracker bot. Konsiderasi utama yang digunakan yaitu dilihat dari sisi poin yang didapatkan tiap bot. Keagresifan Adaptive Tracker dalam menyerang musuh menyebabkan lebih banyak poin yang didapatkan dibandingkan dengan bot lain. Konsiderasi lain adalah kemampuan membidik tiap bot. Kemampuan Adaptive Tracker bot dalam membidik musuh yang bergerak baik secara linear maupun sirkular menjadikan dia jauh lebih akurat dibandingkan bot-bot yang lain. Meskipun setelah melakukan testing didapatkan bahwa Linear Tracker bot menang melawan Adaptive Tracker bot, Kami menilai bahwa kejadian tersebut adalah kasus khusus yang muncul akibat keunggulan Linear Tracker bot dalam membidik target yang bergerak linear dibandingkan Adaptive Tracker bot.

## BAB 4: Implementasi dan Pengujian

### 4.1. Pseudocode

#### 4.1.1. Adaptive Tracker Bot

```
AdaptiveTrackerBot()  
    l_board <- 100  
    r_board <- 700  
    u_board <- 100  
    d_board <- 500  
    clockwise <- 1  
    running_away <- false  
    is_shooting <- false  
  
    enemies <- array of 4 elements (alive, direction,  
bearing, distance)  
  
    Start()  
  
GetPredictedShot(myX, myY, enemyX, enemyY, enemyHeading,  
enemyVelocity)  
    enemyHeading <- enemyHeading *  $\pi$  / 180  
    dx <- enemyX - myX  
    dy <- enemyY - myY  
    distance <- sqrt(dx^2 + dy^2)  
  
    IF distance < 75 THEN  
        bulletPower <- 4.0  
    ELSE IF distance < 100 THEN  
        bulletPower <- 3.0  
    ELSE IF distance < 125 THEN  
        bulletPower <- 2.0  
    ELSE IF distance < 175 THEN
```

```

bulletPower <- 1.0
ELSE
bulletPower <- 0.5

bulletSpeed <- 20.0 - 3.0 * bulletPower
vx <- enemyVelocity * cos(enemyHeading)
vy <- enemyVelocity * sin(enemyHeading)

a <- vx^2 + vy^2 - bulletSpeed^2
b <- 2.0 * (dx * vx + dy * vy)
c <- dx^2 + dy^2

discriminant <- b^2 - 4.0 * a * c
t <- 0.0

IF abs(a) > 1e-9 AND discriminant >= 0.0 THEN
sqrtDisc <- sqrt(discriminant)
t1 <- (-b + sqrtDisc) / (2.0 * a)
t2 <- (-b - sqrtDisc) / (2.0 * a)

IF t1 > 0.0 AND t2 > 0.0 THEN
t <- min(t1, t2)
ELSE IF t1 > 0.0 THEN
t <- t1
ELSE IF t2 > 0.0 THEN
t <- t2
ELSE
t <- 0.0
ELSE
t <- 0.0

predictedX <- enemyX + vx * t
predictedY <- enemyY + vy * t
firingAngle <- atan2(predictedY - myY, predictedX -

```

```

myX) * 180 /  $\pi$ 

RETURN (bulletPower, firingAngle)

Run()
  BodyColor <- black

  AdjustRadarForBodyTurn <- false
  AdjustGunForBodyTurn <- true
  AdjustRadarForGunTurn <- false

  range <- 1
  cur_enemy_count <- EnemyCount
  running_away <- false
  is_shooting <- false

  WHILE IsRunning DO
    IF NOT running_away AND NOT is_shooting THEN
      SetTurnGunLeft(20)
      SetTurnLeft(MaxTurnRate / range)
      SetForward(20 * clockwise)
      Go()

    IF position_on_edge() AND NOT running_away THEN
      run_angle <- calcDodgeAngle()
      snakeMove(CalcBearing(run_angle), 100)
      clockwise <- clockwise * -1

    IF EnemyCount < cur_enemy_count THEN
      cur_enemy_count <- EnemyCount
      FOR i <- 0 TO 3 DO
        enemies[i].alive <- false

calcDodgeAngle()

```

```

dodge_x <- 0
dodge_y <- 0
total_weight <- 0

FOR i <- 0 TO 3 DO
    IF enemies[i].alive THEN
        rad <- enemies[i].direction *  $\pi$  / 180
        distance <- enemies[i].distance
        weight <- 100.0 / (distance + 1.0)
        dodge_x <- dodge_x - cos(rad) * weight
        dodge_y <- dodge_y - sin(rad) * weight
        total_weight <- total_weight + weight

boundary_weight <- 70.0
battlefield_width <- 800
battlefield_height <- 600

dist_to_left <- X
dist_to_right <- battlefield_width - X
dist_to_bottom <- Y
dist_to_top <- battlefield_height - Y

IF dist_to_left < 100 THEN
    wall_factor <- (100 - dist_to_left) / 100.0
    wall_weight <- boundary_weight * wall_factor
    dodge_x <- dodge_x + wall_weight
    total_weight <- total_weight + wall_weight

IF dist_to_right < 100 THEN
    wall_factor <- (100 - dist_to_right) / 100.0
    wall_weight <- boundary_weight * wall_factor
    dodge_x <- dodge_x - wall_weight
    total_weight <- total_weight + wall_weight

```

```

IF dist_to_bottom < 100 THEN
    wall_factor <- (100 - dist_to_bottom) / 100.0
    wall_weight <- boundary_weight * wall_factor
    dodge_y <- dodge_y + wall_weight
    total_weight <- total_weight + wall_weight

IF dist_to_top < 100 THEN
    wall_factor <- (100 - dist_to_top) / 100.0
    wall_weight <- boundary_weight * wall_factor
    dodge_y <- dodge_y - wall_weight
    total_weight <- total_weight + wall_weight

IF total_weight > 0 THEN
    dodge_x <- dodge_x / total_weight
    dodge_y <- dodge_y / total_weight

result_angle <- atan2(dodge_y, dodge_x) * 180 /  $\pi$ 
RETURN (result_angle + 360) % 360

snakeMove(angle, distance)
    IF NOT is_shooting THEN
        SetTurnGunLeft(10_000)

    running_away <- true
    TurnLeft(angle)

    turn_needed <- 90 / MaxTurnRate
    i <- 0
    WHILE i * MaxSpeed * turn_needed < distance DO
        IF NOT is_shooting THEN
            SetTurnGunLeft(10_000)
            SetTurnLeft(90 * (-1)^i)
            Forward(MaxSpeed * turn_needed)
            i <- i + 1

