

LAPORAN TUGAS BESAR 1

IF2211 STRATEGI ALGORITMA

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



Disusun oleh :

Alfian Hanif Fitria Yustanto	13523073
Carlo Angkisan	13523091
Azfa Radhiyya Hakim	13523115

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

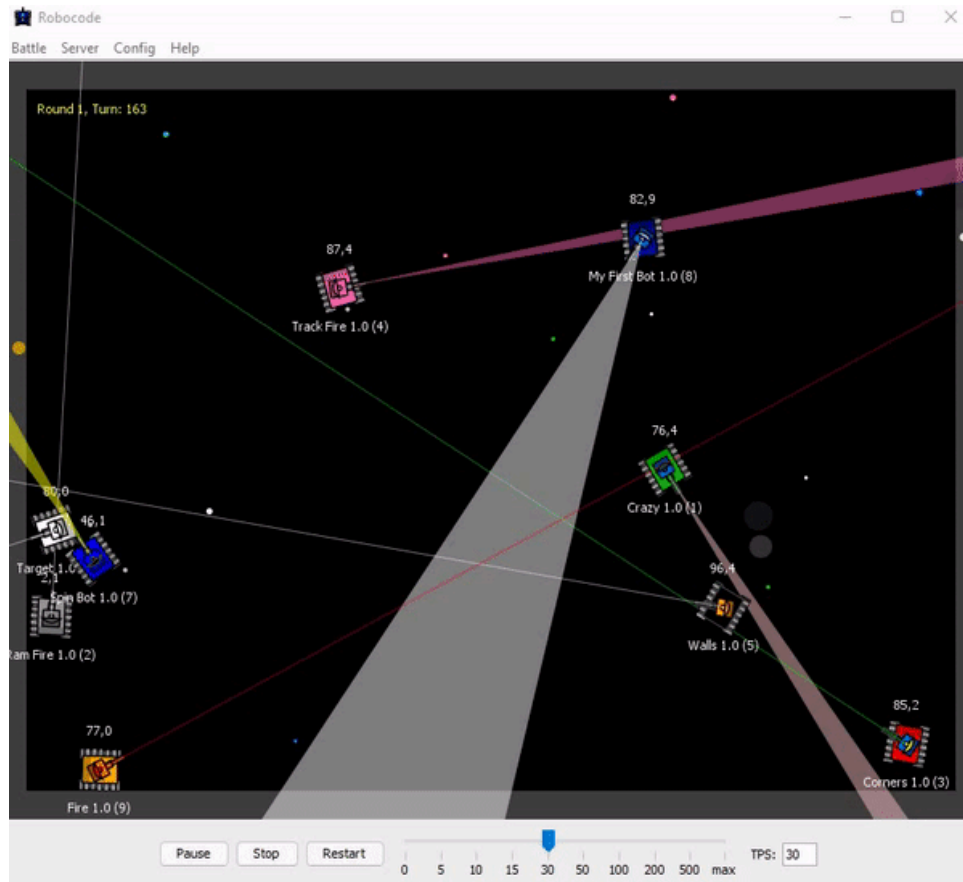
2025

DAFTAR ISI

DAFTAR ISI.....	2
BAB I.....	3
BAB II.....	9
2.1 Algoritma Greedy.....	9
2.2 Game Engine Robocode Tank Royale.....	9
2.3 Alur Menjalankan Program.....	10
2.4 Alur Pengembangan Logika Bot.....	14
BAB III.....	16
3.1 Strategi Greedy Berdasarkan Energi Musuh Terendah (Lowest Energy Enemy).....	16
3.2 Strategi Greedy Berdasarkan Jarak Musuh Terdekat (Nearest Enemy).....	17
3.3 Strategi Greedy Berdasarkan Pusat Keramaian (Crowd Evasion).....	18
3.4 Strategi Greedy Berdasarkan Satu Target Musuh (Single Target Enemy).....	19
3.5 Strategi Greedy yang Diimplementasikan Pada Program.....	20
BAB IV.....	22
4.1 Implementasi dalam Pseudocode.....	22
4.2 Struktur Data, Fungsi, dan Prosedur Strategi Greedy yang Dipilih (Crowd Evasion).....	29
4.3 Analisis dan Pengujian.....	32
BAB V.....	36
5.1 Kesimpulan.....	36
5.2 Saran.....	36
LAMPIRAN.....	37
Tautan Repository Github.....	37
Tautan Video.....	37
Tabel Kelengkapan Spesifikasi.....	37
DAFTAR PUSTAKA.....	38

BAB I

DESKRIPSI TUGAS



Gambar 1 Robocode Tank Royale

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

Pada Tugas Besar pertama Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Tentunya mahasiswa harus menggunakan **strategi greedy** dalam membuat bot ini.

Komponen-komponen dari permainan ini antara lain:

1. Rounds dan Turns

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

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

Pada setiap turn, sebuah bot dapat:

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

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

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

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

2. Batas Waktu Giliran

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

Setiap kali turn baru dimulai, penghitung waktu ulang diatur ulang dan mulai berjalan. Jika batas waktu tercapai dan bot tidak mengirimkan pergerakannya untuk turn tersebut, maka tidak ada perintah yang dikirim ke server. Akibatnya, bot akan melewati turn tersebut. Jika bot melewati turn, ia tidak akan bisa menyesuaikan gerakannya atau menembakkan senjatanya karena server tidak menerima perintah tepat waktu sebelum turn berikutnya dimulai.

