

LAPORAN TUGAS BESAR 1

IF2211 STRATEGI ALGORITMA

**Pemanfaatan Algoritma *Greedy* dalam Pembuatan Bot
Permainan Robocode Tank Royale**



Disusun Oleh:

“pikir nanti”

Albertus Christian Poandy

13523077

Grace Evelyn Simon

13523087

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025**

DAFTAR ISI

BAB 1	2
DESKRIPSI TUGAS	2
BAB 2	8
LANDASAN TEORI	8
2.1 Algoritma Greedy	8
2.2 Robocode Tank Royale Starter Pack	10
2.3 Langkah untuk Menjalankan Robocode Tank Royale	10
2.3 Langkah untuk Mengimplementasikan Bot pada Robocode Tank Royale	12
BAB 3	18
APLIKASI STRATEGI GREEDY	18
3.1 Proses Mapping Persoalan Robocode	18
3.2 Eksplorasi 4 Alternatif Solusi	21
3.3 Analisis Efisiensi dan Efektivitas	26
3.4 Hasil Pemilihan Strategi Greedy	29
BAB 4	31
IMPLEMENTASI DAN PENGUJIAN	31
4.1 Implementasi 4 Alternatif Solusi	31
4.2 Penjelasan Implementasi Solusi	43
4.2 Penjelasan Implementasi Alternatif Solusi	46
4.3 Pengujian	52
4.4 Analisis	53
BAB 5	54
KESIMPULAN DAN SARAN	54
5.1 Kesimpulan	54
5.2 Saran	55
LAMPIRAN	57
Tautan Repository Github	57
Tautan Video	57
Hasil Akhir Tugas Besar	57
DAFTAR PUSTAKA	58

DAFTAR GAMBAR

Gambar 1. Bagian Tubuh Tank	5
Gambar 2. Pemindaian Bot	6
Gambar 3. Pemindaian Bot	6
Gambar 4. Konfigurasi Permainan Robocode Tank Royale	11
Gambar 5. Ilustrasi Perhitungan Ancaman di Tiap Titik	45
Gambar 6. Konfigurasi Arah Bot Derik	47
Gambar 7. Konfigurasi Arah Bot Derik	48
Gambar 8. Konfigurasi arah bot Botol	50
Gambar 9. Hasil Pengujian 1 Keempat Bot	52
Gambar 10. Hasil Pengujian 2 Keempat Bot	52
Gambar 11. Hasil Pengujian 3 Keempat Bot	52
Gambar 12. Hasil Pengujian 4 Keempat bot	52

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. Nama Robocode adalah singkatan dari “Robot code,” yang berasal dari versi asli/pertama permainan ini. Robocode Tank Royale adalah evolusi/versi berikutnya dari permainan ini, di mana *bot* dapat berpartisipasi melalui internet/jaringan. Dalam permainan ini, pemain berperan sebagai *programmer bot* dan tidak memiliki kendali langsung atas permainan. Pemain hanya bertugas untuk membuat program yang menentukan logika atau “otak” bot. Program yang dibuat akan berisi instruksi tentang cara *bot* bergerak, mendeteksi *bot* lawan, menembakkan senjatanya, serta bagaimana *bot* bereaksi terhadap berbagai kejadian selama pertempuran. Pada Tugas Besar pertama Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah *bot* yang nantinya akan dipertandingkan satu sama lain. Tentunya mahasiswa harus menggunakan strategi *greedy* dalam membuat *bot* ini.

Komponen-komponen dari permainan ini antara lain:

1. Rounds dan Turns

Pertempuran dapat terdiri dari beberapa *rounds*. Secara default, satu pertempuran berisi 10 *rounds*, di mana setiap *rounds* akan memiliki pemenang dan yang kalah. Setiap *round* dibagi menjadi beberapa *turns*, yang merupakan unit waktu terkecil. Satu *turn* adalah satu ketukan waktu dan satu putaran permainan. Jumlah *turn* dalam satu *round* tergantung pada berapa lama waktu yang dibutuhkan hingga hanya tersisa bot terakhir yang bertahan. Pada setiap *turn*, sebuah bot dapat:

- Menggerakkan bot, memindai musuh, dan menembakkan senjata.
- Bereaksi terhadap peristiwa seperti saat bot terkena peluru atau bertabrakan dengan bot lain atau dinding.

- Perintah untuk bergerak, berputar, memindai, menembak, dan sebagainya dikirim ke server untuk setiap *turn*.

Perlu diperhatikan bahwa API (*Application Programming Interface*) *bot* resmi secara otomatis mengirimkan niat *bot* ke server di balik layar, sehingga Anda tidak perlu mengkhawatirkannya, kecuali jika Anda membuat API *Bot* sendiri. Pada setiap *turn*, *bot* akan secara otomatis menerima informasi terbaru tentang posisinya dan orientasinya di medan perang. Bot juga akan mendapatkan informasi tentang *bot* musuh ketika mereka terdeteksi oleh pemindai. Perlu diketahui bahwa *game engine* yang akan digunakan pada tugas besar ini tidak mengikuti aturan *default* mengenai komponen Round & Turns.

2. Batas Waktu Giliran

Penting untuk dicatat bahwa setiap *bot* memiliki batas waktu untuk setiap *turn* yang disebut *turn timeout*, biasanya antara 30-50 ms (dapat diatur sebagai aturan pertempuran). Ini berarti bahwa *bot* tidak bisa mengambil waktu sebanyak yang mereka inginkan untuk bergerak dan menyelesaikan *turn* saat ini. Setiap kali *turn* baru dimulai, penghitung waktu ulang diatur ulang dan mulai berjalan. Jika batas waktu tercapai dan *bot* tidak mengirimkan pergerakannya untuk *turn* tersebut, maka tidak ada perintah yang dikirim ke server. Akibatnya, *bot* akan melewatkan *turn* tersebut. Jika *bot* melewatkan *turn*, ia tidak akan bisa menyesuaikan gerakannya atau menembakkan senjatanya karena server tidak menerima perintah tepat waktu sebelum *turn* berikutnya dimulai.

3. Energi

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

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

4. Peluru

Semakin banyak energi (daya tembak) yang digunakan untuk menembakkan peluru, semakin berat peluru tersebut dan semakin lambat gerakannya. Namun, peluru yang lebih berat juga menghasilkan lebih banyak kerusakan dan memungkinkan *bot* mendapatkan lebih banyak energi saat mengenai *bot* musuh. Seperti disebutkan sebelumnya, peluru yang lebih berat akan bergerak lebih lambat. Ini berarti akan membutuhkan waktu lebih lama untuk mencapai target, meningkatkan risiko peluru tidak mengenai sasaran. Sebaliknya, peluru yang lebih ringan bergerak lebih cepat, sehingga lebih mudah mengenai target, tetapi peluru ringan tidak memberikan banyak poin energi saat mengenai *bot* musuh.

5. Panas Meriam (*Gun Heat*)

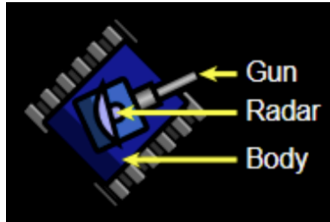
Saat menembakkan peluru, meriam akan menjadi panas. Peluru yang lebih berat menghasilkan lebih banyak panas dibandingkan peluru yang lebih ringan. Ketika meriam terlalu panas, *bot* tidak dapat menembak hingga suhu meriam turun ke nol. Selain itu, meriam juga sudah dalam keadaan panas di awal *round*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

6. Tabrakan

Perlu diperhatikan bahwa *bot* akan menerima kerusakan jika menabrak dinding (batas arena), yang disebut *wall damage*. Hal yang sama juga terjadi jika *bot* bertabrakan dengan *bot* lain. Jika *bot* menabrak *bot* musuh dengan bergerak maju, ini disebut *ramming* (menabrak dengan sengaja), yang akan memberikan sedikit skor tambahan bagi *bot* yang menyerang.

7. Bagian Tubuh Tank

Tubuh *tank* terdiri dari 3 bagian:



Gambar 1. Bagian Tubuh Tank

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

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

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

8. Pergerakan

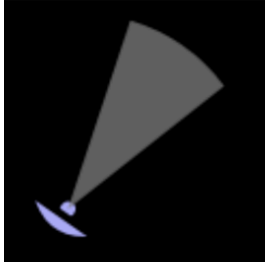
Bot dapat bergerak maju dan mundur hingga kecepatan maksimum. Dibutuhkan beberapa giliran untuk mencapai kecepatan maksimum. *Bot* dapat mengalami percepatan maksimum sebesar 1 unit per giliran dan pengereman dengan perlambatan maksimum 2 unit per giliran. Percepatan dan perlambatan maksimum tidak bergantung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

9. Berbelok

Seperti yang disebutkan sebelumnya, bagian tubuh, *turret* (meriam), dan *radar* dapat berputar secara independen satu sama lain. Jika *turret* atau *radar* tidak diputar, maka keduanya akan mengarah ke arah yang sama dengan tubuh *bot*. Setiap bagian tubuh memiliki kecepatan putar yang berbeda. *Radar* adalah bagian tercepat dan dapat berputar hingga 45 derajat per giliran, yang berarti dapat berputar 360 derajat dalam 8 giliran. *Turret* dan *meriam* dapat berputar hingga 20 derajat per giliran. Bagian paling lambat adalah tubuh *tank*, yang dalam kondisi terbaik dapat berputar hingga 10 derajat per giliran. Namun, ini bergantung pada kecepatan *bot* saat ini. Semakin cepat *bot* bergerak, semakin lambat kemampuannya untuk berbelok. Perlu diperhatikan bahwa tidak ada energi yang dikonsumsi saat *bot* bergerak atau berbelok.

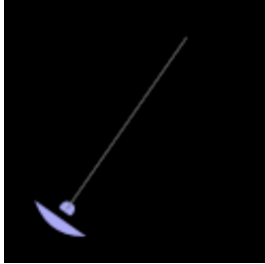
10. Pemindaian

Aspek penting dalam Robocode adalah memindai *bot* musuh menggunakan *radar*. *Radar* dapat mendeteksi *bot* dalam jangkauan hingga 1200 piksel. Musuh yang berada lebih dari 1200 piksel dari *bot* tidak dapat terdeteksi atau dipindai oleh *radar*. Penting untuk diperhatikan bahwa sebuah *bot* hanya dapat memindai *bot* musuh yang berada dalam jangkauan sudut pemindaian (*scan arc*)-nya. Sudut pemindaian ini merupakan “sapuan *radar*” dari arah *radar* sebelumnya ke arah *radar* saat ini dalam satu giliran.



Gambar 2. Pemindaian *Bot*

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 3. Pemindaian *Bot*

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

11. Skor

Pada akhir pertempuran, setiap *bot* akan diranking berdasarkan total skor yang diperoleh masing-masing *bot* selama keseluruhan pertempuran. Tentunya, tujuan utama pada tugas besar ini adalah membuat *bot* yang memberikan skor setinggi mungkin. Berikut adalah rincian komponen skor pada pertempuran:

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

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

Dalam tugas besar ini, mahasiswa diminta untuk bekerja dalam kelompok minimal dua orang dan maksimal tiga orang, dengan lintas kelas dan lintas kampus diperbolehkan. Tujuan dari tugas ini adalah untuk mengimplementasikan algoritma *greedy* pada *bot* permainan Robocode Tank Royale dengan strategi yang dirancang untuk memperoleh skor setinggi mungkin pada akhir pertempuran. Setiap kelompok diwajibkan untuk membuat empat buah *bot* (1 utama dan 3 alternatif) dalam bahasa C# (.net). Strategi *greedy* yang diimplementasikan harus berbeda untuk setiap *bot* yang diimplementasikan dan setiap strategi *greedy* harus menggunakan heuristik yang berbeda. *Bot* yang dibuat tidak boleh sama dengan sampel yang diberikan sebagai contoh, baik dari *starter pack* maupun dari *repository engine* asli. Strategi *greedy* yang dibuat harus dijelaskan dan ditulis secara eksplisit pada laporan. Program harus mengandung komentar yang jelas, dan untuk setiap strategi *greedy* yang disebutkan, harus dilengkapi dengan kode sumber yang dibuat.

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 diambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip “*take what you can get now!*”) 2. dan “berharap” bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

Terdapat beberapa elemen yang diperlukan agar proses pemecahan masalah secara *greedy* lebih mudah untuk dilakukan, yakni sebagai berikut:

1. Himpunan kandidat (C): berisi kandidat yang akan dipilih pada setiap langkah (misal: simpul/sisi di dalam graf, job, task, koin, benda, karakter, dsb)
2. Himpunan solusi (S): berisi kandidat yang sudah dipilih untuk menjadi solusi
3. Fungsi solusi: fungsi yang digunakan untuk menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi
4. Fungsi seleksi (*selection function*): fungsi untuk memilih kandidat berdasarkan strategi *greedy* tertentu yang bersifat heuristik
5. Fungsi kelayakan (*feasible*): fungsi untuk memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak)
6. Fungsi obyektif: fungsi untuk memaksimumkan atau meminimumkan kandidat yang dipilih

Dengan menggunakan elemen-elemen tersebut, dapat dikatakan bahwa algoritma *greedy* melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C; yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu S menyatakan suatu solusi dan S dioptimisasi oleh fungsi obyektif.