```



```

    running_away <- false

OnScannedBot(e)
    distance <- DistanceTo(e.X, e.Y)
    direction <- DirectionTo(e.X, e.Y)
    bearing <- BearingTo(e.X, e.Y)

    PRINT "bot id: " + e.ScannedBotId

    enemies[e.ScannedBotId - 1].alive <- true
    enemies[e.ScannedBotId - 1].direction <- direction
    enemies[e.ScannedBotId - 1].bearing <- bearing
    enemies[e.ScannedBotId - 1].distance <- distance

    IF distance < 200 AND NOT running_away THEN
        run_angle <- calcDodgeAngle()
        PRINT "Run Away Angle: " + run_angle
        snakeMove(CalcBearing(run_angle), 100)

    IF shoot_id % 2 = 1 THEN
        shot <- GetPredictedShot(X, Y, e.X, e.Y,
e.Direction, e.Speed)
        bulletPower <- shot[0]
        firingAngle <- shot[1]

        gunTurn <- NormalizeRelativeAngle(firingAngle -
GunDirection)

        is_shooting <- true
        IF gunTurn < 2.5 THEN
            SetTurnGunLeft(gunTurn)
            Fire(bulletPower)
            shoot_id <- e.ScannedBotId

```

```

        PRINT "shoot bih"
        is_shooting <- false

        shoot_id <- shoot_id + 1

position_on_edge()
    RETURN X <= l_board OR X >= r_board OR Y <= u_board OR
Y >= d_board

OnBotDeath(botDeathEvent)
    running_away <- false

```

#### 4.1.2. Linear Tracker Bot

```

linear_tracker()
    l_board <- 100
    r_board <- 700
    u_board <- 100
    d_board <- 500
    minimal_turn_to_reverse <- 10
    this_hp_bef <- 100
    enemy_hp_bef <- 100
    enemy_id_bef <- -1
    target_X <- 0
    target_Y <- 0

    clockwise <- true
    hit_bot_turn_recovery <- 0

    Start()

getOffset1(distance)
    IF distance >= 120 THEN

```

```

        offset <- 90 - (distance - 120) * 0.2
        RETURN Max(offset, 60)
ELSE IF distance <= 100 THEN
    offset <- 90 + (100 - distance) * 0.8
    RETURN Min(offset, 120)
ELSE
    RETURN 90

getOffset2(distance)
    IF distance <= 100 THEN
        offset <- 90 - (100 - distance) * 0.8
        RETURN Max(offset, 60)
    ELSE IF distance >= 120 THEN
        offset <- 90 + (distance - 120) * 0.3
        RETURN Min(offset, 120)
    ELSE
        RETURN 90

position_on_edge()
    RETURN (X <= l_board OR X >= r_board OR Y <= u_board
OR Y >= d_board)

GetPredictedShot(myX, myY, enemyX, enemyY, enemyHeading,
enemyVelocity)
    enemyHeading <- enemyHeading *  $\pi$  / 180
    dx <- enemyX - myX
    dy <- enemyY - myY
    distance <- sqrt(dx^2 + dy^2)

    IF distance < 75 THEN bulletPower <- 4.0
    ELSE IF distance < 100 THEN bulletPower <- 3.0
    ELSE IF distance < 125 THEN bulletPower <- 2.0
    ELSE IF distance < 175 THEN bulletPower <- 1.0
    ELSE bulletPower <- 0.5

```

```

bulletSpeed <- 20.0 - 3.0 * bulletPower
vx <- enemyVelocity * cos(enemyHeading)
vy <- enemyVelocity * sin(enemyHeading)

a <- vx^2 + vy^2 - bulletSpeed^2
b <- 2.0 * (dx * vx + dy * vy)
c <- dx^2 + dy^2

discriminant <- b^2 - 4.0 * a * c
t <- 0.0

IF abs(a) > 1e-9 AND discriminant >= 0.0 THEN
    sqrtDisc <- sqrt(discriminant)
    t1 <- (-b + sqrtDisc) / (2.0 * a)
    t2 <- (-b - sqrtDisc) / (2.0 * a)

    IF t1 > 0.0 AND t2 > 0.0 THEN t <- min(t1, t2)
    ELSE IF t1 > 0.0 THEN t <- t1
    ELSE IF t2 > 0.0 THEN t <- t2
    ELSE t <- 0.0
ELSE
    t <- 0.0

predictedX <- enemyX + vx * t
predictedY <- enemyY + vy * t
firingAngle <- atan2(predictedY - myY, predictedX -
myX) * 180 / pi

RETURN [bulletPower, firingAngle]

linear_tracker()
BotInfo <- ReadFromFile("linear_tracker.json")

```

```

Run()

  BodyColor <- (197, 133, 48)
  TurretColor <- (205, 205, 205)
  RadarColor <- (255, 0, 127)
  BulletColor <- (255, 0, 255)
  ScanColor <- (255, 0, 127)
  TracksColor <- (155, 155, 155)
  GunColor <- (175, 175, 175)

  WHILE IsRunning DO
    TurnRadarRight(1000)
    minimal_turn_to_reverse <- minimal_turn_to_reverse
- 1

OnScannedBot(e)
  enemy_bearing <- DirectionTo(e.X, e.Y)
  direction_bearing <- BearingTo(e.X, e.Y)
  gun_bearing <- GunBearingTo(e.X, e.Y)
  radar_bearing <- RadarBearingTo(e.X, e.Y)
  correction <- 0
  distance <- DistanceTo(e.X, e.Y)

  turned <- false

  IF minimal_turn_to_reverse < 0 AND position_on_edge()
THEN
    clockwise <- NOT clockwise
    minimal_turn_to_reverse <- 10
    turned <- true

  IF clockwise THEN
    SetTurnLeft(direction_bearing +
getOffset1(distance))
    SetForward(10000)