3. Energi

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

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

4. Peluru

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

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

5. Panas Meriam (Gun Heat)

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

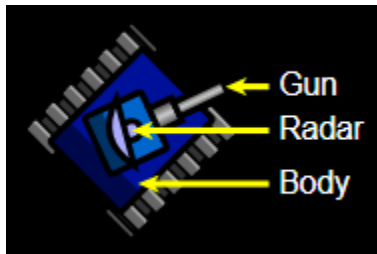
6. Tabrakan

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

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

7. Bagian Tubuh Tank

Tubuh tank terdiri dari 3 bagian:



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

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

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

8. Pergerakan

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

9. Berbelok

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

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

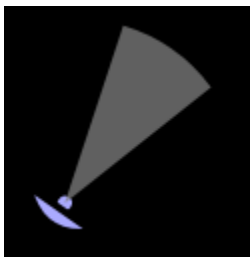
Bagian paling lambat adalah tubuh tank, yang dalam kondisi terbaik dapat berputar hingga 10 derajat per giliran. Namun, ini bergantung pada kecepatan bot saat ini. Semakin cepat bot bergerak, semakin lambat kemampuannya untuk berbelok.

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

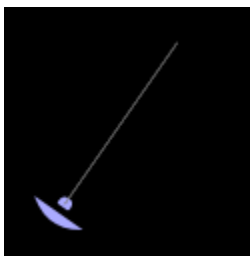
10. Pemindaian

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

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



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



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

11. Skor

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

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

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

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

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

BAB II

LANDASAN TEORI

2.1 Algoritma *Greedy*

Algoritma *greedy* adalah metode pendekatan untuk menyelesaikan masalah secara bertahap, di mana setiap langkahnya selalu memilih opsi terbaik yang tersedia saat itu tanpa mempertimbangkan dampaknya di masa depan. Prinsip utama dari metode ini adalah mengambil keputusan yang tampak paling menguntungkan pada setiap tahap dengan harapan bahwa pilihan-pilihan tersebut akan menghasilkan solusi optimal secara keseluruhan.

Algoritma ini terdiri dari beberapa elemen utama, antara lain :

1. **Himpunan Kandidat (C)**, yaitu kumpulan elemen yang dapat dipilih dalam setiap langkah, seperti simpul atau sisi dalam graf, tugas dalam penjadwalan, dan sebagainya.
2. **Himpunan Solusi (S)**, yaitu kumpulan elemen yang telah dipilih dan menjadi bagian dari solusi akhir.
3. **Fungsi Solusi**, yaitu fungsi yang berfungsi untuk menentukan apakah elemen yang telah dipilih membentuk solusi yang valid.
4. **Fungsi Seleksi**, yaitu fungsi yang menentukan elemen yang akan dipilih dari himpunan kandidat berdasarkan strategi *greedy* tertentu, yang umumnya bersifat heuristik.
5. **Fungsi Kelayakan**, yaitu fungsi yang mengevaluasi apakah elemen yang dipilih memenuhi syarat untuk dimasukkan ke dalam solusi.
6. **Fungsi Objektif**, yaitu fungsi yang digunakan untuk mencapai hasil terbaik dengan cara memaksimalkan atau meminimalkan suatu nilai tertentu.

2.2 Game Engine Robocode Tank Royale

Robocode Tank Royale adalah sebuah *game engine* berbasis pertarungan bot yang dirancang untuk menguji strategi dan algoritma dalam pertempuran tank virtual. Setiap bot dikendalikan oleh program yang dibuat oleh pengguna dan akan bertarung dalam sebuah arena dengan tujuan mengalahkan lawan melalui serangkaian aksi seperti menembak, menabrak, menghindari, dan lain-lain. Pengguna dapat membuat program bot dengan mudah melalui API yang disediakan oleh *game engine*. API ini memungkinkan bot untuk mengakses berbagai fungsi seperti pergerakan, penembakan, dan pendeteksian lawan tanpa perlu mengelola aspek teknis dari *game engine* secara langsung.

Untuk tugas besar ini, *game engine* yang akan digunakan sudah dimodifikasi oleh asisten untuk memudahkan pengamatan pertandingan, antara lain perubahan tema GUI menjadi light theme, penampilan skor dan energi masing-masing bot disamping area permainan, dan penyesuaian durasi pertarungan yang berakhir ketika banyaknya turn

sudah mencapai batas tertentu. Berikut adalah *starter pack* yang diperlukan untuk menjalankan Robocode Tank Royale :

- *Game Engine* dan *Bot Starter Pack*:
<https://github.com/Ariel-HS/tubes1-if2211-starter-pack>
- Referensi API:
[Apis.zip](#)

Berikut garis besar cara kerja Robocode Tank Royale :

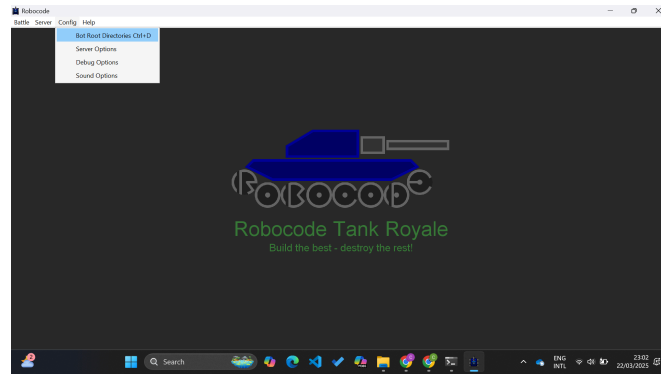
1. Pengguna bot menggunakan bahasa pemrograman C# dan menghubungkannya ke *game engine* melalui API.
2. Bot menerima informasi tentang lingkungan sekitarnya, seperti posisi lawan, energi, dan lain-lain melalui radar pemindaian.
3. Bot menentukan tindakan berdasarkan algoritma yang telah diprogram oleh pengguna, seperti bergerak, menembak, atau menghindari.
4. *Game engine* mengelola pertarungan dengan memperbarui posisi bot, mendeteksi tabrakan, dan menghitung skor berdasarkan aksi yang dilakukan.
5. Pertarungan berlangsung hingga salah satu bot menang atau batas *turn* yang telah ditentukan tercapai.
6. Hasil pertandingan ditampilkan, termasuk skor akhir dan statistik untuk masing-masing bot.