Namun, perlu diingat bahwa solusi optimum global yang dihasilkan oleh algoritma *greedy* belum tentu merupakan solusi yang terbaik karena bisa saja solusi tersebut hanya solusi *sub-optimum* dari permasalahan. Terdapat dua buah alasan mengapa hal tersebut dapat terjadi, yakni sebagai berikut:

1. Algoritma *greedy* tidak beroperasi secara menyeluruh terhadap semua kemungkinan solusi yang ada (sebagaimana pada metode *exhaustive search*)
2. Terdapat beberapa fungsi seleksi yang berbeda, sehingga kita harus memilih fungsi yang tepat jika kita ingin algoritma menghasilkan solusi optimal

Apabila solusi terbaik mutlak tidak terlalu diperlukan, algoritma *greedy* dapat digunakan untuk menghasilkan solusi hampiran (*approximation*), dibandingkan menggunakan algoritma lain yang kebutuhan waktunya eksponensial untuk menghasilkan solusi eksak. Namun, apabila algoritma *greedy* dapat menghasilkan solusi optimal, keoptimalan tersebut harus dapat dibuktikan secara sistematis. Berikut beberapa contoh persoalan yang dapat diselesaikan dengan algoritma *greedy*:

1. Persoalan penukaran uang (*coin exchange problem*)
2. Persoalan memilih aktivitas (*activity selection problem*)
3. Minimisasi waktu di dalam sistem
4. Persoalan knapsack (*knapsack problem*)
5. Penjadwalan job dengan tenggat waktu (*job scheduling with deadlines*)
6. Pohon merentang minimum (*minimum spanning tree*)
7. Lintasan terpendek (*shortest path*)
8. Kode Huffman (*Huffman code*)
9. Pecahan Mesir (*Egyptian fraction*)

2.2 Robocode Tank Royale *Starter Pack*

Starter pack diperlukan dalam membangun algoritma bot yang akan digunakan dalam permainan Robocode Tank Royale. Bot Starter Pack ini dibuat menggunakan bahasa pemrograman C#, dan sudah dimodifikasi dengan sejumlah fitur baru. Berikut adalah beberapa penjelasan tentang Starter Pack terbaru:

1. Theme GUI:

Tampilan antarmuka pengguna (GUI) telah diubah menjadi *light theme*. Ini memudahkan pengguna untuk melihat dan berinteraksi dengan *game engine* dalam keadaan terang, memberikan kenyamanan visual selama permainan.

2. Skor & Energi:

Setiap *bot* akan ditampilkan di samping arena permainan dengan informasi tentang skor dan energi mereka. Hal ini memudahkan pemain untuk memonitor status dan performa *bot* mereka selama pertarungan, memungkinkan perencanaan taktis yang lebih baik.

3. Turn Limit:

Durasi pertarungan tidak lagi bergantung pada banyaknya ronde. Permainan akan berakhir setelah jumlah *turn* mencapai batas tertentu. Setelah itu, pemenang akan ditampilkan secara otomatis.

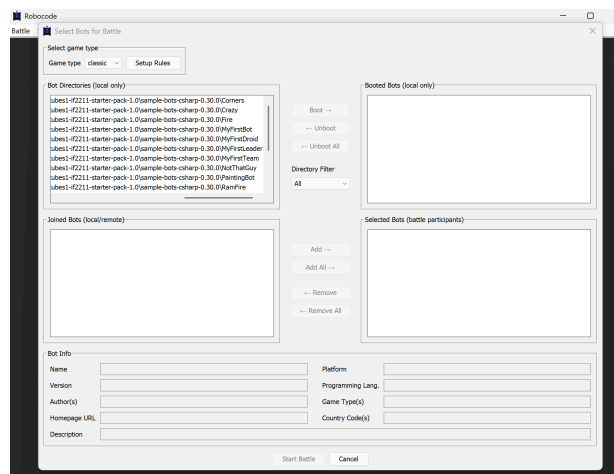
Untuk Source Code *game engine* dan *template* bot telah disediakan pada tautan berikut: [tubes1-if2211-starter-pack](#) . Panduan mengenai cara menjalankan *game engine*, membuat *bot*, dan melihat referensi API dapat dilihat pada tautan berikut: [Get Started With Robocode](#) .

2.3 Langkah untuk Menjalankan Robocode Tank Royale

Untuk menjalankan permainan Robocode Tank Royale, berikut adalah langkah-langkah yang perlu dilakukan:

1. *Download asset* dari *release* terbaru pada link [tubes1-if2211-starter-pack](#) , kemudian download jar “robocode-tankroyale-gui-0.30.0.jar” yang merupakan *game engine* hasil modifikasi asisten.
2. *Download* dan ekstrak “TemplateBot.zip” dan ekstrak *source code* yang berisi *sample bots* yang disediakan Robocode dan *source code game engine* yang dapat di-build ulang menggunakan *gradle* apabila dibutuhkan.
3. Jalankan file .jar aplikasi GUI dengan mengetikkan *command* berikut pada terminal:

```
java -jar robocode-tankroyale-gui-0.30.0.jar
```
4. *Setup* konfigurasi *booter* dengan menekan tombol “Config” → “Bot Root Directories”, kemudian masukkan *directory* yang berisi folder-folder *bot* yang ingin digunakan.
5. Jalankan *battle* dengan cara menekan tombol “Battle” → “Start Battle”. Akan muncul panel konfigurasi permainan seperti berikut:



Gambar 4. Konfigurasi Permainan Robocode Tank Royale

6. Lakukan *boot* pada *bot* yang ingin dimainkan. *Bot* yang telah dipilih untuk di-*boot* akan muncul pada kotak kanan atas. *Bot* yang berhasil di-*boot* akan muncul pada kotak kiri bawah.

7. Tambahkan *bot* ke dalam permainan dengan cara memilih *bot* yang ingin ditambahkan dari kotak kiri bawah. *Bot* yang telah ditambahkan akan otomatis muncul pada kotak kanan bawah. Permainan kemudian dapat dimulai dengan menekan tombol “Start Battle”.

2.3 Langkah untuk Mengimplementasikan Bot pada Robocode Tank Royale

Berikut adalah langkah-langkah untuk mengimplementasikan *bot* agar dapat dijalankan dalam Robocode Tank Royale:

1. Buatlah folder baru yang bernama *bot* Anda (e.g. “BotTemplate”)
2. Buatlah file json dengan fields sebagai berikut:

```
{
  "name": "«nama kelompok»",
  "version": "1.0",
  "authors": [ "«anggota 1»", "«anggota 2»", "«anggota 3»" ],
  "description": "Deskripsi strategi greedy yang digunakan",
  "homepage": "«...»",
  "countryCodes": [ "«enter your country code, e.g. id»" ],
  "platform": "«enter programming platform, e.g. Java or .Net»",
  "programmingLang": "C#"
}
```

Fields “name”, “version”, dan “authors” wajib terisi, sedangkan *fields* lainnya bersifat opsional. Isi *field name* dengan nama kelompok dan *authors* dengan anggota kelompok. Pastikan string *authors* masing-masing tidak lebih panjang dari 20 karakter.

3. Buatlah file source code C# untuk bot Anda (e.g. “BotTemplate.cs”)
4. Susun source code bot Anda dengan kerangka minimal sebagai berikut:

```
using Robocode.TankRoyale.BotApi;
using Robocode.TankRoyale.BotApi.Events;
```

```

public class BotTemplate : Bot
{
    // The main method starts our bot
    static void Main(string[] args)
    {
        new BotTemplate().Start();
    }
    // Constructor, which loads the bot config file
    BotTemplate() : base(BotInfo.FromFile("BotTemplate.json")) { }
    // Called when a new round is started -> initialize and do some movement
    public override void Run()
    {
        BodyColor = Color.FromArgb(0x00, 0x00, 0x00);
        TurretColor = Color.FromArgb(0x00, 0x00, 0x00);
        RadarColor = Color.FromArgb(0x00, 0x00, 0x00);
        BulletColor = Color.FromArgb(0x00, 0x00, 0x00);
        ScanColor = Color.FromArgb(0x00, 0x00, 0x00);
        TracksColor = Color.FromArgb(0x00, 0x00, 0x00);
        GunColor = Color.FromArgb(0x00, 0x00, 0x00);
        // Repeat while the bot is running
        while (IsRunning)
        {
            // Write your bot logic
        }
    }
}

```

Contoh bot yang sudah memiliki fungsi adalah sebagai berikut:

```

using System.Drawing;

```

```

using Robocode.TankRoyale.BotApi;
using Robocode.TankRoyale.BotApi.Events;
public class BotTemplate : Bot
{
    // The main method starts our bot
    static void Main(string[] args)
    {
        new BotTemplate().Start();
    }
    // Constructor, which loads the bot config file
    BotTemplate() : base(BotInfo.FromFile("BotTemplate.json")) { }
    // Called when a new round is started -> initialize and do some movement
    public override void Run()
    {
        BodyColor = Color.FromArgb(0xFF, 0x8C, 0x00); // Dark Orange
        TurretColor = Color.FromArgb(0xFF, 0xA5, 0x00); // Orange
        RadarColor = Color.FromArgb(0xFF, 0xD7, 0x00); // Gold
        BulletColor = Color.FromArgb(0xFF, 0x45, 0x00); // Orange-Red
        ScanColor = Color.FromArgb(0xFF, 0xFF, 0x00); // Bright Yellow
        TracksColor = Color.FromArgb(0x99, 0x33, 0x00); // Dark
        Brownish-Orange
        GunColor = Color.FromArgb(0xCC, 0x55, 0x00); // Medium Orange
        // Repeat while the bot is running
        while (IsRunning)
        {
            Forward(100);
            TurnGunRight(360);
            Back(100);
            TurnGunRight(360);
        }
    }
}

```



```

}
}
// We saw another bot -> fire!
public override void OnScannedBot(ScannedBotEvent evt)
{
    Fire(1);
}
// We were hit by a bullet -> turn perpendicular to the bullet
public override void OnHitByBullet(HitByBulletEvent evt)
{
    // Calculate the bearing to the direction of the bullet
    var bearing = CalcBearing(evt.Bullet.Direction);
    // Turn 90 degrees to the bullet direction based on the bearing
    TurnLeft(90 - bearing);
}
}

```

5. Buatlah file .csproj yang bernama bot Anda (e.g. "BotTemplate.csproj"). File .csproj harus memiliki isi sebagai berikut:

```

<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <RootNamespace>BotTemplate</RootNamespace>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <LangVersion>10.0</LangVersion>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Robocode.TankRoyale.BotApi"
      Version="0.30.0"/>
  </ItemGroup>

```

</Project>

Perhatikan syarat fields file .csproj berikut:

- RootNamespace berisi nama Class dari bot Anda (e.g. BotTemplate).
- TargetFramework berisi versi .Net version yang diperlukan bot untuk dijalankan, yaitu versi .Net Anda. Anda dapat melihat versi .Net (dotnet). Anda dengan menjalankan command “dotnet --version”
- LangVersion berisi versi bahasa C# untuk menjalankan bot Anda
- PackageReference berisi versi Robocode.TankRoyale.BotApi yang digunakan. Untuk Tugas Besar ini, versi yang digunakan adalah “0.30.0”

6. Buatlah file .cmd dan .sh yang bernama bot Anda (e.g. “BotTemplate.cmd” dan “BotTemplate.sh”). Contoh isi file .cmd dan .sh sebagai berikut:

BotTemplate.cmd

```
@echo off
REM TemplateBot.cmd - Run the bot in development or release mode
REM Set MODE=dev for development (default, always rebuilds)
REM Set MODE=release for release (only runs if bin exists)
set MODE=dev
if "%MODE%"=="dev" (
    REM Development mode: always clean, build, and run
    rmdir /s /q bin obj >nul 2>&1
    dotnet build >nul
    dotnet run --no-build >nul
) else if "%MODE%"=="release" (
    REM Release mode: no rebuild if bin exists

    if exist bin\ (
        dotnet run --no-build >nul
```

```

) else (
dotnet build >nul
dotnet run --no-build >nul
)
) else (
echo Error: Invalid MODE value. Use "dev" or "release".
)

```

BotTemplate.sh

```

#!/bin/sh
# This script runs the bot in development mode by default.
# Development Mode (default): Always cleans, rebuilds, and runs.
# Release Mode (commented out): Runs without rebuilding.
# Development mode: always rebuild
rm -rf bin obj
dotnet build
dotnet run --no-build
# Uncomment below for release mode (runs without rebuilding)
# if [ -d "bin" ]; then
# dotnet run --no-build
# else
# dotnet build
# dotnet run --no-build
# fi