```

```

ELSE
    SetTurnLeft(direction_bearing +
getOffset2(distance))
    SetForward(-10000)

IF radar_bearing <= 0 THEN
    correction <- 5
    SetTurnRadarLeft(radar_bearing - correction)
ELSE
    correction <- 5
    SetTurnRadarLeft(radar_bearing + correction)

    shot <- GetPredictedShot(X, Y, e.X, e.Y, e.Direction,
e.Speed)
    bulletPower <- shot[0]
    firingAngle <- shot[1]

    gunTurn <- NormalizeRelativeAngle(firingAngle -
GunDirection)
    SetTurnGunLeft(gunTurn)

IF gunTurn < 2.5 THEN
    Fire(bulletPower)

enemy_hp_bef <- e.Energy
this_hp_bef <- Energy
enemy_id_bef <- e.ScannedBotId

OnHitBot(e)
IF minimal_turn_to_reverse < 5 THEN
    clockwise <- NOT clockwise
    minimal_turn_to_reverse <- 10

```

#### 4.1.3. Spin Shift Bot

```
SpinShiftBot()

    movesBeforeRandomJump <- 300
    moveIncrement <- 1000
    moveCounter <- 0
    random <- Random()

    Start()

Run()

    Set BodyColor <- White
    Set TurretColor <- Red
    Set RadarColor <- Black
    Set ScanColor <- Yellow

    WHILE IsRunning DO
        SetTurnLeft(999999)
        Forward(moveIncrement)
        moveCounter <- moveCounter + 1

        IF moveCounter >= movesBeforeRandomJump THEN
            MoveToRandomCenter()
            moveCounter <- 0

MoveToRandomCenter()

    margin <- 0.8
    targetX <- random.NextDouble() * ArenaWidth * margin
    targetY <- random.NextDouble() * ArenaHeight * margin

    bearing <- BearingTo(targetX, targetY)

    IF bearing >= 0 THEN
```

```

        TurnRight(bearing)
    ELSE
        TurnLeft(-bearing)

    distance <- DistanceTo(targetX, targetY)
    Forward(distance)

    TurnLeft(30)

OnScannedBot(evt)
    Fire(3)

OnHitBot(e)
    bearing <- BearingTo(e.X, e.Y)

    IF -10 < bearing AND bearing < 10 THEN
        Fire(3)

    IF e.IsRammed THEN
        TurnLeft(30)

```

#### 4.1.4. Snake Bot

```

SnakeBot()
    l_board <- 100
    r_board <- 700
    u_board <- 100
    d_board <- 500
    clockwise <- 1
    running_away <- false
    is_shooting <- false

```



```

    enemies <- array of 4 elements (alive, direction,
bearing, distance)

    Start()

GetPredictedShot(myX, myY, enemyX, enemyY, enemyHeading,
enemyVelocity)
    enemyHeading <- enemyHeading *  $\pi$  / 180
    dx <- enemyX - myX
    dy <- enemyY - myY
    distance <- sqrt(dx^2 + dy^2)

    IF distance < 75 THEN bulletPower <- 4.0
    ELSE IF distance < 100 THEN bulletPower <- 3.0
    ELSE IF distance < 125 THEN bulletPower <- 2.0
    ELSE IF distance < 175 THEN bulletPower <- 1.0
    ELSE bulletPower <- 0.5

    bulletSpeed <- 20.0 - 3.0 * bulletPower
    vx <- enemyVelocity * cos(enemyHeading)
    vy <- enemyVelocity * sin(enemyHeading)

    a <- vx^2 + vy^2 - bulletSpeed^2
    b <- 2.0 * (dx * vx + dy * vy)
    c <- dx^2 + dy^2

    discriminant <- b^2 - 4.0 * a * c
    t <- 0.0

    IF abs(a) > 1e-9 AND discriminant >= 0.0 THEN
        sqrtDisc <- sqrt(discriminant)
        t1 <- (-b + sqrtDisc) / (2.0 * a)
        t2 <- (-b - sqrtDisc) / (2.0 * a)

```

```

        IF t1 > 0.0 AND t2 > 0.0 THEN t <- min(t1, t2)
        ELSE IF t1 > 0.0 THEN t <- t1
        ELSE IF t2 > 0.0 THEN t <- t2
        ELSE t <- 0.0
    ELSE
        t <- 0.0

    predictedX <- enemyX + vx * t
    predictedY <- enemyY + vy * t
    firingAngle <- atan2(predictedY - myY, predictedX -
myX) * 180 /  $\pi$ 

    RETURN (bulletPower, firingAngle)

Run()
    BodyColor <- black

    AdjustRadarForBodyTurn <- false
    AdjustGunForBodyTurn <- true
    AdjustRadarForGunTurn <- false

    range <- 1
    cur_enemy_count <- EnemyCount
    running_away <- false
    is_shooting <- false

    WHILE IsRunning DO
        IF NOT running_away AND NOT is_shooting THEN
            SetTurnGunLeft(20)
            SetTurnLeft(MaxTurnRate / range)
            SetForward(20 * clockwise)
            Go()

        IF position_on_edge() AND NOT running_away THEN

```

```

        run_angle <- calcDodgeAngle()
        snakeMove(CalcBearing(run_angle), 100)
        clockwise <- clockwise * -1

    IF EnemyCount < cur_enemy_count THEN
        cur_enemy_count <- EnemyCount
        FOR i <- 0 TO 3 DO
            enemies[i].alive <- false

calcDodgeAngle()
    dodge_x <- 0
    dodge_y <- 0
    total_weight <- 0

    FOR i <- 0 TO 3 DO
        IF enemies[i].alive THEN
            rad <- enemies[i].direction *  $\pi$  / 180
            distance <- enemies[i].distance
            weight <- 100.0 / (distance + 1.0)
            dodge_x <- dodge_x - cos(rad) * weight
            dodge_y <- dodge_y - sin(rad) * weight
            total_weight <- total_weight + weight

    boundary_weight <- 70.0
    battlefield_width <- 800
    battlefield_height <- 600

    dist_to_left <- X
    dist_to_right <- battlefield_width - X
    dist_to_bottom <- Y
    dist_to_top <- battlefield_height - Y

    IF dist_to_left < 100 THEN
        wall_factor <- (100 - dist_to_left) / 100.0