2.3 Alur Menjalankan Program

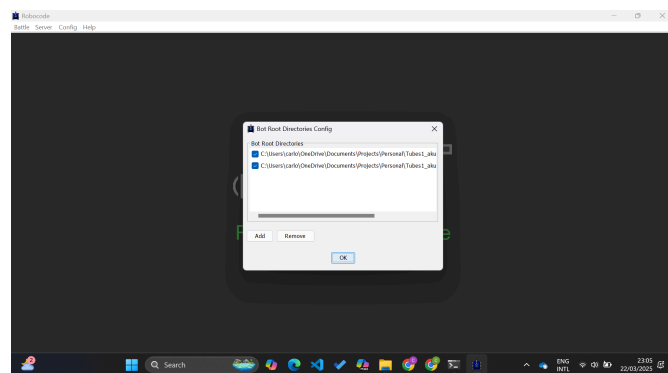
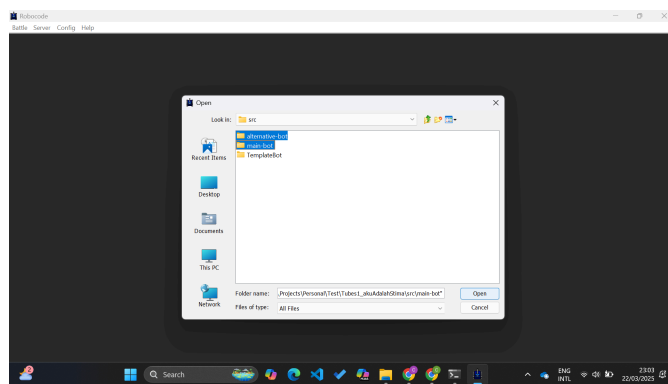
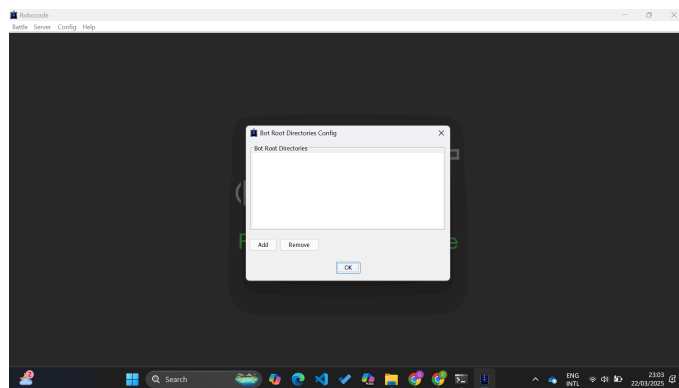
Berikut adalah langkah-langkah untuk menjalankan program:

1. Unduh dan instal .NET 6.0 sesuai dengan sistem operasi yang digunakan melalui link berikut:
<https://dotnet.microsoft.com/en-us/download/dotnet/6.0>
2. Unduh dan instal Java Development Kit (JDK) melalui link berikut:
<https://www.oracle.com/java/technologies/downloads/>
3. Unduh file “robocode-tankroyale-gui-0.30.0.jar”, yaitu *game engine* yang telah dimodifikasi asisten, melalui link berikut:
<https://github.com/Ariel-HS/tubes1-if2211-starter-pack>
4. Unduh dan ekstrak sample bot yang telah disiapkan melalui link berikut:
https://github.com/AlfianHanifFY/Tubes1_akuAdalahStima/releases/tag/v1.0
5. Buka terminal dan navigasikan ke lokasi tempat file “robocode-tankroyale-gui-0.30.0.jar” tersimpan, kemudian jalankan perintah berikut:

```
java -jar robocode-tankroyale-gui-0.30.0.jar
```
6. Lakukan konfigurasi booter sebagai berikut :
 - 1) Klik tombol "Config", kemudian pilih "Bot Root Directories".

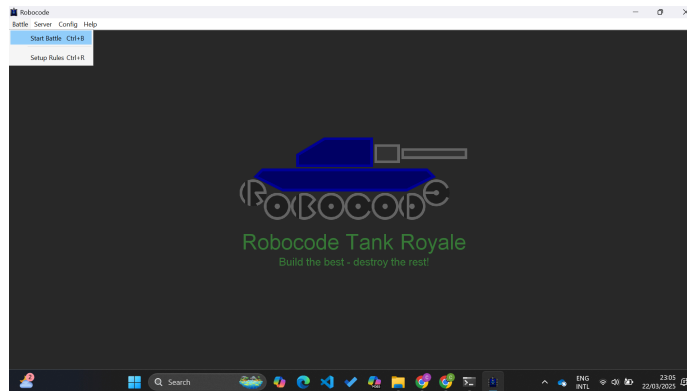


- 2) Tambahkan *sample bot* yang telah diunduh dengan mengklik tombol "Add", lalu pilih folder main-bot dan/atau alternative-bot yang terdapat dalam folder src. Kemudian, klik "OK" untuk menyimpan konfigurasi.