```

BAB 3

APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping* Persoalan Robocode

Dalam permainan Robocode Tank Royale, *bot* dikendalikan oleh algoritma yang dapat dirancang menggunakan strategi *greedy*. Algoritma *greedy* bekerja dengan memilih aksi yang paling menguntungkan pada setiap langkah berdasarkan keadaan permainan saat itu, tanpa mempertimbangkan konsekuensi jangka panjang. Strategi ini cocok untuk situasi di mana keputusan lokal (langkah demi langkah) dapat menghasilkan hasil yang optimal secara keseluruhan. Berikut adalah komponen-komponen utama dari algoritma *greedy* yang diterapkan dalam Robocode Tank Royale:

1. Himpunan Kandidat

Himpunan kandidat adalah kumpulan semua aksi atau pergerakan yang dapat dipilih oleh bot dalam setiap langkah permainan. Dalam konteks Robocode Tank Royale, himpunan kandidat mencakup berbagai opsi yang bisa dilakukan oleh bot selama bertarung:

Bergerak:

- Maju (ke arah tertentu)
- Mundur (menghindari musuh atau rintangan)
- Berputar (untuk menghindari tembakan musuh atau mencari posisi yang lebih baik)

Menembak:

- Menembak musuh yang terdeteksi
- Memilih daya tembak (*firepower*) yang sesuai berdasarkan jarak ke musuh

Memindai Musuh:

- Mengarahkan radar untuk memindai musuh di sekitar arena permainan

2. Himpunan Solusi

Himpunan solusi merupakan himpunan dari semua pilihan aksi yang valid yang bisa dilakukan oleh *bot*. Pilihan-pilihan ini terbatas pada himpunan kandidat, namun hanya yang memenuhi kriteria tertentu yang dianggap sebagai solusi yang sah. Contoh solusi yang valid:

- Tembak dengan kekuatan 1 jika musuh dekat
- Maju 100 unit jika tidak ada musuh dalam jarak tembak
- Berputar 45 derajat jika ada tembok di dekat *bot*
- Mundur jika mendeteksi musuh di belakang

3. Fungsi Solusi

Fungsi solusi merupakan fungsi yang mengevaluasi setiap solusi yang dihasilkan dari himpunan kandidat dan memetakan keputusan tersebut. Dalam hal ini, fungsi solusi membantu *bot* memilih mana aksi yang paling sesuai dengan kondisi saat ini. Beberapa faktor yang dapat digunakan dalam fungsi solusi meliputi:

- Jarak ke musuh: Pilih tembakan dengan *firepower* yang sesuai berdasarkan jarak ke musuh
- Keamanan: Pilih untuk bergerak jika ada tembok atau *bot* lain yang menghalangi

Contoh fungsi solusi:

- Jika musuh dalam jarak tembak, tembak dengan *firepower* 1
- Jika ada tembok, pilih untuk berputar ke arah yang aman
- Jika *bot* jauh dari musuh, pilih untuk bergerak maju secara acak

4. Fungsi Seleksi

Fungsi seleksi adalah fungsi yang memilih aksi terbaik berdasarkan kondisi saat ini. Fungsi ini akan memilih salah satu solusi terbaik dari himpunan solusi berdasarkan aturan tertentu. Dalam hal ini, fungsi seleksi memilih aksi terbaik yang dapat dilakukan *bot*

dengan tujuan untuk memaksimalkan peluang meraih kemenangan (poin). Contoh fungsi seleksi:

- Jika musuh dalam jangkauan tembak, pilih aksi menembak dengan kekuatan tertinggi yang aman
- Jika *bot* dikelilingi musuh atau tembok, pilih untuk berputar untuk menghindari serangan
- Jika *bot* jauh dari musuh, pilih untuk bergerak maju secara acak atau memindai musuh

5. Fungsi Kelayakan

Fungsi kelayakan berfungsi untuk memeriksa apakah solusi yang dipilih valid dan memungkinkan untuk dieksekusi. Fungsi ini penting dalam algoritma *greedy* untuk memastikan bahwa solusi yang dipilih tidak menyebabkan *bot* bergerak ke tempat yang membahayakan atau memilih aksi yang tidak memungkinkan. Contoh fungsi kelayakan:

Contoh:

- Cek apakah *bot* bisa bergerak maju tanpa menabrak tembok
- Cek apakah *bot* bisa menembak tanpa kehabisan energi
- Cek apakah *bot* berada dalam jangkauan radar untuk mendeteksi musuh

6. Fungsi Obyektif

Fungsi obyektif adalah fungsi yang mengukur kinerja solusi berdasarkan tujuan permainan, yaitu untuk memenangkan pertandingan. Fungsi ini digunakan untuk mengevaluasi keuntungan atau kerugian dari setiap langkah yang diambil oleh *bot*. Beberapa contoh tujuan yang dapat diukur oleh fungsi obyektif:

- Memaksimalkan *damage* yang diberikan ke musuh: *Bot* harus memilih untuk menembak atau bergerak ke posisi yang memudahkan serangan
- Minimalkan kerusakan pada *bot*: Pilih untuk bergerak atau berputar untuk menghindari tembakan musuh atau serangan

- Maksimalkan waktu bertahan hidup: Pilih aksi yang meningkatkan kesempatan *bot* untuk bertahan lebih lama di arena

Fungsi objektif dapat berbentuk skor atau nilai yang dihitung berdasarkan kinerja *bot* dalam berbagai parameter permainan, misalnya:

- Skor bertahan hidup
- Jumlah tembakan yang tepat

3.2 Eksplorasi 4 Alternatif Solusi

Terdapat berbagai alternatif solusi yang dapat diterapkan untuk memenangkan permainan Robocode Tank Royale. Berikut merupakan alternatif solusi yang diciptakan oleh kelompok kami dengan tujuan untuk mendapatkan poin tertinggi dalam permainan Robocode Tank Royale.

1. Alternatif Solusi 1 – Derik

Derik adalah *bot* berbasis algoritma *greedy* dengan strategi utama melakukan *locking* pada musuh yang berhasil terdeteksi melalui *radar*. Setelah target dideteksi, Derik akan menjaga jarak tertentu dengan lawan, baik dengan mendekat (jika jarak terlalu jauh) ataupun menjauh (jika jarak terlalu dekat) untuk mendapatkan posisi yang lebih menguntungkan. Selama permainan berlangsung, Derik akan bergerak mengitari (memutari) musuh yang telah di-*lock* nya dalam pola osilasi yang acak. Gerakan memutar dan osilasi secara acak ini bertujuan untuk mempersulit prediksi posisi oleh *bot* lawan sekaligus menjaga akurasi tembakan Derik terhadap musuh yang telah dikunci. Pendekatan ini bersifat cukup agresif dengan fokus pada satu target dalam satu waktu, sesuai dengan prinsip *greedy*, yaitu memilih tindakan terbaik berdasarkan kondisi saat ini tanpa mempertimbangkan langkah jangka panjang dan tanpa mempertimbangkan keadaan *bot* lainnya.

Pemetaan elemen *greedy*:

- Himpunan kandidat:
Pergerakan → maju mendekati musuh, mundur menjauhi musuh, berputar ke kiri/kanan, bersilasi, bergerak secara acak (*random*).

Penembakan → pengaturan *firepower* berdasarkan jarak, *linear targeting*, *circular targeting*, prediktif berdasarkan kecepatan dan arah *bot* musuh.

Radar → pemindaian terus-menerus, *radar lock* apabila mendeteksi musuh.

Strategi bertahan → menjaga jarak aman dari *bot-bot* musuh, menghindari *wall damage*, menargetkan satu musuh untuk mengurangi jumlah *bot* lawan.

- Himpunan solusi: Kumpulan pergerakan dan strategi yang memenuhi kriteria/valid untuk dipilih.
- Fungsi solusi: Jika musuh terdeteksi, evaluasi musuh untuk memutuskan apakah akan menembak atau tidak. Jika mendekati tembok, evaluasi jarak ke tembok untuk memutuskan apakah akan berbalik arah atau berputar. Jika musuh terlalu dekat, evaluasi jarak ke musuh untuk memutuskan apakah akan menjauh atau mendekat. Jika tidak ada musuh, evaluasi posisi saat ini untuk memutuskan arah selanjutnya.
- Fungsi seleksi: Jika ada *bot* yang terdeteksi radar, dekati/jauhi hingga mencapai jarak tertentu yang optimal (pada implementasi ditetapkan sebesar 25 *pixel*), lalu lakukan gerakan memutar *bot* musuh sambil menembak.
- Fungsi kelayakan: Cek apakah *bot* bisa bergerak tanpa menabrak tembok. Cek apakah *bot* bisa menembak tanpa kehabisan energi (memiliki cukup energi). Cek apakah *bot* berada dalam jangkauan radar untuk mendeteksi musuh dan mampu memindai area tanpa terhalang.
- Fungsi objektif: Jumlah poin yang didapatkan maksimum dan *bot* mampu bertahan hidup dengan baik.

2. Alternatif Solusi 2 - Botol

Botol adalah *bot* berbasis algoritma *greedy* dengan strategi yang lebih agresif, yaitu melakukan *locking* dan *ramming* pada musuh yang berhasil terdeteksi melalui *radar*. Setelah target dideteksi, Botol akan secara langsung bergerak mendekat dan menabrak (*ramming*) target selagi terus menembakkan peluru. Strategi ini bertujuan untuk memberikan tekanan yang besar kepada *bot* lawan dengan mendekat secara cepat dan memaksakan pertempuran jarak dekat. Pendekatan ini adalah pendekatan “*high risk high*

reward” yang lebih beresiko, tetapi dalam kondisi pertempuran yang tepat, Botol bisa menjadi *bot* yang sangat kuat dan mendominasi.

Pemetaan elemen greedy:

- Himpunan kandidat:
 - Pergerakan → maju mendekati musuh, mundur menjauhi musuh, berputar ke kiri/kanan, bersilasi, bergerak secara acak (*random*).
 - Penembakan → pengaturan *firepower* berdasarkan jarak, *linear targeting*, *circular targeting*, prediktif berdasarkan kecepatan dan arah *bot* musuh.
 - Radar → pemindaian terus-menerus, *radar lock* apabila mendeteksi musuh.
 - Strategi bertahan → menjaga jarak aman dari *bot-bot* musuh, menghindari *wall damage*, menargetkan satu musuh untuk mengurangi jumlah *bot* lawan.
- Himpunan solusi: Kumpulan pergerakan dan strategi yang memenuhi kriteria/valid untuk dipilih.
- Fungsi solusi: Jika musuh terdeteksi, evaluasi musuh untuk memutuskan apakah akan menembak atau tidak. Jika mendekati tembok, evaluasi jarak ke tembok untuk memutuskan apakah akan berbalik arah atau berputar. Jika musuh terlalu dekat, evaluasi jarak ke musuh untuk memutuskan apakah akan menjauh atau mendekat. Jika tidak ada musuh, evaluasi posisi saat ini untuk memutuskan arah selanjutnya.
- Fungsi seleksi: Jika ada bot yang terdeteksi radar, segera bergerak mendekat dan terus maju hingga menabrak (*ramming*) bot lawan sambil menembakkan peluru.
- Fungsi solusi: Cek apakah *bot* bisa bergerak tanpa menabrak tembok. Cek apakah *bot* bisa menembak tanpa kehabisan energi (memiliki cukup energi). Cek apakah *bot* berada dalam jangkauan radar untuk mendeteksi musuh dan mampu memindai area tanpa terhalang.
- Fungsi objektif: Jumlah poin yang didapatkan maksimum dan *bot* mampu bertahan hidup dengan baik.

3. Alternatif Solusi 3 – NotThatGuy

NotThatGuy adalah *bot* yang menggunakan algoritma *greedy* dengan strategi utama berupa arah pergerakan yang acak dengan tetap menghindari tembok untuk mencegah

wall damage. Jika mendekati tembok, *bot* akan berbalik arah. Jika mendeteksi musuh terlalu dekat, *bot* akan menjauh. Untuk penembakan, *bot* NotThatGuy menerapkan penembakan prediktif. *Bot* akan menembak musuh yang terdeteksi saat ini dengan memprediksi pergerakan musuh selanjutnya berdasarkan kecepatan dan arahnya. Secara *default*, *radar* akan terus berputar untuk memindai area sekitar. Namun, ketika musuh terdeteksi, *radar* akan mengunci musuh tersebut. Jika musuh sudah tidak terdeteksi oleh *radar*, *radar* kembali ke mode pemindaian terus-menerus.