```

```

        wall_weight <- boundary_weight * wall_factor
        dodge_x <- dodge_x + wall_weight
        total_weight <- total_weight + wall_weight

    IF dist_to_right < 100 THEN
        wall_factor <- (100 - dist_to_right) / 100.0
        wall_weight <- boundary_weight * wall_factor
        dodge_x <- dodge_x - wall_weight
        total_weight <- total_weight + wall_weight

    IF dist_to_bottom < 100 THEN
        wall_factor <- (100 - dist_to_bottom) / 100.0
        wall_weight <- boundary_weight * wall_factor
        dodge_y <- dodge_y + wall_weight
        total_weight <- total_weight + wall_weight

    IF dist_to_top < 100 THEN
        wall_factor <- (100 - dist_to_top) / 100.0
        wall_weight <- boundary_weight * wall_factor
        dodge_y <- dodge_y - wall_weight
        total_weight <- total_weight + wall_weight

    IF total_weight > 0 THEN
        dodge_x <- dodge_x / total_weight
        dodge_y <- dodge_y / total_weight

    result_angle <- atan2(dodge_y, dodge_x) * 180 /  $\pi$ 
    RETURN (result_angle + 360) % 360

snakeMove(angle, distance)
    IF NOT is_shooting THEN
        SetTurnGunLeft(10_000)

    running_away <- true

```

```

TurnLeft(angle)

turn_needed <- 90 / MaxTurnRate
i <- 0
WHILE i * MaxSpeed * turn_needed < distance DO
  IF NOT is_shooting THEN
    SetTurnGunLeft(10_000)
    SetTurnLeft(90 * (-1)^i)
    Forward(MaxSpeed * turn_needed)
    i <- i + 1

running_away <- false

OnScannedBot(e)
  distance <- DistanceTo(e.X, e.Y)
  direction <- DirectionTo(e.X, e.Y)
  bearing <- BearingTo(e.X, e.Y)

  PRINT "bot id: " + e.ScannedBotId

  enemies[e.ScannedBotId - 1].alive <- true
  enemies[e.ScannedBotId - 1].direction <- direction
  enemies[e.ScannedBotId - 1].bearing <- bearing
  enemies[e.ScannedBotId - 1].distance <- distance

  IF distance < 200 AND NOT running_away THEN
    run_angle <- calcDodgeAngle()
    PRINT "Run Away Angle: " + run_angle
    snakeMove(CalcBearing(run_angle), 100)

  IF shoot_id % 2 = 1 THEN
    shot <- GetPredictedShot(X, Y, e.X, e.Y,
e.Direction, e.Speed)
    bulletPower <- shot[0]

```

```

        firingAngle <- shot[1]

        gunTurn <- NormalizeRelativeAngle(firingAngle -
GunDirection)

        is_shooting <- true
        IF gunTurn < 2.5 THEN
            SetTurnGunLeft(gunTurn)
            Fire(bulletPower)
            shoot_id <- e.ScannedBotId
            is_shooting <- false

        shoot_id <- shoot_id + 1

position_on_edge()
    RETURN X <= l_board OR X >= r_board OR Y <= u_board OR
Y >= d_board

OnBotDeath(botDeathEvent)
    running_away <- false

```

#### 4.2. Source Code

Fungsi / Bagian Kode	Penjelasan
<pre> int l_board = 100; int r_board = 700; int u_board = 100; int d_board = 500; int minimal_turn_to_reverse = 10; double target_X, target_Y; List&lt;List&lt;ScanData&gt;&gt; scan_data; </pre>	<p>Tahap inisialisasi untuk proses kedepannya.</p> <p>L_board adalah batas kiri.</p> <p>R_board adalah batas kanan.</p> <p>U_board adalah batas atas</p> <p>D_board adalah batas bawah</p> <p>Minimal_turn_to_reverse adalah</p>

<pre>int counter_shot = 0; int counter_hit = 0;</pre>	<p>minimal turn yang dibutuhkan agar bot bisa melakukan reverse direction. Hal ini dilakukan karena jika tidak ada batasan minimal, bot bisa terus-terusan melakukan reverse, sehingga malah menyebabkan bot diam di tempat. Scan_data adalah data hasil scan dari tiap-tiap bot lawan Counter_shot adalah banyaknya tembakan yang dilancarkan Counter_hit adalah banyaknya tembakan yang tepat sasaran Clockwise adalah boolean untuk menunjukkan apakah bot saat ini bergerak searah jarum jam atau tidak</p>
<pre>class ScanData {     public double X;     public double Y;     public double Direction;     public double Speed;     public double Energy;     public int ID;     public int Turn; }  public void UpdateData(double X, double Y, double Direction, double Speed, double Energy, int ID) {     scan_data[ID].Add(new ScanData {X = X, Y = Y, Direction =</pre>	<p>Kelas dan Fungsi untuk melakukan update kepada data. Maksimal data yang disimpan hanya 10 data.</p>

```

Direction, Speed = Speed, Energy =
Energy, ID = ID, Turn = TurnNumber});
    if (scan_data[ID].Count > 10)
    {

scan_data[ID].RemoveAt(0);
        }

        while (scan_data[ID][0].Turn
< TurnNumber - 10) {

scan_data[ID].RemoveAt(0);
        }

        //Console.WriteLine("ID : " +
ID);
    }

scan_data[ID].RemoveAt(0);
    }

    while (scan_data[ID][0].Turn
< TurnNumber - 10) {

scan_data[ID].RemoveAt(0);
        }

        //Console.WriteLine("ID : " +
ID);
    }