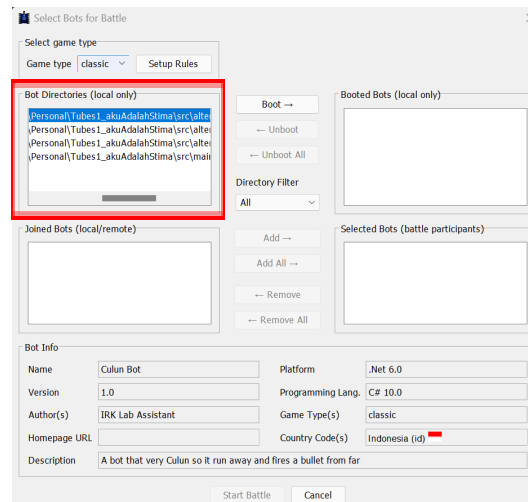


7. Untuk menjalankan pertandingan dapat dilakukan sebagai berikut:

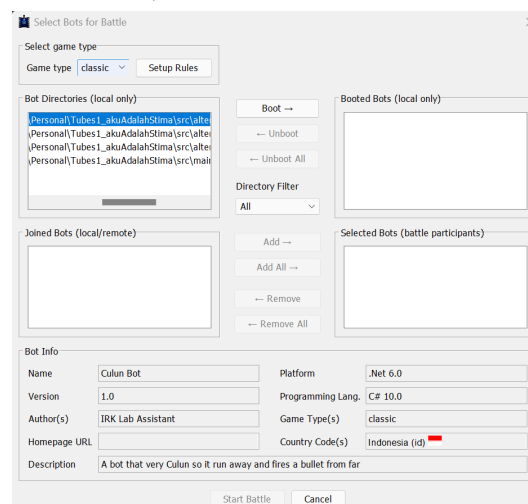
1) Klik tombol “Battle”, kemudian pilih “Start Battle”



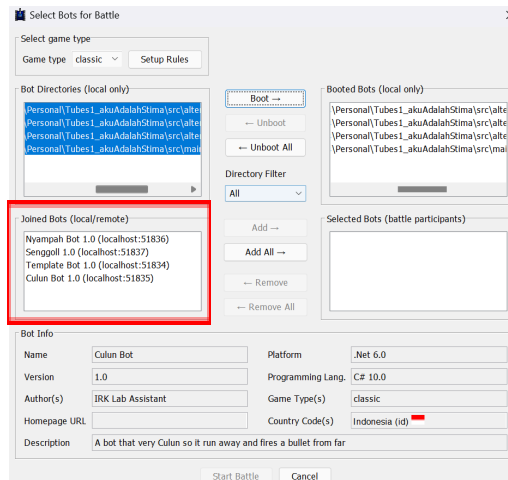
2) Bot-bot yang telah dikonfigurasi akan muncul pada kotak sebelah kiri-atas.



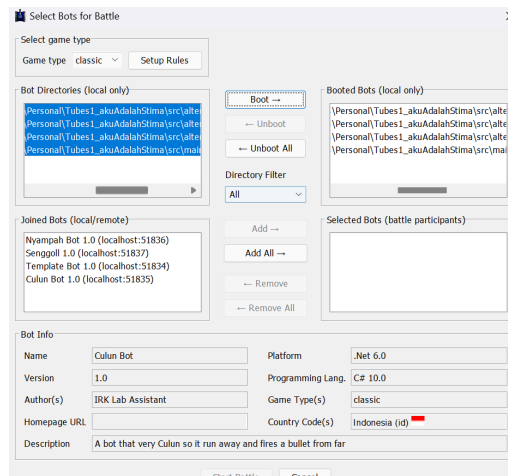
3) Pilih bot yang akan dimainkan, lalu klik "Boot →" untuk memuatnya.



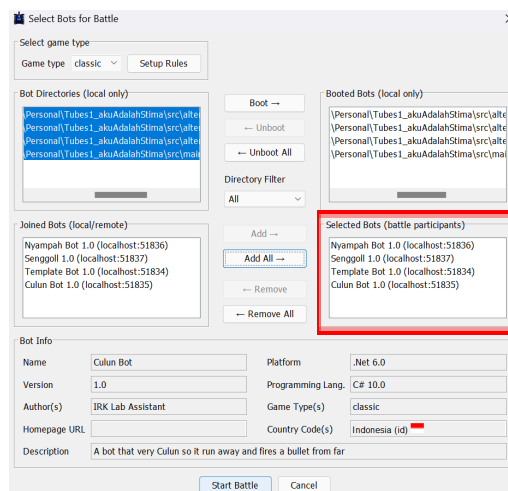
4) Bot yang berhasil dimuat akan muncul di kotak sebelah kiri-bawah.



5) Tambahkan bot ke dalam arena dengan mengklik "Add→" atau gunakan "Add All→" untuk memasukkan semua bot.

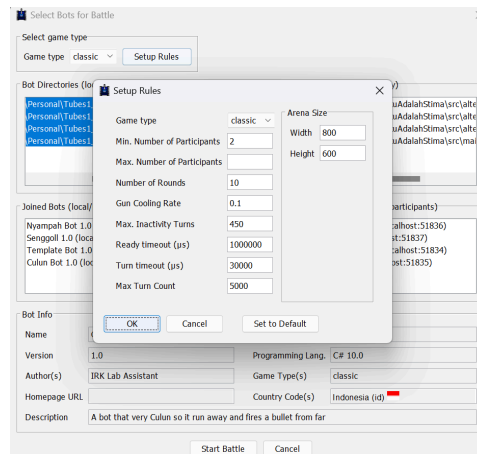


6) Bot yang telah ditambahkan akan muncul di kotak kanan-bawah, dan permainan dapat dimulai dengan menekan tombol "Start Battle".