Pemetaan elemen *greedy*:

- Himpunan kandidat:
 - Pergerakan → maju mendekati musuh, mundur menjauhi musuh, berputar ke kiri/kanan, bersilasi, bergerak secara acak (*random*).
 - Penembakan → pengaturan *firepower* berdasarkan jarak, *linear targeting*, *circular targeting*, prediktif berdasarkan kecepatan dan arah *bot* musuh.
 - Radar* → pemindaian terus-menerus, *radar lock* apabila mendeteksi musuh.
 - Strategi bertahan → menjaga jarak aman dari *bot-bot* musuh, menghindari *wall damage*, menargetkan satu musuh untuk mengurangi jumlah *bot* lawan.
- Himpunan solusi: Kumpulan pergerakan dan strategi yang memenuhi kriteria/valid untuk dipilih.
- Fungsi solusi: Jika musuh terdeteksi, evaluasi musuh untuk memutuskan apakah akan menembak atau tidak. Jika mendekati tembok, evaluasi jarak ke tembok untuk memutuskan apakah akan berbalik arah atau berputar. Jika musuh terlalu dekat, evaluasi jarak ke musuh untuk memutuskan apakah akan menjauh atau mendekat. Jika tidak ada musuh, evaluasi posisi saat ini untuk memutuskan arah selanjutnya.
- Fungsi seleksi: Jika musuh terdeteksi dan dalam jarak optimal, pilih untuk menembak dengan *firepower* yang sesuai. Jika mendekati tembok, pilih untuk berbalik arah. Jika musuh terlalu dekat, pilih untuk menjauhi. Jika tidak ada musuh, pilih untuk bergerak secara acak.
- Fungsi kelayakan: Cek apakah *bot* bisa bergerak tanpa menabrak tembok. Cek apakah *bot* bisa menembak tanpa kehabisan energi (memiliki cukup energi). Cek

apakah *bot* berada dalam jangkauan radar untuk mendeteksi musuh dan mampu memindai area tanpa terhalang.

- Fungsi objektif: Jumlah poin yang didapatkan maksimum dan *bot* mampu bertahan hidup dengan baik.

4. Alternatif Solusi 4 – Emo

Emo adalah *bot* yang berfokus pada menghindari ancaman dengan cara mencari posisi *bot* yang paling sedikit kemungkinan ditembak. *Bot* akan memilih posisi yang paling aman dengan meminimalkan tingkat ancaman (*threatLevel*) dari musuh. Selain itu, *bot* Emo menembak musuh dengan memprediksi kemana *bot* musuh akan bergerak saat peluru tiba.

Pemetaan elemen *greedy*:

- Himpunan kandidat
Pergerakan → maju mendekati musuh, mundur menjauhi musuh, berputar ke kiri/kanan, bersilasi, bergerak secara acak (*random*).
Penembakan → pengaturan *firepower* berdasarkan jarak, *linear targeting*, *circular targeting*, prediktif berdasarkan kecepatan dan arah *bot* musuh.
Radar → pemindaian terus-menerus, *radar lock* apabila mendeteksi musuh.
Strategi bertahan → menjaga jarak aman dari *bot-bot* musuh, menghindari *wall damage*, menargetkan satu musuh untuk mengurangi jumlah *bot* lawan.
- Himpunan solusi: Kumpulan pergerakan dan strategi yang memenuhi kriteria/valid untuk dipilih.
- Fungsi solusi: Jika musuh terdeteksi, evaluasi musuh untuk memutuskan apakah akan menembak atau tidak. Jika mendekati tembok, evaluasi jarak ke tembok untuk memutuskan apakah akan berbalik arah. Jika musuh terlalu dekat, evaluasi jarak ke musuh untuk memutuskan apakah akan menjauh atau mendekat. Jika tidak ada musuh, evaluasi posisi saat ini untuk memutuskan arah selanjutnya.
- Fungsi seleksi: Jika musuh terdeteksi dan dalam jarak optimal, pilih untuk menembak. Pilih posisi dengan ancaman terendah dan bergerak ke posisi tersebut. Tembakkan diatur berdasarkan jarak *bot* Emo ke *bot* musuh.

- Fungsi kelayakan: Cek apakah *bot* bisa bergerak maju tanpa menabrak tembok. Cek apakah *bot* bisa menembak tanpa kehabisan energi. Cek apakah *bot* berada dalam jangkauan radar untuk mendeteksi musuh.
- Fungsi objektif: Jumlah poin yang didapatkan maksimum dan *bot* mampu bertahan hidup dengan baik.

3.3 Analisis Efisiensi dan Efektivitas

Dari segi efisiensi, tidak ada algoritma yang membutuhkan proses komputasi yang kompleks pada setiap pendekatan. Pada tiap *turn*, rata-rata kompleksitas algoritma yang digunakan adalah $O(1)$, yaitu hanya melakukan kalkulasi matematis ataupun pemanggilan API, tanpa adanya proses perulangan. Hal ini juga karena pada *bot* Derik, Botol, dan NotThatGuy, diterapkan pendekatan *locking*, sehingga *bot* musuh menargetkan hanya satu *bot* pada suatu waktu. Namun, pada *bot* Emo, terdapat suatu perulangan untuk mengkalkulasikan faktor risiko dari setiap *bot* yang telah di-*scan*, sehingga kompleksitas algoritma yang digunakan adalah $O(N)$, dengan N adalah jumlah *bot* yang hidup dan telah di-*scan* pada suatu *turn*. Berikut adalah analisis alternatif *bot* dari segi efektivitasnya:

1. Derik

Bot Derik efektif dalam melawan *bot* yang bergerak *mindlessly* dengan pola teratur, ataupun pada *bot* yang melakukan *locking* pada satu target pada suatu waktu seperti jenis-jenis *ram bot*. Hal ini karena pergerakan Derik yang memutar dapat menjadi target yang cukup sulit untuk di-*track* sehingga peluru *bot* lawan rentan *miss*, dan Derik dapat terus menembakkan peluru dalam jarak optimal. Namun, *bot* ini kurang efektif dalam melawan *bot* yang memiliki pergerakan strategis, misalnya *bot* yang bergerak pada lokasi dengan *minimum risk*, karena Derik dapat dengan mudah ditembak dari jarak jauh sementara ia sedang melakukan *locking* pada *bot* lainnya.

2. Botol

Bot Botol adalah *bot* yang sangat kuat dan efektif dalam melawan *bot* lain yang memiliki pergerakan simpel, misalnya *bot* *walls* yang hanya bernavigasi di tepi arena. Namun, ketika melawan *bot* dengan *movement* yang lebih *advanced*, Botol sering kali kewalahan,

terutama karena ia dapat dengan mudah ditarget oleh bot lain ketika sedang melakukan *locking* pada bot lainnya. Namun, pada posisi *spawn* di awal permainan yang baik, Botol mungkin saja meraih keuntungan besar di bagian awal dan sukses pada *round* tersebut.

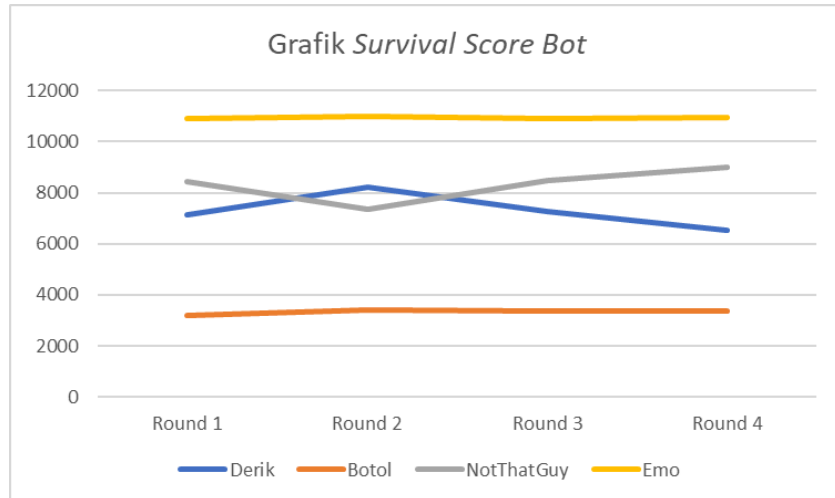
3. NotThatGuy

Bot NotThatGuy efektif untuk melawan jenis *bot* yang bergerak lurus dengan kecepatan konstan atau berdiam di tempat yang mudah diprediksi dengan *linear targeting*. *Bot* yang tidak memiliki *radar* yang terus memantau sekitarnya juga akan kesulitan untuk melacak pergerakan acak *bot* NotThatGuy. Namun, *bot* ini kurang efektif untuk melawan pergerakan yang tidak terduga dan kompleks. *Bot* yang memiliki sistem penembakan presisi tinggi juga akan lebih efektif untuk melawan *bot* NotThatGuy.

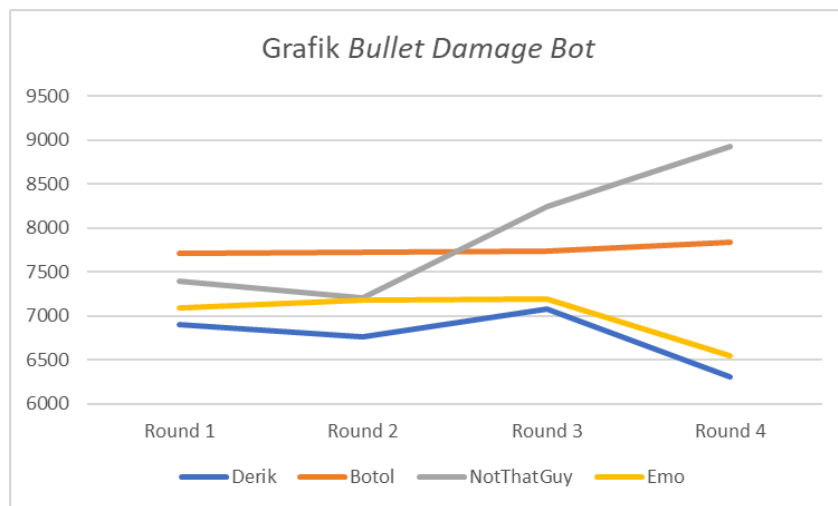
4. Emo

Bot Emo efektif untuk melawan jenis *bot* yang bergerak lurus dengan kecepatan konstan atau berdiam di tempat yang mudah diprediksi, hal ini karena prediksi linear *bot* ini sangat efektif untuk melawan musuh dengan pola pergerakan yang sederhana. *Bot* ini juga efektif untuk melawan banyak *bot* di medan arena, karena *bot* ini mampu berpindah ke posisi dengan ancaman paling minimum. Namun, *bot* Emo kurang efektif untuk melawan *bot* dengan pola pergerakan yang tidak terduga dan sering berubah arah, seperti pola osilasi atau pergerakan melingkar.

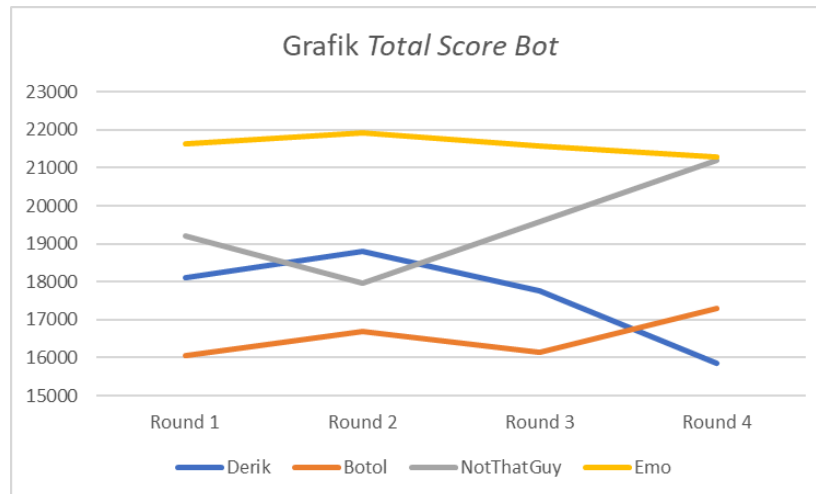
Kami juga menganalisis efektivitas keempat *bot* secara nyata melalui pengujian keempat *bot* di medan arena. Berikut merupakan grafik hasil analisis efektivitas keempat alternatif solusi sebagai gambaran untuk menentukan strategi *greedy* utama yang dipilih oleh tim kami:



Grafik 1. Tingkat Efektivitas *Bot* Berdasarkan *Survival Score*



Grafik 2. Tingkat Efektivitas *Bot* Berdasarkan *Bullet Damage*



Grafik 3. Tingkat Efektivitas *Bot* Berdasarkan *Total Score*

3.4 Hasil Pemilihan Strategi *Greedy*

Berdasarkan grafik analisis efisiensi dan efektivitas, solusi strategi *greedy* tiap *bot* dapat dievaluasi berdasarkan tiga metrik utama: *survival score* yaitu poin yang didapatkan berdasarkan seberapa lama *bot* dapat bertahan dalam satu *round*; *bullet damage* yaitu jumlah total *damage* dari tembakan peluru yang berhasil diberikan pada *bot* lain; *total score* yaitu jumlah total poin yang didapatkan.

Pada grafik *survival score*, terlihat bahwa Emo selalu memiliki nilai tertinggi di setiap ronde. Ini membuktikan bahwa strategi yang digunakan oleh Emo sangat efektif dalam bertahan. Dengan menghindari pertempuran jarak dekat dan bergerak menuju area dengan tingkat ancaman rendah, Emo mampu bertahan lebih lama dibandingkan bot lainnya. Di lain sisi, Botol secara konsisten memiliki *survival score* yang lebih rendah, mengingat strateginya yang sangat agresif sehingga lebih rentan untuk tidak bertahan lama.