```

```

public bool IsTurning(int ID) {
    int len =
scan_data[ID].Count;

```

Fungsi untuk menentukan apakah  
bot dengan id=ID sedang berputar  
atau tidak



<pre>         if (len &lt; 2) {             return false;         }          if (scan_data[ID][len - 1].Direction - scan_data[ID][len - 2].Direction &gt; 0) {             return true;         }          return false;     } </pre>	
<pre> double GetTurnRateDegreesPerTick(int ID) {     if (scan_data[ID].Count &lt; 2)         return 0.0;      int len = scan_data[ID].Count;     double latestDir = scan_data[ID][len - 1].Direction;     double prevDir = scan_data[ID][len - 2].Direction;     double delta = latestDir - prevDir;     double turnRate = delta;      if (turnRate &gt; 180)         turnRate -= 360;     else if (turnRate &lt; -180)         turnRate += 360;      return turnRate; // in degrees/turn } </pre>	<p>Fungsi untuk mendapatkan turn rate (seberapa cepat suatu bot berputar)</p>

<pre> public double getOffset1(double distance) {     if (distance &gt;= 120) {         double offset = 90 - (distance - 120) * 0.2;         return Math.Max(offset, 60);     } else if (distance &lt;= 100) {         double offset = 90 + (100 - distance) * 0.8;         return Math.Min(offset, 120);     } else {         return 90;     } } </pre>	<p>Fungsi untuk mendapatkan offset untuk <i>planetary movement</i> jika arah pergerakan searah jarum jam</p>
<pre> public double getOffset2(double distance) {     if (distance &lt;= 100) {         double offset = 90 - (100 - distance) * 0.8;         return Math.Max(offset, 60);     } else if (distance &gt;= 120) {         double offset = 90 + (distance - 120) * 0.3;         return Math.Min(offset, 120);     } else {         return 90;     } } </pre>	<p>Fungsi untuk mendapatkan offset untuk <i>planetary movement</i> jika arah pergerakan berlawanan arah jarum jam</p>
<pre> public bool position_on_edge() {     return (X &lt;= l_board    X &gt;= r_board    Y &lt;= u_board    Y &gt;= d_board); } </pre>	<p>Fungsi untuk pengecekan apakah posisi bot saat ini berada di tepian papan permainan atau tidak.</p>

}	
<pre> public static double[] GetPredictedShot(     double myX, double myY,     double enemyX, double enemyY,     double enemyHeading, double enemyVelocity ) {     enemyHeading = enemyHeading * Math.PI / 180;     double dx = enemyX - myX;     double dy = enemyY - myY;     double distance = Math.Sqrt(dx * dx + dy * dy);      double bulletPower;     if (distance &lt; 75)     {         bulletPower = 4.0;     }     else if (distance &lt; 100)     {         bulletPower = 3.0;     }     else if (distance &lt; 125)     {         bulletPower = 2.0;     }     else if (distance &lt; 175)     {         bulletPower = 1.0;     }     else </pre>	<p>Fungsi untuk memprediksi pergerakan lawan (asumsi pergerakan lawan linear)</p>

```

{
    bulletPower = 0.5;
}

double bulletSpeed = 20.0 - 3.0 *
bulletPower;

double vx = enemyVelocity *
Math.Cos(enemyHeading);
double vy = enemyVelocity *
Math.Sin(enemyHeading);

double a = vx * vx + vy * vy -
bulletSpeed * bulletSpeed;
double b = 2.0 * (dx * vx + dy * vy);
double c = dx * dx + dy * dy;

double discriminant = b * b - 4.0 * a
* c;
double t = 0.0;

if (Math.Abs(a) > 1e-9 &&
discriminant >= 0.0) {
    double sqrtDisc =
Math.Sqrt(discriminant);
    double t1 = (-b + sqrtDisc) /
(2.0 * a);
    double t2 = (-b - sqrtDisc) /
(2.0 * a);

    if (t1 > 0.0 && t2 > 0.0) {
        t = Math.Min(t1, t2);
    }
    else if (t1 > 0.0) {

```

```

        t = t1;
    }
    else if (t2 > 0.0) {
        t = t2;
    }
    else {
        t = 0.0;
    }
}
else {
    // No valid solution => fallback
to direct aim
    t = 0.0;
}

double predictedX = enemyX + vx * t;
double predictedY = enemyY + vy * t;

// Console.WriteLine("Enemy XY : " +
enemyX + " " + enemyY);
// Console.WriteLine("Predicted XY :
" + predictedX + " " + predictedY);
// Console.WriteLine("Velocity XY : "
+ vx + " " + vy);

double firingAngle =
Math.Atan2(predictedY - myY, predictedX -
myX);

firingAngle = firingAngle * 180 /
Math.PI;

//Console.WriteLine("Predicted shot:
bulletPower={0}, firingAngle={1}",

```

```
bulletPower, firingAngle);

    return new double[] { bulletPower,
firingAngle };
}
```

```
public static double[]
GetTurningPredictiveShot(
    double myX, double myY,
    double enemyX, double enemyY,
    double enemyHeadingDeg,
    double enemySpeed,
    double turnRateDeg,
    double desiredBulletPower
)
{
    double enemyHeading =
enemyHeadingDeg * Math.PI / 180.0;
    double turnRate = turnRateDeg *
Math.PI / 180.0;

    double bulletSpeed = 20.0 - 3.0 *
desiredBulletPower;

    double bestT = 0.0;
    double minDiff = double.MaxValue;
    double maxTime = 50;
    double step = 0.5;

    for (double t = 0.0; t < maxTime;
t += step)
    {
        double[] prediction =
PredictPositionWithArc(
            enemyX, enemyY,
```