8. *Game engine* ini juga menyediakan fitur "Setup Rules" untuk menyesuaikan aturan permainan, seperti jumlah turn dan round, ukuran arena, dan pengaturan lainnya.



2.4 Alur Pengembangan Logika Bot

Berikut adalah langkah-langkah untuk mengembangkan bot dalam Robocode Tank Royale:

1. Unduh *template bot* melalui link berikut:
<https://github.com/Ariel-HS/tubes1-if2211-starter-pack>
2. Di dalamnya terdapat file dengan ekstensi `.cs`, yang merupakan tempat untuk menuliskan logika bot.
3. *Template bot* ini sudah memiliki kerangka dasar yang harus ada dalam setiap bot yaitu sebagai berikut:

- *Import library*

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

`Robocode.TankRoyale.BotApi` berisi API untuk mengontrol bot dan `Robocode.TankRoyale.BotApi.Events` berisi *event handler* yang menangani berbagai kejadian dalam permainan, seperti ketika bot melihat lawan

atau terkena tembakan. Untuk lebih lengkapnya terkait *event handler* dan hal-hal apa saja yang dapat dilakukan dapat melihat referensi API berikut:

[Apis.zip](#)

- Kelas utama dan *method* `main`

```
public class BotTemplate : Bot
{
    static void Main(string[] args)
    {
        new BotTemplate().Start();
    }
}
```

Kelas `BotTemplate : Bot` merupakan kelas utama yang akan dijalankan dan mewarisi fungsi dari kelas `Bot`. Di dalam kelas ini, pengguna dapat memanfaatkan *method event listener* maupun membuat *method* baru untuk mengembangkan logika bot sesuai kebutuhan. Sedangkan, fungsi `Main(string[] args)` berperan sebagai titik awal eksekusi program, yang bertugas untuk menjalankan bot.

- Konstruktor

```
BotTemplate() : base(BotInfo.FromFile("BotTemplate.json")) { }
```

Konstruktor ini membaca konfigurasi bot dari file `BotTemplate.json`, yang berisi informasi bot seperti nama bot, nama pembuat, deskripsi, dan lain-lain.

- *Method* `Run()`

```
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
    }
}
```

Method `Run()` merupakan fungsi utama yang dijalankan saat ronde dimulai dan menjadi tempat utama untuk menuliskan logika bot. Pada bagian awal *method* ini, pengguna dapat menyesuaikan tampilan bot dengan mengatur warna berbagai komponennya, seperti badan, turret, radar, dan peluru. Setelah itu, program akan memasuki loop utama `while (IsRunning)`, yang memastikan bot terus berjalan selama pertandingan berlangsung. Dalam loop ini, pengguna dapat mengimplementasikan strategi pergerakan, penyerangan, dan respons terhadap berbagai kondisi dalam permainan.

BAB III

APLIKASI STRATEGI GREEDY

3.1 Strategi *Greedy* Berdasarkan Energi Musuh Terendah (*Lowest Energy Enemy*)

Strategi *Greedy Lowest Energy Enemy* adalah strategi yang mengutamakan penyerangan musuh berdasarkan tingkat energi musuh terendah. Prinsip utama strategi ini adalah memindai seluruh musuh yang ada dalam arena, dan setiap kali pemindaian selesai dilakukan (sebesar 360 derajat), maka data yang diperoleh akan dibandingkan untuk mendapatkan bot dengan energi terendah. Bot dengan energi terendah kemudian akan diserang dengan menyesuaikan kekuatan tembakan berdasarkan jarak untuk efisiensi energi dan keakuratan.

1. Pemetaan Elemen *Greedy*

- a. **Himpunan Kandidat** : Semua musuh yang terdeteksi dalam radar pemindaian.
- b. **Himpunan Solusi** : Musuh dengan energi terendah yang dipilih untuk diserang dalam satu putaran pemindaian.
- c. **Fungsi Solusi** : Memilih satu musuh dari himpunan kandidat yang memiliki energi terendah dan valid.
- d. **Fungsi Seleksi** : Menentukan musuh dengan energi terendah setelah pemindaian 360 derajat.
- e. **Fungsi Kelayakan** : Memastikan musuh masih berada dalam jangkauan tembakan dan dapat diserang.
- f. **Fungsi Objektif** : Memaksimal skor dengan fokus pada *Bullet Damage Bonus*, yaitu bonus apabila peluru berhasil membunuh bot musuh.

2. Analisis Efisiensi Solusi

Strategi ini memiliki efisiensi yang baik. Pemindaian radar 360 derajat merupakan operasi konstan dengan kompleksitas $O(1)$. Setiap kali bot musuh terdeteksi, informasinya disimpan dalam *dictionary* dengan kompleksitas $O(1)$ untuk operasi penyimpanan. Bagian terpenting dari algoritma *greedy* ini adalah iterasi terhadap seluruh bot musuh yang terdeteksi untuk menemukan bot dengan energi terendah, yang memiliki kompleksitas $O(n)$, di mana n adalah jumlah musuh yang terdeteksi. Semakin banyak musuh dalam arena, semakin lama waktu yang dibutuhkan untuk menentukan target. Secara keseluruhan, kompleksitas waktu algoritma ini adalah $O(n)$, yang tergolong sangat efisien karena bersifat linear. Tidak ada algoritma *sorting* atau struktur data kompleks yang digunakan, sehingga cocok untuk lingkungan *real-time* seperti Robocode, di mana keputusan harus dibuat dengan cepat.