Pada grafik *bullet damage*, terlihat bahwa NotThatGuy dan Botol saling bergantian dalam mengisi posisi tertinggi, di mana Botol memiliki hasil yang lebih konsisten. Hal ini membuktikan bahwa strategi Botol untuk melakukan *locking* pada suatu target selagi menabrak secara agresif secara konsisten mampu menghasilkan poin dari peluru yang relatif tinggi. Sementara itu, NotThatGuy yang juga menerapkan strategi *locking* pada tembakannya, berhasil mendapatkan poin yang cukup tinggi, walaupun hasilnya lebih fluktuatif.

Pada grafik *total score*, dapat dilihat bahwa Emo cukup mendominasi dan selalu unggul dalam setiap ronde. Ini menunjukkan bahwa strategi *greedy* yang digunakan oleh Emo untuk pergi ke tempat dengan risiko paling rendah dapat membuatnya lebih fleksibel terhadap banyak situasi, serta efektif dalam mendapatkan poin yang lebih banyak, terutama melalui tingkat *survival* yang tinggi.

Maka, Emo dipilih sebagai bot dengan strategi paling optimal. Emo berhasil bertahan lebih lama dibandingkan lawan-lawannya, lebih fleksibel terhadap berbagai situasi dan tidak terlalu bergantung pada posisi *spawn*, memiliki efektivitas serangan yang tinggi, serta mampu memanfaatkan celah dalam pertempuran untuk mendapatkan poin yang lebih banyak.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi 4 Alternatif Solusi

```
class Derik : Bot

    // initialize
    isRadarLocked <- false
    shoot <- false
    maju <- true
    preferredDist <- 25
    offset <- 10
    wait <- 50

    procedure Run()
        // memisahkan perputaran Gun dengan Body
        AdjustGunForBodyTurn <- true
        // memisahkan perputaran Radar dengan Body
        AdjustRadarForBodyTurn <- true
        // memisahkan perputaran Radar dengan Gun
        AdjustRadarForGunTurn <- true

        // radar berputar terus-menerus (hingga mendeteksi musuh)
        SetTurnRadarRight(INFINITY)
        SetForward(100)

        while(isRunning) do

            Movement()
            // logika mengatur tembakan
            if (shoot) then
                HandleShoot()

            // logika mengatur Locking radar
            if (isRadarLocked and RadarTurnRemaining < 0.01) then
                isRadarLocked <- false
                SetTurnRadarRight(INFINITY)
            Go()
```

```

procedure OnScannedBot(e)
    // update informasi enemy
    enemy <- e
    shoot <- true

procedure OnTick(e)
    // kurangi waktu menunggu tiap tick
    if wait > 0 then
        wait <- wait - 1

    // jika wait 0, putar balik arah bot, kemudian set kembali waktu
    menunggu
    if wait <= 0 then
        maju <- not maju
        wait <- random(30, 75)

procedure HandleShoot()
    bulletSpeed <- 20 - (3 * optimalFirepower)
    timeToHit <- DistanceTo(enemy.X, enemy.Y) / bulletSpeed

    enemyVelocityX <- enemy.Speed * cos(ToRadians(enemy.Direction))
    enemyVelocityY <- enemy.Speed * sin(ToRadians(enemy.Direction))

    predictedX <- enemy.X + enemyVelocityX * timeToHit
    predictedY <- enemy.Y + enemyVelocityY * timeToHit

    // putar Gun ke arah yang diprediksi
    firingAngle <- GunBearingTo(predictedX, predictedY)
    SetTurnGunLeft(firingAngle)

    // Lakukan Locking target
    radarOffset <- NormalizeAngle(RadarBearingTo(enemy.X, enemy.Y))
    SetTurnRadarLeft(radarOffset)
    isRadarLocked <- true

    if GunHeat = 0 and GunTurnRemaining < 3 then
        SetFire(firepower)
    shoot <- false

procedure Movement()

```

```

// jika dekat wall, putar balik arah
if IsNearWall() and wait <= 0 then
    wait <- 50
    maju <- not maju

// mengatur maju/mundurnya bot
if maju then
    SetForward(100)
else
    SetBack(100)

// jika telah melakukan locking target, lakukan pergerakan memutar
if shoot then
    angleToEnemy <- BearingTo(enemy.X, enemy.Y)

    // jika sedang bergerak mundur, putar angleToEnemy untuk
    menyesuaikan
    if not maju then angleToEnemy <- angleToEnemy + 180

    // jika jarak terlalu dekat, gerak menjauh
    if DistanceTo(enemy.X, enemy.Y) < preferredDist - offset then
        adjustAngle <- 150
    // jika jarak terlalu jauh, gerak mendekat
    else if DistanceTo(enemy.X, enemy.Y) > preferredDist + offset
then
        adjustAngle <- 60
    // jika jarak telah optimal, gerak melingkar
    else
        adjustAngle <- 90

    if angleToEnemy > 0 then
        angleToEnemy <- angleToEnemy - adjustAngle
    else
        angleToEnemy <- angleToEnemy + adjustAngle

    MoveInDirection(angleToEnemy)

```

```

class Derik : Bot

    // initialize

```

```

isRadarLocked <- false
shoot <- false
maju <- true
preferredDist <- 25
offset <- 10
wait <- 50

procedure Run()
    // memisahkan perputaran Gun dengan Body
    AdjustGunForBodyTurn <- true
    // memisahkan perputaran Radar dengan Body
    AdjustRadarForBodyTurn <- true
    // memisahkan perputaran Radar dengan Gun
    AdjustRadarForGunTurn <- true

    // radar berputar terus-menerus (hingga mendeteksi musuh)
    SetTurnRadarRight(INFINITY)
    SetForward(100)

    while(isRunning) do

        Movement()
        // logika mengatur tembakan
        if (shoot) then
            HandleShoot()

        // logika mengatur locking radar
        if (isRadarLocked and RadarTurnRemaining < 0.01) then
            isRadarLocked <- false
            SetTurnRadarRight(INFINITY)
        Go()

procedure OnScannedBot(e)
    // update informasi enemy
    enemy <- e
    shoot <- true

procedure OnTick(e)
    // kurangi waktu menunggu tiap tick
    if wait > 0 then
        wait <- wait - 1

```

```
// jika wait 0, putar balik arah bot, kemudian set kembali waktu menunggu
```

```
if wait <= 0 then  
    maju <- not maju  
    wait <- random(30, 75)
```

```
procedure HandleShoot()
```

```
    bulletSpeed <- 20 - (3 * optimalFirepower)  
    timeToHit <- DistanceTo(enemy.X, enemy.Y) / bulletSpeed
```

```
    enemyVelocityX <- enemy.Speed * cos(ToRadians(enemy.Direction))  
    enemyVelocityY <- enemy.Speed * sin(ToRadians(enemy.Direction))
```

```
    predictedX <- enemy.X + enemyVelocityX * timeToHit  
    predictedY <- enemy.Y + enemyVelocityY * timeToHit
```

```
// putar Gun ke arah yang diprediksi  
    firingAngle <- GunBearingTo(predictedX, predictedY)  
    SetTurnGunLeft(firingAngle)
```

```
// lakukan Locking target  
    radarOffset <- NormalizeAngle(RadarBearingTo(enemy.X, enemy.Y))  
    SetTurnRadarLeft(radarOffset)  
    isRadarLocked <- true
```

```
    if GunHeat = 0 and GunTurnRemaining < 3 then  
        SetFire(firepower)  
    shoot <- false
```

```
procedure Movement()
```

```
// jika telah melakukan Locking target
```

```
if shoot then  
    // gerak menuju musuh  
    angleToEnemy <- BearingTo(enemy.X, enemy.Y)  
    MoveInDirection(angleToEnemy)
```

```
class NotThatGuy : Bot
```

```
// inisialisasi  
    random <- new Random()
```

```

movingForward <- true

enemyX <- 0
enemyY <- 0
enemyDirection <- 0
enemySpeed <- 0

isRadarLocked <- false
shoot <- false

procedure Run()
  // warna bot
  BodyColor <- Color.FromArgb(0x99, 0x99, 0x99)
  TurretColor <- Color.FromArgb(0x88, 0x88, 0x88)
  RadarColor <- Color.FromArgb(0x66, 0x66, 0x66)

  // memisahkan perputaran Gun dengan Body
  AdjustGunForBodyTurn <- true
  // memisahkan perputaran Radar dengan Body
  AdjustRadarForBodyTurn <- true
  // memisahkan perputaran Radar dengan Gun
  AdjustRadarForGunTurn <- true

  // Radar berputar terus menerus hingga mendeteksi musuh
  SetTurnRadarRight(INFINITY)

  while IsRunning do
    MoveRandomly()
    // logika mengatur tembakan
    if (shoot) then
      HandleShoot()

    // logika mengatur locking radar
    if (isRadarLocked and Abs(RadarTurnRemaining)) < 0.01 then
      isRadarLocked <- false
      SetTurnRadarRight(INFINITY)
    Go()

  procedure HandleShoot()
    // kalkulasi arah tembakan
    bulletSpeed <- 20 - (3 * GetOptimalFirepower(DistanceTo(enemyX,
enemyY)))

```

```

timeToHit <- DistanceTo(enemyX, enemyY) / bulletSpeed

// prediksi posisi bot musuh
enemyVelocityX <- enemySpeed *
Cos(DegreesToRadians(enemyDirection))
enemyVelocityY <- enemySpeed *
Sin(DegreesToRadians(enemyDirection))
predictedX <- enemyX + enemyVelocityX * timeToHit
predictedY <- enemyY + enemyVelocityY * timeToHit

// mengubah arah Gun ke posisi yang sudah diprediksi
firingAngle <- GunBearingTo(predictedX, predictedY)
SetTurnGunLeft(firingAngle)

// Lock radar ke bot musuh
radarOffset <- NormalizeAngle(RadarBearingTo(enemyX, enemyY))
SetTurnRadarLeft(radarOffset)
isRadarLocked <- true

// menembakkan peluru
firepower <- GetOptimalFirepower(DistanceTo(enemyX, enemyY))
if GunHeat == 0 and Abs(GunTurnRemaining) < 3.0 then
    SetFire(firepower)
shoot <- false

procedure OnScannedBot(e)
    // update informasi musuh
    enemyX <- e.X
    enemyY <- e.Y
    enemyDirection <- e.Direction
    enemySpeed <- e.Speed
    shoot <- true

procedure MoveRandomly()
    // ubah arah ketika mendekati wall
    if IsNearWall() then
        SetTurnRight(random.Next(90, 180))
        SetForward(150)
        return

    // ubah arah ketika mendekati bot musuh
    if IsNearBot() then

```

```

        SetTurnRight(random.Next(45, 90))
        SetBack(100)
        return

    moveDistance <- random.Next(200, 500)
    turnAngle <- random.Next(45, 180)
    SetForward(moveDistance)
    SetTurnRight(turnAngle)

    procedure IsNearWall()
        return X < 100 or X > ArenaWidth - 100 or Y < 100 or Y >
ArenaHeight - 100

    procedure IsNearBot()
        return Speed < 1.5

    procedure OnHitWall(e)
        ReverseDirection()

    procedure OnHitBot(e)
        ReverseDirection()

    procedure ReverseDirection()
        if movingForward then
            SetBack(40000)
            movingForward <- false
        else
            SetForward(40000)
            movingForward <- true

    procedure NormalizeAngle(angle)
        while angle > 180 do angle <- angle - 360
        while angle < -180 do angle <- angle + 360
        return angle

    // sesuaikan firepower dengan jarak bot ke musuh
    procedure GetOptimalFirepower(distance)
        if distance < 200 then
            return 3
        else if distance < 500 then
            return 2
        else

```



```
return 1
```

```
procedure DegreesToRadians(degrees)  
    return degrees * (PI / 180)
```

```
class Emo : Bot
```

```
    // inisialisasi
```

```
    safeDistance <- 25
```

```
    minMoveRange <- 100
```

```
    maxMoveRange <- 200
```

```
    stopDistance <- 5
```

```
    angleDivision <- 36
```

```
    targetX <- 0
```

```
    targetY <- 0
```

```
    randomizer <- new Random()
```

```
    opponentList <- new List<OpponentInfo>()
```

```
    guessFactors <- new Dictionary<int, double>()
```

```
    shoot <- false
```

```
    enemy <- null
```

```
    isRadarLocked <- false
```

```
    procedure Run()
```

```
        // warna bot
```

```
        BodyColor <- Color.Red
```

```
        TurretColor <- Color.Red
```

```
        RadarColor <- Color.Red
```

```
        // memisahkan perputaran Gun dengan Body
```

```
        AdjustRadarForBodyTurn <- true
```

```
        // memisahkan perputaran Radar dengan Gun
```

```
        AdjustRadarForGunTurn <- true
```

```
        // memisahkan perputaran Radar dengan Body
```

```
        AdjustGunForBodyTurn <- true
```

```
        // radar berputar terus-menerus (hingga mendeteksi musuh)
```

```
        SetTurnRadarRight(INFINITY)
```

```
    while IsRunning do
```

```

    Movement()
    // logika mengatur tembakan
    if (shoot) then
        HandleShoot()

    // logika mengatur Locking radar
    if (isRadarLocked and Abs(RadarTurnRemaining) < 0.01) then
        isRadarLocked <- false
        SetTurnRadarRight(INFINITY)
    Go()

procedure Movement()
    // strategi minimum risk
    if DistanceRemaining < stopDistance then
        optimalX <- X
        optimalY <- Y
        minThreat <- MAX_VALUE

        for i <- 0 to angleDivision - 1 do
            angle <- (2 * PI / angleDivision) * i
            moveRange <- randomizer.NextDouble() * (maxMoveRange -
minMoveRange) + minMoveRange
            posX <- X + moveRange * Cos(angle)
            posY <- Y + moveRange * Sin(angle)

            if posX < safeDistance or posX > ArenaWidth - safeDistance
or
                posY < safeDistance or posY > ArenaHeight - safeDistance
then
                continue

            threatLevel <- EvaluateThreat(posX, posY)
            if threatLevel < minThreat then
                minThreat <- threatLevel
                optimalX <- posX
                optimalY <- posY

        targetX <- optimalX
        targetY <- optimalY

    adjustAngle <- BearingTo(targetX, targetY) * PI / 180
    SetTurnLeft(180 / PI * Tan(adjustAngle))