Fungsi untuk memprediksi  
pergerakan bot (asumsi pergerakan  
berputar)

```

enemyHeading, enemySpeed, turnRate,
t);

    double predX = prediction[0];
    double predY = prediction[1];

    double dist = Distance(myX,
myY, predX, predY);
    double bulletDist =
bulletSpeed * t;

    double diff = Math.Abs(dist -
bulletDist);
    if (diff < minDiff)
    {
        minDiff = diff;
        bestT = t;
    }
}

double refinedMinDiff = minDiff;
double refinedBestT = bestT;
double refinedStep = step / 10.0;
double refinedStart = Math.Max(0,
bestT - step);
double refinedEnd = bestT + step;

for (double t = refinedStart; t <
refinedEnd; t += refinedStep)
{
    double[] prediction =
PredictPositionWithArc(
        enemyX, enemyY,
        enemyHeading, enemySpeed, turnRate,
        t);

```

```

        double predX = prediction[0];
        double predY = prediction[1];

        double dist = Distance(myX,
myY, predX, predY);
        double bulletDist =
bulletSpeed * t;

        double diff = Math.Abs(dist -
bulletDist);
        if (diff < refinedMinDiff)
        {
            refinedMinDiff = diff;
            refinedBestT = t;
        }
    }

    double[] finalPrediction =
PredictPositionWithArc(
        enemyX, enemyY, enemyHeading,
enemySpeed, turnRate, refinedBestT);

    double angleRadians =
Math.Atan2(finalPrediction[1] - myY,
finalPrediction[0] - myX);
    double firingAngleDeg =
angleRadians * 180.0 / Math.PI;

    return new double[] {
desiredBulletPower, firingAngleDeg };
}

private static double[]
PredictPositionWithArc(
    double x, double y, double
heading, double speed, double

```



```

turnRate, double time)
{
    if (Math.Abs(turnRate) <
0.000001)
    {
        return new double[] {
            x + speed * time *
Math.Cos(heading),
            y + speed * time *
Math.Sin(heading),
            heading
        };
    }

    double radius = speed / turnRate;

    double centerX = x - radius *
Math.Sin(heading);
    double centerY = y + radius *
Math.Cos(heading);

    double newHeading = heading +
turnRate * time;

    double newX = centerX + radius *
Math.Sin(newHeading);
    double newY = centerY - radius *
Math.Cos(newHeading);

    return new double[] { newX, newY,
newHeading };
}

```

```

BodyColor      = Color.FromArgb(197, 133,
48);

```

Memberikan warna kepada bagian-bagian dari tank

<pre> TurretColor    = Color.FromArgb(205, 205, 205); RadarColor      = Color.FromArgb(255, 0, 127); BulletColor     = Color.FromArgb(255, 0, 255); ScanColor       = Color.FromArgb(255, 0, 127); TracksColor     = Color.FromArgb(155, 155, 155); GunColor        = Color.FromArgb(175, 175, 175); </pre>	
<pre> while (IsRunning) {     TurnRadarRight(1000);     minimal_turn_to_reverse--; } </pre>	<p>Fungsi untuk menjalankan bot secara terus menerus. Variabel <code>minimal_turn_to_reverse</code> akan dikurangi pada tiap turn nya.</p>
<pre> double enemy_bearing = DirectionTo(e.X, e.Y); double direction_bearing = BearingTo(e.X, e.Y); double gun_bearing = GunBearingTo(e.X, e.Y); double radar_bearing = RadarBearingTo(e.X, e.Y); double correction = 0; double distance = DistanceTo(e.X, e.Y); </pre>	<p>Inisialisasi untuk proses kedepannya setelah mendapatkan scan dari suatu bot</p>
<pre> // ----- TANK MOVEMENT ----- //     bool turned = false;     if (minimal_turn_to_reverse &lt; 0 &amp;&amp; position_on_edge()) { </pre>	<p>Bagian untuk mengendalikan pergerakan tank : <i>planetary movement</i></p>

<pre>         clockwise = !clockwise;         minimal_turn_to_reverse = 10;         turned = true;     }      if (clockwise) {         SetTurnLeft(direction_bearing + getOffset1(distance));         SetForward(10000);     }     else {         SetTurnLeft(direction_bearing + getOffset2(distance));         SetForward(-10000);     } </pre>	
<pre> // ----- RADAR MOVEMENT -----     if (radar_bearing &lt;= 0) {         correction = 5;  SetTurnRadarLeft(radar_bearing - correction);     }     else {         correction = 5;  SetTurnRadarLeft(radar_bearing + correction);     } </pre>	<p>Bagian untuk mengendalikan pergerakan radar : tracking</p>
<pre> double[] shot = GetPredictedShot(X, Y, e.X, e.Y, e.Direction, e.Speed);     double bulletPower = shot[0]; </pre>	<p>Bagian untuk mengendalikan pergerakan bidikan : linear</p>

```

        double fir// ----- HANDLE GUN
        ----- //
        UpdateData(e.X, e.Y, e.Direction,
        e.Speed, e.Energy, e.ScannedBotId);

        double[] shot;
        if (!IsTurning(e.ScannedBotId)) shot =
        GetPredictedShot(X, Y, e.X, e.Y,
        e.Direction, e.Speed);
        else {
            if (distance < 75) {
                shot =
                GetTurningPredictiveShot(X, Y, e.X, e.Y,
                e.Direction, e.Speed,
                GetTurnRateDegreesPerTick(e.ScannedBotId)
                , 4.0);
            }
            else if (distance < 100) {
                shot =
                GetTurningPredictiveShot(X, Y, e.X, e.Y,
                e.Direction, e.Speed,
                GetTurnRateDegreesPerTick(e.ScannedBotId)
                , 3.0);
            }
            else if (distance < 125) {
                shot =
                GetTurningPredictiveShot(X, Y, e.X, e.Y,
                e.Direction, e.Speed,
                GetTurnRateDegreesPerTick(e.ScannedBotId)
                , 2.0);
            }
            else if (distance < 175) {
                shot =
                GetTurningPredictiveShot(X, Y, e.X, e.Y,
                e.Direction, e.Speed,
                GetTurnRateDegreesPerTick(e.ScannedBotId)
                , 1.0);
            }
        }
    }
}