3. Analisis Efektivitas Solusi

Strategi ini efektif ketika musuh dengan energi rendah mudah dieliminasi, terutama dalam pertempuran akhir atau saat lawan bergerak lambat atau diam,

sehingga bot dapat mengurangi jumlah musuh dengan cepat. Namun, strategi ini menjadi tidak efektif jika musuh dengan energi rendah memiliki mobilitas tinggi dan berada pada posisi yang jauh, serta sulit ditembak atau ketika ada bot dengan energi lebih tinggi yang lebih agresif dan berbahaya. Selain itu, dalam tahap awal permainan dengan banyak musuh, fokus pada target berenergi rendah bisa kurang optimal dibandingkan strategi lain seperti menyerang musuh terdekat atau menghindari pertempuran langsung.

3.2 Strategi *Greedy* Berdasarkan Jarak Musuh Terdekat (*Nearest Enemy*)

Strategi *Greedy Nearest Enemy* adalah strategi yang memfokuskan penyerangan musuh berdasarkan jarak terdekat. Prinsip utama strategi ini adalah memindai musuh yang ada dalam arena dan mengumpulkan data yang didapat dalam setiap satu putaran radar penuh (360 derajat). Ketika radar mendeteksi keberadaan robot, data disimpan ke dalam suatu *dictionary* yang berisi *Distance*, *Bearing*, dan *Energy*. Dalam setiap pemindaian, tidak ada jaminan bahwa setiap robot yang ada di arena terdeteksi. Jika tidak ada robot yang terpindai, maka radar akan kembali berputar dan memindai ulang. Jika terdapat satu robot yang terpindai, maka robot tersebutlah yang terpilih untuk diserang. Jika terdapat dua atau tiga robot terpindai, maka akan dibandingkan *Distance* dari masing-masing robot. Setelah di *lock target*, robot ini akan berputar sejauh besar *Bearing* yang tersimpan. Robot kemudian akan mendekati target dan menembak. Jika jarak dianggap terlalu jauh (> 300), maka robot tidak akan menembak. Jika jarak berada di atas 200, maka akan ditembak dengan *power* 1. Jika jarak berada di antara 100 dan 200, maka akan ditembak dengan *power* 2. Jika jarak berada di bawah 100, maka akan ditembak dengan *power* 3. Setelah melakukan tembakan, robot kemudian akan memulai pemindaian kembali dan melakukan hal yang sama hingga ronde berakhir.

1. Pemetaan Elemen *Greedy*

- a. **Himpunan Kandidat** : Semua musuh yang terdeteksi dalam radar pemindaian.
- b. **Himpunan Solusi** : Musuh dengan jarak terdekat yang dipilih untuk diserang dalam satu putaran pemindaian.
- c. **Fungsi Solusi** : Memilih satu musuh dari himpunan kandidat yang memiliki jarak terdekat dan valid.
- d. **Fungsi Seleksi** : Menentukan musuh dengan jarak terdekat setelah pemindaian 360 derajat.
- e. **Fungsi Kelayakan** : Memastikan musuh masih berada dalam jangkauan tembakan dan dapat diserang.
- f. **Fungsi Objektif** : Mengoptimalkan skor dengan meningkatkan serangan secara maksimal dengan selalu menargetkan musuh terdekat.

2. Analisis Efisiensi Solusi

Strategi ini memiliki efisiensi yang baik. Pemindaian radar 360 derajat berlangsung sebagai operasi konstan dengan kompleksitas $O(1)$. Ketika bot musuh

terdeteksi, informasinya disimpan dalam *dictionary* dengan kompleksitas waktu akses $O(1)$. Bagian utama dari algoritma ini adalah menelusuri seluruh bot yang terdeteksi untuk menemukan musuh dengan jarak paling dekat, yang memiliki kompleksitas $O(n)$, di mana n adalah jumlah musuh yang terdeteksi. Semakin banyak musuh dalam arena, semakin lama waktu yang dibutuhkan untuk menentukan target. Secara keseluruhan, kompleksitas algoritma ini adalah $O(n)$, yang efisien karena bersifat linear.

3. Analisis Efektivitas Solusi

Efektifitas algoritma ini dapat ditentukan dari tingkat keberhasilan robot dalam menarget musuh. Algoritma ini sangat kuat untuk menyerang robot lain yang tidak bergerak terlalu aktif sebab posisi yang tersimpan ketika terpindai radar tidak akan berbeda jauh ketika robot melakukan penyerangan. Sebaliknya, algoritma ini akan relatif lebih sulit untuk menyerang robot yang bergerak sangat aktif (contoh: *crazy*) karena posisi yang selalu berubah.

3.3 Strategi *Greedy* Berdasarkan Pusat Keramaian (*Crowd Evasion*)

Strategi *Greedy Crowd Evasion* adalah strategi *greedy* yang mengutamakan keselamatan Bot dengan menghindari daerah yang memiliki kepadatan bot musuh tertinggi. Bot akan menghindari keramaian dengan memulai pemindaian 360 derajat guna mencatat lokasi musuh, lalu menghitung rata-rata koordinat mereka untuk menentukan pusat keramaian. Selanjutnya, bot akan bergerak ke arah titik kabur yaitu hasil pencerminan titik keramaian terhadap pusat peta permainan dan melakukan penembakkan dari titik kabur. Bot akan kembali menghindari keramaian jika Bot terkena serangan atau ada bot musuh yang mendekat.