```

```

        SetForward(DistanceTo(targetX, targetY) * Cos(adjustAngle))

    // menghitung arah tembakan dengan linear targeting berdasarkan
    // kecepatan dan arah musuh
    procedure HandleShoot()
        bulletSpeed <- 20 - (3 * GetOptimalFirepower(DistanceTo(enemy.X,
        enemy.Y)))
        timeToHit <- DistanceTo(enemy.X, enemy.Y) / bulletSpeed

        enemyVelocityX <- enemy.Speed *
        Cos(DegreesToRadians(enemy.Direction))
        enemyVelocityY <- enemy.Speed *
        Sin(DegreesToRadians(enemy.Direction))
        predictedX <- enemy.X + enemyVelocityX * timeToHit
        predictedY <- enemy.Y + enemyVelocityY * timeToHit

        firingAngle <- GunBearingTo(predictedX, predictedY)
        SetTurnGunLeft(NormalizeAngle(firingAngle))

        if EnemyCount == 1 then
            radarOffset <- NormalizeAngle(RadarBearingTo(enemy.X, enemy.Y))
            SetTurnRadarLeft(radarOffset)
            isRadarLocked <- true

        firepower <- GetOptimalFirepower(DistanceTo(enemy.X, enemy.Y))
        if GunHeat == 0 and Abs(GunTurnRemaining) < 3.0 then
            SetFire(firepower)
            shoot <- false

    // prosedur untuk pindah ke tempat paling minim ancaman
    procedure EvaluateThreat(candidateX, candidateY)
        threatLevel <- 0
        for each opponent in opponentList do
            if opponent.IsActive then
                distSquared <- Pow(candidateX - opponent.LastX, 2) +
                Pow(candidateY - opponent.LastY, 2)
                energyFactor <- opponent.LastPower / 100.0
                distanceFactor <- 1.0 / Max(distSquared, 1e-6)
                threat <- energyFactor * distanceFactor

                if distSquared < 10000 then
                    threat <- threat * 2

```

```

        threatLevel <- threatLevel + threat
    return threatLevel

procedure OnScannedBot(e)
    opponent <- opponentList.Find(o => o.BotId == e.ScannedBotId)
    if opponent == null then
        opponent <- new OpponentInfo { BotId = e.ScannedBotId }
        opponentList.Add(opponent)

    opponent.Update(e.X, e.Y, e.Speed, e.Direction, e.Energy)

    shoot <- true
    if enemy == null then
        enemy <- e
    else if enemy.ScannedBotId == e.ScannedBotId or DistanceTo(e.X,
e.Y) < DistanceTo(enemy.X, enemy.Y) then
        enemy <- e

procedure OnBotDeath(e)
    // jika sebuah bot kehabisan energi, dihapus dari list bot hidup
    opponent <- opponentList.Find(o => o.BotId == e.VictimId)
    if opponent != null then
        opponent.IsActive <- false

    if enemy.ScannedBotId == e.VictimId then
        enemy <- null

procedure GetOptimalFirepower(distance)
    // sesuaikan firepower dengan jarak bot ke musuh
    if Energy < 20 then
        return 1
    if distance < 50 then
        return 3
    else if distance < 300 then
        return 2
    else if distance < 600 then
        return 1.5
    else
        return 1

procedure DegreesToRadians(degrees)

```

```

        return degrees * (PI / 180)

    procedure NormalizeAngle(angle)
        while angle > 180 do angle <- angle - 360
        while angle < -180 do angle <- angle + 360
        return angle

class OpponentInfo
    BotId <- 0
    LastX <- 0
    LastY <- 0
    LastSpeed <- 0
    LastHeading <- 0
    LastPower <- 0
    IsActive <- true
    movementHistory <- new List<double>()

    procedure Update(x, y, speed, heading, energy)
        // update state musuh
        LastX <- x
        LastY <- y
        LastSpeed <- speed
        LastHeading <- heading
        LastPower <- energy
        IsActive <- true

        angleChange <- NormalizeAngle(heading - LastHeading)
        movementHistory.Add(angleChange)
        if movementHistory.Count > 100 then
            movementHistory.RemoveAt(0)

    procedure NormalizeAngle(angle)
        while angle > 180 do angle <- angle - 360
        while angle < -180 do angle <- angle + 360
        return angle

```

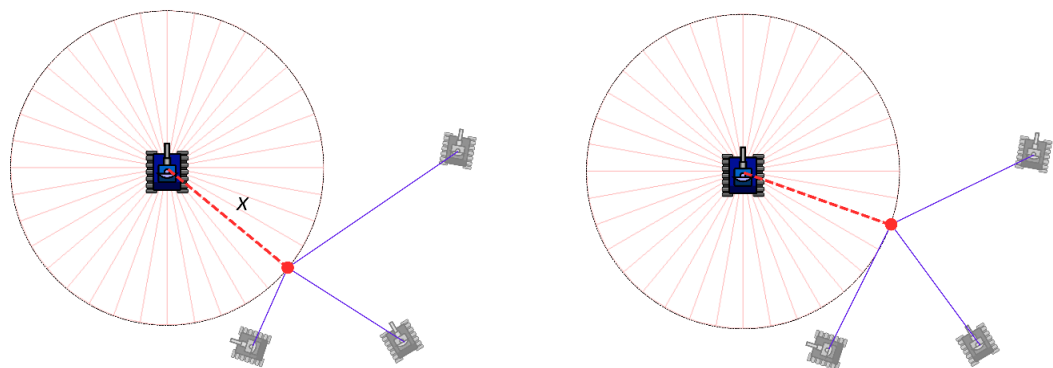
4.2 Penjelasan Implementasi Solusi

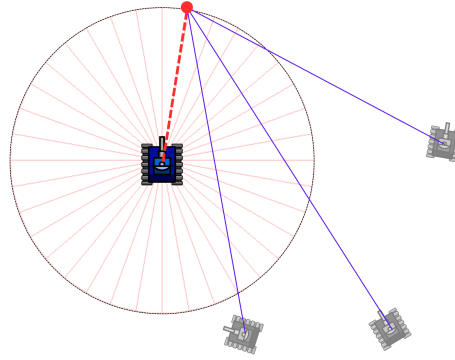
1. Emo

Bahasa pemrograman yang digunakan untuk mengembangkan *bot* adalah C#, di mana program dikembangkan dengan paradigma Pemrograman Berbasis Objek (*Object*

Oriented Programming). *Bot* Emo diimplementasikan dengan meng-*inherit* *parent class* *Bot* yang telah disediakan dengan berbagai *api-api* yang dapat dipanggil. Terdapat beberapa atribut dan metode (fungsi) yang mengatur logika pergerakan bot:

- Run – adalah metode utama yang mengatur logika dan pergerakan *bot*. Di dalam metode ini, terdapat *loop* utama yang akan memanggil fungsi-fungsi lain seperti Movement untuk mengatur pergerakan dan HandleShoot untuk mengatur penembakan. *Bot* juga akan terus memantau *radar* untuk mendeteksi musuh.
- Movement – adalah metode ini mengatur logika pergerakan *bot*. *Bot* akan memilih posisi optimal berdasarkan evaluasi tingkat ancaman (*EvaluateThreat*) dari musuh. Jika bot berada terlalu dekat dari target, *bot* akan menyesuaikan jaraknya dengan bergerak menjauh. Tingkat ancaman ini dihitung berdasarkan jarak ke target dan total energi yang dimiliki target. Tingkat ancaman akan dihitung pada 36 buah titik di sekitar lingkaran dalam jarak x tertentu yang ditentukan secara acak dalam range 100 hingga 200 piksel dari Emo. Pada setiap titik, tingkat ancaman akan dihitung dengan menjumlahkan hasil perkalian faktor jarak dan faktor energi dari setiap bot yang masih aktif dan telah di-*scan* radar. Kemudian, Emo akan bergerak ke titik dengan tingkat ancaman yang paling kecil, untuk meminimumkan risiko terlibat dalam pertempuran jarak dekat.





Gambar 5. Ilustrasi Perhitungan Ancaman di Tiap Titik

- HandleShoot – adalah metode untuk mengatur logika penembakan. *Bot* akan memprediksi posisi musuh berdasarkan kecepatan dan arah pergerakan musuh, kemudian menembakkan peluru dengan kekuatan yang optimal (GetOptimalFirepower). Prediksi posisi musuh dilakukan dengan asumsi musuh bergerak lurus dengan kecepatan konstan.
- OnScannedBot – adalah metode yang dipanggil setiap kali *bot* berhasil mendeteksi musuh melalui *radar*. Informasi musuh yang terdeteksi akan diperbarui, dan *bot* akan memutuskan apakah akan menembak atau tidak berdasarkan jarak dan prioritas musuh.
- OnBotDeath – adalah metode yang dipanggil ketika sebuah *bot* mati. Jika *bot* yang mati adalah musuh yang sedang ditarget, *bot* akan menghapusnya dari daftar target.
- EvaluateThreat – adalah metode untuk menghitung tingkat ancaman dari suatu posisi berdasarkan jarak dan energi musuh. Semakin dekat dan semakin kuat musuh, semakin tinggi tingkat ancamannya.
- GetOptimalFirepower – adalah metode untuk mengembalikan kekuatan tembakan optimal berdasarkan jarak ke musuh. Semakin dekat jaraknya, semakin besar kekuatan tembakan yang digunakan.
- NormalizeAngle – metode ini menormalkan sudut ke dalam rentang $[-180, 180]$ derajat untuk memudahkan perhitungan arah.
- DegreesToRadians – adalah metode untuk mengonversi sudut dari derajat ke radian.

- OpponentInfo – kelas ini digunakan untuk menyimpan informasi tentang musuh, seperti posisi terakhir, kecepatan, arah, dan energi. Informasi ini digunakan untuk memprediksi pergerakan musuh dan menentukan strategi penembakan.

4.2 Penjelasan Implementasi Alternatif Solusi

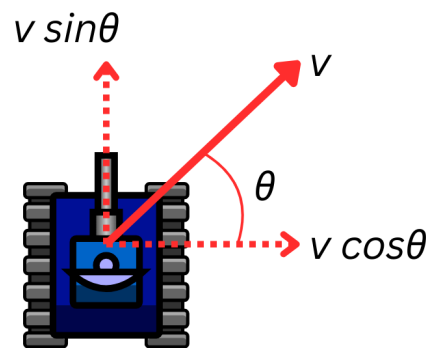
1. Derik

Bahasa pemrograman yang digunakan untuk mengembangkan *bot* adalah C#, di mana program dikembangkan dengan paradigma Pemrograman Berbasis Objek (*Object Oriented Programming*). *Bot* Derik diimplementasikan dengan meng-*inherit* *parent class* *Bot* yang telah disediakan dengan berbagai *api-api* yang dapat dipanggil. Terdapat beberapa atribut dan metode (fungsi) yang mengatur logika pergerakan bot:

- Run – adalah metode utama yang mengatur logika dan pergerakan *bot*. Di dalam metode ini, terdapat *loop* utama yang akan memanggil fungsi-fungsi lain yang mengatur pergerakan, serangan, ataupun reaksi *bot* terhadap suatu *event* di arena. Lebih tepatnya, fungsi yang akan dipanggil pada *loop* utama adalah fungsi *HandleShoot* untuk mengatur tembakan, dan *Movement* untuk mengatur pergerakan.
- OnTick – adalah metode yang dipanggil setiap satu ‘*tick*’ atau satu *turn*, yaitu satuan waktu terkecil dalam permainan.
- OnScannedBot – adalah metode yang dipanggil setiap kali *bot* berhasil mendeteksi musuh melalui radar. Metode ini adalah metode yang bersifat *blocking*, sehingga yang dilakukan pada metode ini hanyalah logika simpel dan tidak berat secara komputasi, yaitu memperbarui informasi tentang musuh yang di-*scan*.
- OnHitWall – adalah metode dipanggil ketika *bot* menabrak dinding arena.
- IsNearWall – adalah metode yang akan mengembalikan nilai *true* jika *bot* berada dekat dengan dinding arena, yaitu berjarak sekitar 50 piksel dari dinding arena.
- MoveInDirection – adalah metode untuk melakukan pergerakan belok menggunakan sudut yang telah dinormalisasi

- *GetOptimalFirepower* – adalah metode untuk mendapatkan *firepower* yang dianggap paling optimal berdasarkan jarak dan energi musuh yang menjadi target
- *HandleShoot* – adalah metode yang mengatur logika utama dalam melakukan penembakan peluru pada musuh yang ditarget. Peluru akan ditembakkan dengan asumsi *bot* musuh akan bergerak lurus ke arah yang dituju dan dengan kecepatan konstan.