```

prediction

```

    }
    else {
        shot =
GetTurningPredictiveShot(X, Y, e.X, e.Y,
e.Direction, e.Speed,
GetTurnRateDegreesPerTick(e.ScannedBotId)
, 0.5);
    }
}
double bulletPower = shot[0];
double firingAngle = shot[1];

double gunTurn =
NormalizeRelativeAngle(firingAngle -
GunDirection);
SetTurnGunLeft(gunTurn);

// ----- FIRE ----- //
SetFire(bulletPower);
ingAngle = shot[1];

    double gunTurn =
NormalizeRelativeAngle(firingAngle -
GunDirection);
    SetTurnGunLeft(gunTurn);

    if (gunTurn < 2.5) {
        Fire(bulletPower);
    }

```

```

public override void
OnBulletHit(BulletHitBotEvent
bulletHitBotEvent) {
    counter_hit += 1;
    counter_shot += 1;
}

```

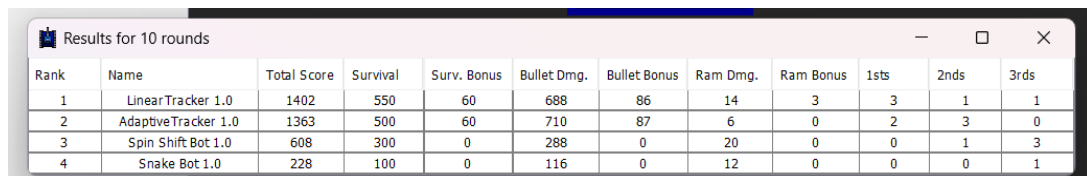
Fungsi untuk menghitung hitrate dari bot.

<pre> }  public override void OnBulletHitWall(BulletHitWallEvent bulletHitWallEvent) {     counter_shot += 1; }  public override void OnBulletHitBullet(BulletHitBulletEvent bulletHitBulletEvent) {     counter_shot += 1; } </pre>	
<pre> public override void OnHitBot(HitBotEvent e) {     if (minimal_turn_to_reverse &lt; 5) {         clockwise = !clockwise;         minimal_turn_to_reverse = 10;     } } </pre>	<p>Fungsi untuk melakukan reverse apabila bot saat ini menabrak bot lain.</p>
<pre> private static double Distance(double x1, double y1, double x2, double y2) {     double dx = x2 - x1;     double dy = y2 - y1;     return Math.Sqrt(dx*dx + dy*dy); } </pre>	<p>Fungsi untuk menentukan jarak dari 2 buah titik : Phytagorean theorem</p>

### 4.3. Testing (Pengujian)

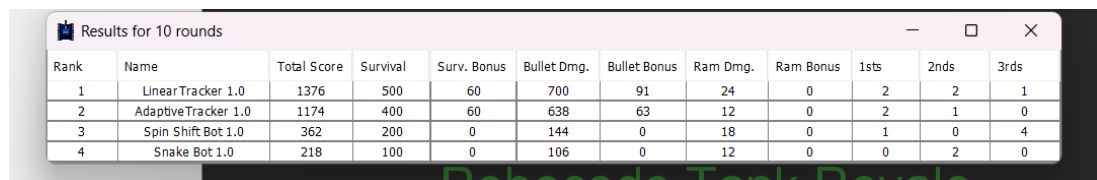
Untuk mengukur performa dan efektivitas bot yang telah dikembangkan, dilakukan proses pengujian dengan format pertarungan langsung antar bot. Setiap pengujian dilakukan dalam skenario **1 vs 1 vs 1 vs 1**, yaitu empat bot saling bertarung dalam satu arena secara bersamaan. Tujuan dari pengujian ini adalah untuk mengevaluasi strategi pergerakan, bidikan, dan radar secara menyeluruh dalam kondisi kompetitif.

Setiap bot akan bertarung melawan tiga bot lainnya dalam satu pertandingan, dan pengujian dilakukan sebanyak **3 ronde** agar diperoleh hasil yang lebih stabil dan tidak dipengaruhi oleh faktor keberuntungan atau posisi awal. Hasil dari pengujian tersebut adalah sebagai berikut :



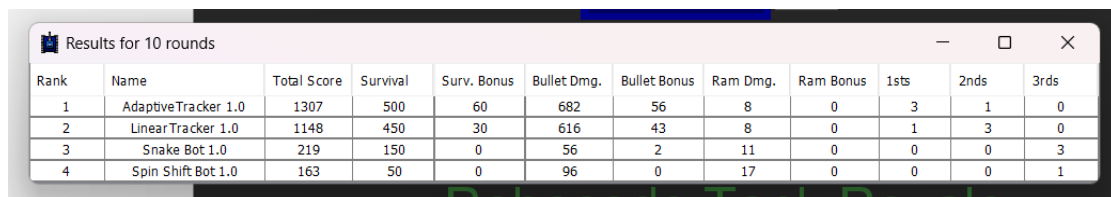
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	LinearTracker 1.0	1402	550	60	688	86	14	3	3	1	1
2	AdaptiveTracker 1.0	1363	500	60	710	87	6	0	2	3	0
3	Spin Shift Bot 1.0	608	300	0	288	0	20	0	0	1	3
4	Snake Bot 1.0	228	100	0	116	0	12	0	0	0	1

Gambar 4.1. Hasil pengujian ronde 1



Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	LinearTracker 1.0	1376	500	60	700	91	24	0	2	2	1
2	AdaptiveTracker 1.0	1174	400	60	638	63	12	0	2	1	0
3	Spin Shift Bot 1.0	362	200	0	144	0	18	0	1	0	4
4	Snake Bot 1.0	218	100	0	106	0	12	0	0	2	0

Gambar 4.2. Hasil pengujian ronde 2



Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	AdaptiveTracker 1.0	1307	500	60	682	56	8	0	3	1	0
2	LinearTracker 1.0	1148	450	30	616	43	8	0	1	3	0
3	Snake Bot 1.0	219	150	0	56	2	11	0	0	0	3
4	Spin Shift Bot 1.0	163	50	0	96	0	17	0	0	0	1

Gambar 4.3. Hasil pengujian ronde 3

#### 4.4. Analisis Pengujian

Seperti yang terlihat pada bagian sebelumnya, bahwa bot *LinearTracker* dan *AdaptiveTracker* sangat mendominasi permainan, dimana bot tersebut selalu berada di posisi nomer pertama atau kedua. Sementara bot lainnya, yaitu *Snake Bot* dan *Spin Shift Bot* berada di posisi nomor ketiga atau keempat.