1. Pemetaan Elemen *Greedy*

- a. **Himpunan Kandidat** : Semua kemungkinan lokasi di medan perang yang dapat diproses menjadi titik tujuan bot.
- b. **Himpunan Solusi** : Titik kabur yang dipilih berdasarkan pencerminan pusat keramaian terhadap tengah peta.
- c. **Fungsi Solusi** : Menentukan titik kabur yang optimal untuk menjauh dari pusat keramaian guna meningkatkan peluang bertahan.
- d. **Fungsi Seleksi** : Memilih titik yang merupakan hasil pencerminan posisi rata-rata musuh terhadap tengah peta.
- e. **Fungsi Kelayakan** : Memastikan titik kabur yang dipilih merupakan daerah aman untuk melakukan penyerangan jarak jauh.
- f. **Fungsi Objektif** : Meminimalkan risiko terkena serangan dengan menjauh dari keramaian sambil tetap dapat menyerang musuh secara efektif hal ini dapat memaksimalkan *survival score* dan *bullet score*.

2. Analisis Efisiensi Solusi

Strategi ini memiliki efisiensi yang baik karena pemilihan titik kabur dilakukan menggunakan operasi aritmatika sederhana, yaitu perhitungan rata-rata posisi musuh dan penentuan titik berlawanan dengan kompleksitas $O(1)$, sehingga tidak membebani perhitungan. Selain itu, pemrosesan setiap bot juga berjalan dalam $O(1)$ untuk setiap musuh yang terdeteksi. Secara keseluruhan, algoritma ini memiliki kompleksitas $O(n)$, di mana n adalah jumlah musuh yang terdeteksi, tetapi dengan *overhead* minimal karena tidak melibatkan pengurutan atau struktur data yang kompleks. Pergerakan bot dan mekanisme penghindaran serangan tetap dalam batas waktu yang dapat ditangani secara real-time, sehingga bot dapat merespons dengan cepat terhadap perubahan kondisi pertempuran. Mengandalkan data yang diperoleh langsung dari pemindaian, strategi ini mampu beradaptasi dengan kondisi permainan.

3. Analisis Efektivitas Solusi

Dalam hal efektivitas, strategi ini membuat bot mampu menghindari pusat keramaian, sehingga mengurangi kemungkinan terkena serangan dari banyak musuh sekaligus. Pola pergerakan melingkar saat menyerang juga meningkatkan peluang menghindari tembakan musuh, sementara tetap dapat menyerang target yang terdeteksi radar. Namun, Bot memiliki kelemahan yaitu ketika musuh tersebar merata, pencerminan terhadap tengah peta mungkin tidak selalu menghasilkan titik kabur yang optimal. Meskipun begitu, strategi ini tetap efektif dalam situasi di mana musuh cenderung berkumpul di satu daerah.

3.4 Strategi *Greedy* Berdasarkan Satu Target Musuh (*Single Target Enemy*)

Strategi *Greedy Single Target Enemy* adalah strategi *greedy* yang mengutamakan penyerangan yang terfokus kepada satu robot yang menjadi target. Strategi ini bekerja dengan cara memindai arena untuk mencari musuh. Ketika berhasil mendeteksi satu target, bot akan menyimpan ID musuh tersebut sebagai target utama kemudian mengejar dan menembaknya pada jarak tertentu. Jika target musuh berhasil dihancurkan, bot akan kembali memindai arena untuk mencari lawan baru dan mengulangi proses yang sama.

1. Pemetaan Elemen *Greedy*

- a. **Himpunan Kandidat** : Semua musuh yang terdeteksi di arena.
- b. **Himpunan Solusi** : Satu musuh yang dipilih sebagai target utama.
- c. **Fungsi Solusi** : Menentukan apakah musuh yang dipilih masih ada dan dapat diserang.
- d. **Fungsi Seleksi** : Memilih satu musuh pertama yang terdeteksi atau mencari target baru jika musuh sebelumnya telah hancur.
- e. **Fungsi Kelayakan** : Memastikan musuh yang dipilih masih ada dan dalam jangkauan serangan.
- f. **Fungsi Objektif** : Mengeliminasi target secepat mungkin dengan mengejar dan menembak pada jarak optimal.

2. Analisis Efisiensi Solusi

Strategi ini memiliki efisiensi yang sangat baik dengan kompleksitas waktu $O(1)$ untuk semua operasi utamanya. Algoritma ini tidak membutuhkan struktur data kompleks atau iterasi karena hanya melacak satu target pada satu waktu menggunakan beberapa variabel sederhana. Tidak ada perhitungan yang melibatkan iterasi, sorting, atau struktur data kompleks yang dapat meningkatkan kompleksitas. Bahkan saat mencari target baru, kompleksitasnya tetap $O(1)$ karena hanya melibatkan penugasan variabel sederhana. Secara keseluruhan, algoritma ini sangat ringan secara komputasi dan ideal untuk respons *real-time*.

3. Analisis Efektivitas Solusi

Dalam hal efektivitas, bot diharapkan dengan cepat mengeliminasi musuh satu per satu, membuatnya kuat dalam pertarungan individu. Namun, strategi ini memiliki kelemahan jika bot terlalu fokus pada satu target tanpa mempertimbangkan musuh lain yang dapat menyerang dari arah berbeda. Selain itu, jika target terus bergerak cepat atau memiliki strategi penghindaran, bot mungkin kesulitan mengejar dan menembak secara efektif. Secara keseluruhan, strategi ini sangat baik untuk duel atau situasi dengan sedikit lawan.