Kecepatan v dari musuh dapat dipecah menjadi dua komponen v_x dan v_y



Gambar 6. Konfigurasi Arah *Bot* Derik

Dengan mendapatkan informasi posisi musuh x dan y pada arena, kecepatan, dan sudut hadap musuh, posisi akhir musuh dapat diprediksi (dengan asumsi musuh bergerak linear tanpa berbelok).

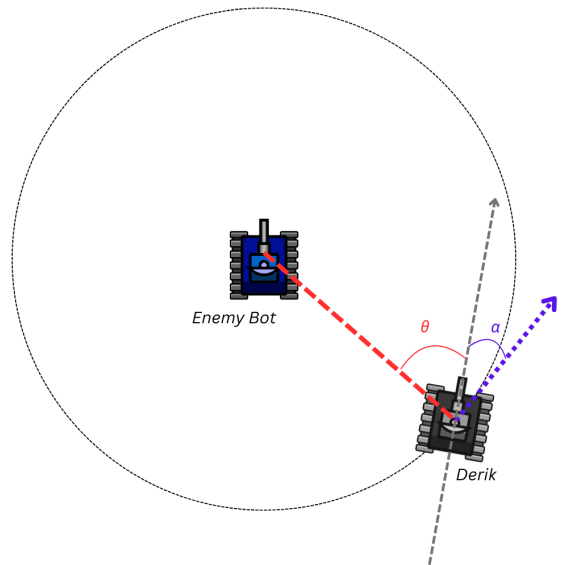
$$pred_x = x + tv_x \cos \theta$$

$$pred_y = y + tv_y \sin \theta$$

Dengan t adalah waktu (banyak *turn*) yang dibutuhkan peluru untuk mencapai musuh. Waktu (banyak *turn*) yang dibutuhkan peluru diaproksimasi dengan waktu yang dibutuhkan peluru untuk mencapai posisi Bot target saat ini.

- *Movement* – adalah metode yang mengatur logika utama dalam pergerakan *bot*. Jika target berada pada jarak yang lebih jauh dari jarak ideal, Derik akan mendekat. Jika target berada pada jarak yang lebih dekat dari jarak ideal, Derik akan menjauh. Jika target telah berada pada jarak yang ideal, Derik akan bergerak mengitari (memutari) target. Namun, pergerakan perputaran ini tidak konstan. Derik akan berputar dalam pola osilasi, yaitu pergantian arah putaran (bergantian

antara searah dan berlawanan arah jarum jam) setelah beberapa *turn* telah terlewatkan.



Gambar 7. Konfigurasi Arah Bot Derik

Untuk bergerak mengitari *bot* lawan, Derik harus bergerak maju (atau mundur) dengan sudut 90 derajat relatif terhadap posisi bot musuh. Misal θ adalah sudut yang terbentuk antara *bot* musuh dengan arah hadap Derik. Maka, untuk bergerak mengitari lawan, Derik harus berotasi sebesar α , di mana:

$$\alpha = 90^\circ - \theta$$

Maka, untuk bergerak menjauh ataupun mendekat, yang perlu dilakukan adalah mengubah sudut relatif terhadap posisi bot musuh yang diinginkan. Pada implementasi, untuk bergerak menjauh:

$$\alpha = 120^\circ - \theta$$

Sedangkan untuk bergerak mendekat:

$$\alpha = 60^\circ - \theta$$

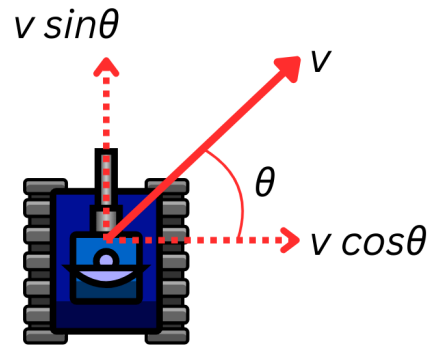
- `NormalizeAngle` – adalah metode untuk melakukan normalisasi sudut (dalam derajat) ke dalam *range* $[-180, 180]$ derajat.
- `ToRadians` – adalah metode untuk mengonversikan satuan sudut derajat menjadi radian
- `ToDegrees` – adalah metode untuk mengonversikan satuan sudut radian menjadi derajat

2. Botol

Bahasa pemrograman yang digunakan untuk mengembangkan *bot* adalah C#, di mana program dikembangkan dengan paradigma Pemrograman Berbasis Objek (*Object Oriented Programming*). *Bot* Botol diimplementasikan dengan meng-*inherit* *parent class* *Bot* yang telah disediakan dengan berbagai *api-api* yang dapat dipanggil. Terdapat beberapa atribut dan metode (fungsi) yang mengatur logika pergerakan bot:

- *Run* – adalah metode utama yang mengatur logika dan pergerakan *bot*. Di dalam metode ini, terdapat *loop* utama yang akan memanggil fungsi-fungsi lain yang mengatur pergerakan, serangan, ataupun reaksi bot terhadap suatu *event* di arena. Lebih tepatnya, fungsi yang akan dipanggil pada *loop* utama adalah fungsi *HandleShoot* untuk mengatur tembakan, dan *Movement* untuk mengatur pergerakan.
- *OnTick* – adalah metode yang dipanggil setiap satu ‘*tick*’ atau satu *turn*, yaitu satuan waktu terkecil dalam permainan.
- *OnScannedBot* – adalah metode yang dipanggil setiap kali *bot* berhasil mendeteksi musuh melalui *radar*. Metode ini adalah metode yang bersifat *blocking*, sehingga yang dilakukan pada metode ini hanyalah logika simpel dan tidak berat secara komputasi, yaitu memperbarui informasi tentang musuh yang di-*scan*.
- *OnHitWall* – adalah metode dipanggil ketika *bot* menabrak dinding arena.
- *IsNearWall* – adalah metode yang akan mengembalikan nilai *true* jika *bot* berada dekat dengan dinding arena, yaitu berjarak sekitar 50 piksel dari dinding arena.
- *MoveInDirection* – adalah metode untuk melakukan pergerakan belok menggunakan sudut yang telah dinormalisasi
- *GetOptimalFirepower* – adalah metode untuk mendapatkan *firepower* yang dianggap paling optimal berdasarkan jarak dan energi musuh yang menjadi target
- *HandleShoot* – adalah metode yang mengatur logika utama dalam melakukan penembakan peluru pada musuh yang ditarget. Peluru akan ditembakkan dengan asumsi *bot* musuh akan bergerak lurus ke arah yang dituju dan dengan kecepatan konstan.

Kecepatan v dari musuh dapat dipecah menjadi dua komponen v_x dan v_y



Gambar 8. Konfigurasi arah *bot* Botol

Dengan mendapatkan informasi posisi musuh x dan y pada arena, kecepatan, dan sudut hadap musuh, posisi akhir musuh dapat diprediksi (dengan asumsi musuh bergerak linear tanpa berbelok).

$$pred_x = x + tv_x \cos \theta$$

$$pred_y = y + tv_y \sin \theta$$

Dengan t adalah waktu (banyak *turn*) yang dibutuhkan peluru untuk mencapai musuh. Waktu (banyak *turn*) yang dibutuhkan peluru diaproksimasi dengan waktu yang dibutuhkan peluru untuk mencapai posisi Bot target saat ini.

- Movement – adalah metode yang mengatur logika utama dalam pergerakan *bot*. Setiap *turn*, *bot* akan mendekat ke arah (dan bahkan menabrak) musuh yang telah ditarget, yaitu dengan berotasi sebesar θ sambil bergerak maju, di mana θ adalah sudut yang dibentuk antara Botol relatif terhadap posisi *bot* musuh.
- NormalizeAngle – adalah metode untuk melakukan normalisasi sudut (dalam derajat) ke dalam *range* $[-180, 180]$ derajat.
- ToRadians – adalah metode untuk mengonversikan satuan sudut derajat menjadi radian
- ToDegrees – adalah metode untuk mengonversikan satuan sudut radian menjadi derajat

3. NotThatGuy

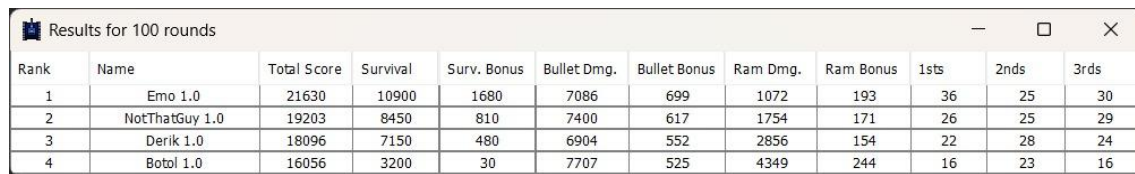
Bahasa pemrograman yang digunakan untuk mengembangkan *bot* adalah C#, di mana program dikembangkan dengan paradigma Pemrograman Berbasis Objek (*Object Oriented Programming*). Bot NotThatGuy diimplementasikan dengan meng-*inherit* *parent class* Bot yang telah disediakan dengan berbagai *api-api* yang dapat dipanggil. Terdapat beberapa atribut dan metode (fungsi) yang mengatur logika pergerakan bot:

- Run – adalah metode utama yang mengatur logika dan pergerakan bot. Di dalamnya terdapat *loop* utama yang akan memanggil fungsi-fungsi lain seperti MoveRandomly untuk mengatur pergerakan dan HandleShoot untuk mengatur penembakan.
- OnScannedBot – adalah metode yang dipanggil setiap kali *bot* berhasil mendeteksi musuh melalui *radar*. Metode ini memperbarui informasi tentang musuh yang terdeteksi dan menjadi tanda bahwa *bot* harus menembak (*shoot = true*).
- MoveRandomly – metode ini mengatur pergerakan bot secara acak dengan jarak tempuh antara 200 hingga 500 piksel. Jika bot mendekati dinding atau bot lain, bot akan berbalik arah atau menjauhi (berputar ke kanan).
- IsNearWall – adalah metode untuk mendeteksi apakah *bot* dekat dengan *wall* arena, mengembalikan true jika benar.
- IsNearBot – adalah metode untuk mendeteksi apakah *bot* dekat dengan *bot* lain. Jika true, maka kecepatan *bot* NotThatGuy akan diturunkan untuk mencegah *collision* dengan *bot* lain.
- OnHitWall – metode ini dipanggil ketika *bot* menabrak dinding arena, *bot* akan membalik arah gerakannya.
- OnHitBot – metode ini dipanggil ketika *bot* menabrak *bot* lain, *bot* akan membalik arah gerakannya.
- ReverseDirection – adalah metode yang dipanggil untuk membalik arah gerakan *bot* NotThatGuy.
- NormalizeAngle – adalah metode yang berfungsi untuk menormalisasi sudut (dalam derajat) ke dalam rentang [-180, 180].

- `GetOptimalFirePower` – adalah metode untuk mengembalikan kekuatan tembakan (*firepower*) yang optimal berdasarkan jarak ke musuh.
- `DegreesToRadians` – adalah metode untuk mengonversi sudut dari derajat ke radian.