Bot *AdaptiveTracker* dan *LinearTracker* keduanya memiliki strategi pergerakan tank yang sama persis, strategi pergerakan radar yang juga sama persis, serta strategi pembidikan yang sangat mirip, linear tracker selalu mengasumsikan bahwa tank lawan bergerak secara linear, sementara *AdaptiveTracker* dapat memilih 1 dari 2 kemungkinan, yaitu asumsi bergerak secara linear atau bergerak secara melingkar. Kesuksesan 2 bot tersebut menunjukkan bahwa strategi pergerakan tank dan strategi pergerakan radar yang digunakan merupakan pilihan yang tepat.

Pada *LinearTracker* dan *AdaptiveTracker*, strategi pergerakan tank yang digunakan adalah *planetary movement*, dimana bot akan selalu bergerak secara melingkar mengitari suatu bot lain sembari menjaga jarak dari tembok dan bot yang sedang dijadikan target. Dikarenakan pergerakannya yang selalu melingkar tersebut, mengakibatkan bot lain kesulitan untuk melepaskan tembakan yang tepat sasaran. Selain itu, karena bot tersebut bergerak dengan mengitar bot lain, sangat menambah kemungkinan adanya *stray bullet* yang awalnya ditujukan pada bot tersebut, akan tetapi malah mengenai bot lain.

Di satu sisi, pergerakan radar yang mengunci lawan juga merupakan faktor dari keberhasilan bot tersebut. Hal tersebut dikarenakan apabila pergerakan radar bersifat mengunci, maka semakin sedikit pergerakan radar yang perlu dilakukan, sehingga pergerakan arah bidikan dan radar tidak berbeda jauh, sehingga sangat jarang terjadi adanya peluru *late reaction*.

Salah satu aspek terpenting dalam keberhasilan bot *AdaptiveTracker* dan *LinearTracker* pada percobaan ini adalah bagaimana bot-bot tersebut melakukan prediksi pada gerakan lawan sebelum menembak. Adanya prediksi tersebut sangat



meningkatkan kemampuan bertahan hidup bot dan juga score yang bisa didapatkan. Dimana setiap peluru yang mengenai lawan akan memberikan Energy kepada bot tersebut. Semakin akurat peluru yang ditembakkan, semakin besar peluang bot untuk menjadi pemenang.

Salah satu kejanggalan yang dapat diamati adalah bagaimana LinearTracker lebih konsisten dalam meraih kemenangan jika dibandingkan dengan AdaptiveTracker, yang mana merupakan bot dengan kemampuan prediksi yang jauh lebih unggul, terbukti pada babak penjelasan Analisis Efisiensi dan Efektivitas, bot AdaptiveTracker dapat memberikan 49.3% hit rate pada CrazyBot, serta 72.3% hit rate pada SpinBot. Dibandingkan dengan LinearTracker yang hanya dapat memberikan 37.5% hit rate pada CrazyBot, serta 46.2% hit rate pada SpinBot. Hal tersebut dikarenakan karena kedua bot tersebut selalu menjadi 2 bot yang bertahan sampai akhir. Dimana, ketika hanya 2 bot tersebut yang bertahan hidup, maka bot tersebut akan bergerak secara linear dikarenakan samanya strategi pergerakan tank yang digunakan, yaitu *planetary movement*. Karena kedua bot bergerak secara linear, maka LinearTracker memiliki keuntungan yang lebih tinggi dibandingkan dengan bot AdaptiveTracker, dimana AdaptiveTracker bisa saja menganggap bahwa bot sedang bergerak secara melingkar, padahal sebenarnya selalu bergerak secara linear.

## **BAB 5: Kesimpulan dan Saran**

### **5.1. Kesimpulan**

Dengan ini, kami berhasil membuat bot untuk permainan robocode dengan pendekatan algoritma greedy. Dari tugas ini dapat ditarik kesimpulan bahwa algoritma greedy dapat menjadi alternatif strategi pemecahan masalah yang tepat dalam menentukan solusi optimum dari kondisi sekarang (maksimum/minimum lokal), walaupun solusi yang berupa maksimum/minimum lokal belum tentu merupakan solusi yang paling tepat (maksimum/minimum global). Namun, solusi yang dihasilkan algoritma greedy memiliki optimasi yang cukup baik dalam decision making seperti pada kasus bot di permainan robocode, sebab dalam kebanyakan kasus, solusi yang dihasilkan oleh algoritma greedy cukup mendekati solusi maksimum/minimum global.

### **5.2. Saran**

Saran pengembangan untuk tugas besar ini adalah:

1. Mengerjakan bot tidak mendekati deadline agar bot dapat lebih sempurna.
2. Bereksperimen dengan lebih banyak bot.

### **5.3. Komentar**

Tugas besar ini memiliki banyak batasan sehingga tidak terlalu menarik dikerjakan. Gim yang digunakan memiliki beberapa aspek RNG yang membuat “keberuntungan” menjadi komponen penentu kemenangan. Gim yang menurut penulis laporan akan membuat tugas besar menjadi jauh lebih menarik adalah gim yang sangat memerlukan strategi, misal [Online programming contest and hackathon " CodinGame Spring Challenge 2025"](#). Tentu penulis juga menyadari bahwa menyiapkan gim seperti ini akan menjadi tantangan untuk asisten dan juga batasan Greedy untuk tugas besar ini akan menjadi sulit untuk ditepati.

Komentar Fariz : *"Ne quid exspectes, et nihil te franget."*

## **LAMPIRAN**

Repo: [https://github.com/BoredAngel/Tubes1\\_Atmint-Cabang-DoaAyahRestulbu](https://github.com/BoredAngel/Tubes1_Atmint-Cabang-DoaAyahRestulbu)

Video : <https://linktr.ee/Ryzz17>

## **DAFTAR PUSTAKA**

[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)