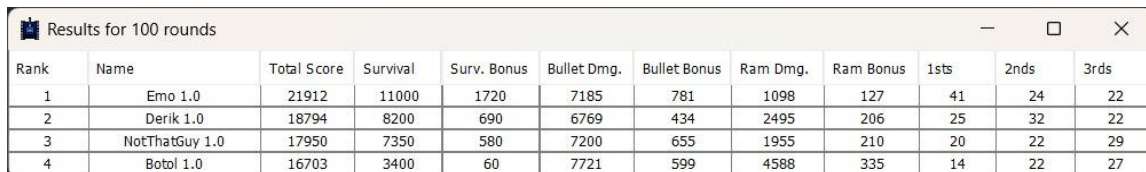
4.3 Pengujian

Berikut merupakan hasil pengujian keempat alternatif *bot* (1 vs 1 vs 1 vs 1) sebanyak 4 kali (metode *best of seven*) dengan masing-masing pengujian dilakukan sebanyak 100 *rounds*:



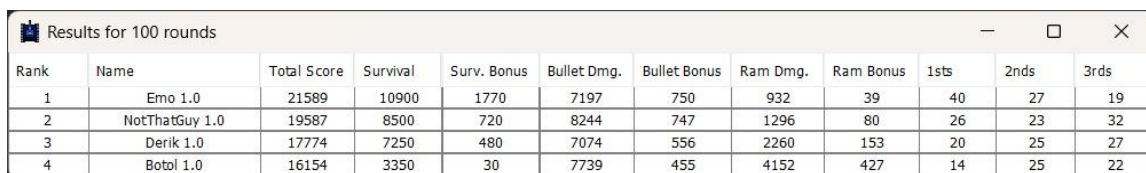
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Emo 1.0	21630	10900	1680	7086	699	1072	193	36	25	30
2	NotThatGuy 1.0	19203	8450	810	7400	617	1754	171	26	25	29
3	Derik 1.0	18096	7150	480	6904	552	2856	154	22	28	24
4	Botol 1.0	16056	3200	30	7707	525	4349	244	16	23	16

Gambar 9. Hasil Pengujian 1 Keempat *Bot*



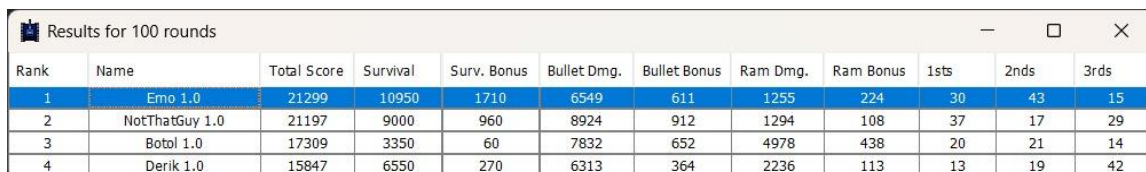
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Emo 1.0	21912	11000	1720	7185	781	1098	127	41	24	22
2	Derik 1.0	18794	8200	690	6769	434	2495	206	25	32	22
3	NotThatGuy 1.0	17950	7350	580	7200	655	1955	210	20	22	29
4	Botol 1.0	16703	3400	60	7721	599	4588	335	14	22	27

Gambar 10. Hasil Pengujian 2 Keempat *Bot*



Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Emo 1.0	21589	10900	1770	7197	750	932	39	40	27	19
2	NotThatGuy 1.0	19587	8500	720	8244	747	1296	80	26	23	32
3	Derik 1.0	17774	7250	480	7074	556	2260	153	20	25	27
4	Botol 1.0	16154	3350	30	7739	455	4152	427	14	25	22

Gambar 11. Hasil Pengujian 3 Keempat *Bot*



Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Emo 1.0	21299	10950	1710	6549	611	1255	224	30	43	15
2	NotThatGuy 1.0	21197	9000	960	8924	912	1294	108	37	17	29
3	Botol 1.0	17309	3350	60	7832	652	4978	438	20	21	14
4	Derik 1.0	15847	6550	270	6313	364	2236	113	13	19	42

Gambar 12. Hasil Pengujian 4 Keempat *bot*

4.4 Analisis

Berdasarkan hasil pengujian, *bot* Emo secara konsisten meraih posisi pertama dengan skor yang relatif lebih tinggi dibandingkan *bot* lainnya. Keunggulan ini didorong oleh strategi *greedy* yang lebih efektif, di mana Emo selalu bergerak menuju area dengan tingkat ancaman yang paling rendah berdasarkan posisi *bot* lain di arena. Dengan menghindari pertempuran jarak dekat dan memilih posisi yang aman, Emo dapat bertahan lebih lama untuk memperoleh poin *survival* yang tinggi, serta mendapatkan lebih banyak peluang untuk menyerang lawan yang sedang bertarung.

Keunggulan strategi ini semakin terlihat karena karakteristik *bot* lawan, terutama Derik dan Botol, yang menggunakan pendekatan *locking* yang berfokus pada suatu target dalam satu waktu. Akibatnya, ketika Derik dan Botol sedang bertarung satu sama lain atau menyerang *bot* lain, Emo dapat dengan mudah mengambil posisi yang lebih strategis, menghindari bahaya, dan mendapatkan poin tanpa banyak gangguan. Ketika Emo menjadi target *locking* dari Derik dan Botol, ia juga dapat dengan mudah menghindar dan bergerak menjauh. Pola permainan seperti ini menjadikan Emo tidak hanya lebih efisien dalam menyerang, tetapi juga lebih sulit untuk dikalahkan karena jarang berada dalam situasi yang berisiko tinggi.

Di sisi lain, NotThatGuy beberapa kali berhasil meraih posisi kedua dan bahkan mampu mendekati skor Emo dalam beberapa pertandingan. Keunggulan NotThatGuy dibandingkan *bot* lain adalah pola pergerakannya yang lebih acak dan dinamis sehingga sulit diprediksi. Meskipun ia juga menerapkan strategi *locking* dalam menyerang, gerakannya yang dinamis seringkali membuatnya terhindar dari pertempuran yang terlalu intens. Hal ini memberikan keuntungan tersendiri, karena ia lebih jarang menjadi sasaran utama dan memiliki peluang yang lebih besar untuk bertahan hingga akhir pertandingan.

Sementara itu, Derik dan Botol berada dalam keadaan yang kurang menguntungkan. Keduanya lebih sering mengalami kekalahan karena strategi yang mereka miliki mempunyai kelemahan terhadap lawan yang dihadapi (terutama Emo). Dengan pola permainan yang cenderung statis dan berfokus pada satu target, mereka lebih mudah dieksploitasi oleh *bot* lain. Meskipun dalam beberapa kesempatan mereka masih bisa meraih kemenangan, terutama jika mendapatkan *spawn point* yang lebih menguntungkan sehingga mereka dapat menarget Emo dari awal pertandingan dan mendapatkan keuntungan yang lebih besar, peluang untuk berada di posisi tersebut lebih kecil dibandingkan kemungkinan terjebak di tempat yang kurang strategis. Ketika mendapatkan posisi *spawn* yang buruk, mereka cenderung langsung terlibat dalam pertempuran jarak dekat yang sengit sejak awal, sehingga sulit bagi mereka untuk bertahan dalam jangka waktu yang lama.

BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada tugas besar ini, telah dilakukan implementasi dan pengujian terhadap empat alternatif strategi *greedy* dalam permainan Robocode Tank Royale. Dari hasil percobaan dan analisis, dapat disimpulkan bahwa penggunaan algoritma *greedy* cukup efektif dalam menentukan aksi bot secara cepat berdasarkan kondisi permainan saat itu. Algoritma *greedy* bekerja dengan memilih keputusan terbaik pada setiap langkah tanpa mempertimbangkan dampak jangka panjang. Dalam konteks permainan ini, strategi *greedy* memungkinkan *bot* untuk mengambil tindakan seperti menembak, menghindar, atau bergerak ke posisi yang lebih menguntungkan berdasarkan data yang tersedia pada saat itu. Dari hasil pengujian, ditemukan bahwa strategi *bot* Emo memberikan hasil yang paling optimal dibandingkan strategi lainnya. Strategi ini unggul dalam beberapa aspek, sebagaimana Emo selalu bergerak menuju area dengan tingkat ancaman yang paling rendah berdasarkan posisi *bot* lain di arena. Dengan menghindari pertempuran jarak dekat dan memilih posisi yang aman, Emo dapat bertahan lebih lama untuk memperoleh poin *survival* yang tinggi, serta mendapatkan lebih banyak peluang untuk menyerang lawan yang sedang bertarung. Namun, terdapat beberapa keterbatasan yang masih ditemukan dalam implementasi strategi *greedy* ini, yakni:

1. Kurangnya perencanaan jangka panjang

Karena algoritma *greedy* hanya mempertimbangkan keuntungan langsung, *bot* terkadang terjebak dalam situasi yang merugikan dalam jangka panjang.

2. Ketidakmampuan mengantisipasi pergerakan musuh secara kompleks

Bot cenderung tidak memiliki mekanisme prediksi yang baik terhadap pola pergerakan lawan, sehingga seringkali kesulitan menghadapi bot dengan strategi yang lebih adaptif.

3. Keterbatasan dalam pemindaian radar

Strategi *greedy* yang diterapkan belum sepenuhnya memanfaatkan mekanisme *scanning* yang optimal, sehingga bot terkadang mengalami kesulitan dalam mendeteksi musuh yang bergerak cepat atau berada di luar jangkauan radar utama.

Hasil evaluasi dari beberapa uji coba yang dilakukan menunjukkan bahwa *bot* Emo menghasilkan skor rata-rata 21.607,5; dengan tingkat kemenangan 100%. Namun, strategi ini masih bisa dikembangkan lebih lanjut untuk menghadapi lawan dengan pola pergerakan yang lebih kompleks atau *bot* yang memiliki strategi pertahanan lebih kuat. Secara keseluruhan, penerapan algoritma *greedy* dalam Robocode Tank Royale memberikan hasil yang cukup baik dalam skenario tertentu, terutama dalam situasi yang membutuhkan pengambilan keputusan cepat dan efisien. Namun, untuk mencapai performa yang lebih baik, strategi *greedy* perlu dikombinasikan dengan pendekatan lain agar dapat mengatasi kelemahan utamanya, yaitu kurangnya pertimbangan terhadap konsekuensi jangka panjang.

5.2 Saran

Berdasarkan hasil pengujian dan analisis terhadap implementasi strategi *greedy*, terdapat beberapa saran pengembangan untuk meningkatkan kinerja *bot* di masa mendatang.

1. Penggabungan dengan Algoritma Lain untuk Meningkatkan Performa

Saat ini, bot hanya menggunakan algoritma *greedy* sebagai dasar pengambilan keputusan. Namun, algoritma ini memiliki kelemahan dalam mengantisipasi skenario jangka panjang. Oleh karena itu, disarankan untuk mengkombinasikan algoritma *greedy* dengan pendekatan lain, contohnya *Dynamic Programming* untuk menyimpan hasil keputusan sebelumnya dan menggunakannya kembali, sehingga *bot* dapat belajar dari pertempuran yang telah terjadi. Dengan menggabungkan algoritma lain, bot menjadi lebih adaptif terhadap berbagai kondisi dan tidak hanya bergantung pada keputusan lokal yang menjadi sifat algoritma *greedy*.

2. Optimasi Pemilihan Aksi Berdasarkan Kondisi Pertempuran

Dalam implementasi saat ini, *bot* memilih tindakan berdasarkan pendekatan *greedy* tanpa mempertimbangkan *trade-off* antara menyerang dan bertahan. Oleh karena itu, strategi ini dapat dikaji lebih lanjut untuk memungkinkan bot menjadi lebih fleksibel dalam memilih strategi berdasarkan kondisi pertempuran, bukan hanya sekadar mengambil keputusan terbaik dalam jangka pendek.

3. Melakukan Lebih Banyak Pengujian terhadap Variasi Lawan

Sejauh ini, pengujian *bot* telah dilakukan dengan skenario tertentu. Namun, untuk memastikan bahwa strategi yang digunakan benar-benar efektif, disarankan untuk melakukan:

- Pengujian dengan berbagai tipe *bot*

Menghadapi bot dengan strategi yang berbeda (agresif, defensif, atau adaptif) untuk mengevaluasi kelemahan *bot* terhadap lawan yang lebih beragam.

- Simulasi dengan jumlah lawan yang lebih banyak

Menguji bot dalam pertempuran dengan lebih banyak musuh untuk melihat bagaimana strategi *greedy* bekerja dalam situasi yang lebih kompleks.

Melalui pengujian yang lebih luas, dapat diperoleh wawasan yang lebih mendalam mengenai kekuatan dan kelemahan bot, serta strategi apa yang perlu dikembangkan lebih lanjut.

LAMPIRAN

Tautan Repository Github

https://github.com/albertchriss/Tubes1_pikirnanti

Tautan Video

https://youtu.be/K_sIZoaVcMM?si=_bGTz0NUU30X2fvI

Hasil Akhir 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.	✓	

DAFTAR PUSTAKA

Munir, Rinaldi. (2025). “Algoritma Greedy (Bagian 1). Institut Teknologi Bandung. Diakses 19 Maret 2025.

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/04-Algoritma-Greedy-\(2025\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/04-Algoritma-Greedy-(2025)-Bag1.pdf)

Munir, Rinaldi. (2025). “Algoritma Greedy (Bagian 2). Institut Teknologi Bandung. Diakses 19 Maret 2025.

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/05-Algoritma-Greedy-\(2025\)-Bag2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/05-Algoritma-Greedy-(2025)-Bag2.pdf)

Munir, Rinaldi. (2025). “Algoritma Greedy (Bagian 3). Institut Teknologi Bandung. Diakses 19 Maret 2025.

[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)

Robocode Developers. (n.d.). “Getting started with Robocode Tank Royale”. Robocode Tank Royale. Diakses 21 Maret 2025.

<https://robocode-dev.github.io/tank-royale/tutorial/getting-started.html#bots-and-teams>

Robocode Developers. (n.d.). “Robocode Tank Royale repository”. GitHub. Diakses 21 Maret 2025.

<https://github.com/robocode-dev/tank-royale>