3.5 Strategi Greedy yang Diimplementasikan Pada Program

Setelah mempertimbangkan berbagai alternatif strategi *greedy* yang telah dijelaskan pada bagian 3.1 hingga 3.4 serta melakukan serangkaian pengujian, kami memutuskan untuk mengimplementasikan strategi *Greedy* Berdasarkan Pusat Keramaian (*Crowd Evasion*) sebagai bot utama kami. Strategi ini dipilih karena mampu meningkatkan peluang bertahan dengan menghindari area yang memiliki kepadatan musuh tertinggi. Dalam Robocode Tank Royale, daya tahan bot merupakan hal yang sangat penting karena terdapat sistem penilaian memberikan poin berdasarkan *Survival Score* dan *Last Survival Bonus*.

Pemilihan ini didasarkan pada beberapa pertimbangan utama. Pertama, dibandingkan dengan strategi lain seperti *Lowest Energy Enemy* atau *Nearest Enemy*, strategi ini lebih bersifat defensif dan mengurangi risiko bot tereliminasi lebih awal dengan menghindari pertempuran langsung. Kedua, strategi ini tetap memungkinkan bot untuk menyerang lawan dari posisi yang lebih aman, sehingga tetap dapat mengumpulkan poin tanpa mengambil risiko berlebih. Ketiga, dari segi efisiensi komputasi, strategi ini memiliki kompleksitas $O(n)$ karena hanya memerlukan pemindaian radar dan perhitungan sederhana untuk menentukan titik kabur, tanpa memerlukan iterasi yang kompleks atau algoritma sorting tambahan.

Strategi ini diharapkan dapat menunjukkan performa yang baik dalam situasi di mana musuh cenderung berkumpul di satu area. Bot dapat secara otomatis menjauh dari ancaman dan tetap menyerang lawan dari jarak yang lebih aman. Namun, dalam kondisi di mana musuh tersebar merata, strategi ini dinilai kurang optimal karena titik kabur yang

dihitung tidak selalu memberikan posisi terbaik. Meskipun demikian, secara keseluruhan, strategi *Greedy* Berdasarkan Pusat Keramaian (*Crowd Evasion*) memberikan keseimbangan antara pertahanan dan serangan serta menunjukkan performa yang konsisten dan maksimal dalam berbagai skenario pertempuran.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi dalam Pseudocode

1. Implementasi Strategi *Greedy* Berdasarkan Energi Musuh Terendah (*Lowest Energy Enemy*)

```
Program Strategi Lowest Energy Enemy  
{ Implementasi dari NyampahBot, strategi greedy yang mencari Bot musuh  
yang memiliki energi terkecil. }
```

KAMUS

```
scannedBots : Set<(integer, ScannedBotInfo)>  
isScanning : boolean  
moved_distance : real
```

PROGRAM UTAMA

```
// Loop utama selama bot berjalan  
while (IsRunning) do  
    // Memulai pemutaran radar  
    StartNewScanCycle()  
  
    // Bot akan terus menerus memutar radar selama permainan  
    TurnRightRadar(360);  
  
    // Jika radar telah berputar secara penuh  
    if (RadarTurnRemaining = 0) then  
        // Panggil fungsi untuk menyerang target  
        FireAtWeakestTarget()  
    endif  
  
    // Targeting telah selesai dan dilanjutkan targeting berikutnya  
    FinishScanCycle()  
endwhile
```

FUNGSI DAN PROSEDUR

```
procedure OnRoundedStart(RoundStartEvent e)  
{ Prosedur awal ketika round baru dimulai }  
    PROGRAM  
        Forward(10)  
        CLEAR scannedBots
```

```
procedure StartNewScanCycle()  
{ Prosedur ketika akan dimulai pemindaian }  
    PROGRAM  
        isScanning ← true  
        CLEAR scannedBots
```

```
procedure FinishScanCycle()  
{ Prosedur ketika pemindaian telah selesai }  
    PROGRAM  
        isScanning ← false  
        scannedBots.Clear()
```

```
procedure Move()  
{ Prosedur untuk basis gerakan robot }  
    KAMUS LOKAL  
        distance : real
```

```

PROGRAM
    distance ← random() * move_distance + 20
    SetForward(distance)

procedure FireAtWeakestTarget()
{ Prosedur untuk menembak target yang telah ditentukan }
    KAMUS LOKAL
        weakestBotId : integer
        distance : real
    PROGRAM
        weakestBotId ← 1
        lowestEnergy ← MaxValue
        i traversal [0..len(scannedBots)-1]
            if (scannedBots[i].energy < lowestEnergy) then
                weakestBotId ← scannedBots[i].id
                lowestEnergy ← scannedBots[i].energy
            endif

        if (weakestBotId ≠ -1) then
            TurnLeft(scannedBots[weakestId].bearing)
            SmartFire(scannedBots[weakestId].distance)
            Move()
        endif

procedure SmartFire(input: real distance)
{ Prosedur untuk menyesuaikan kekuatan tembakan berdasarkan jarak }
    KAMUS LOKAL
        power : real
    PROGRAM
        if (distance > 300) then
            // Tidak menembak
        else if (distance > 200) then
            power ← 1
        else if (distance > 100) then
            power ← 2
        else
            power ← 3
        endif

        Fire(power)

procedure OnScannedBot(ScannedBotEvent e)
{ Procedure yang dipanggil ketika bot berhasil melakukan scan pada bot musuh }
    KAMUS LOKAL
        distance : real
        bearing : real
    PROGRAM
        if (isScanning = true) then
            distance ← DistanceTo(e.X, e.Y)
            bearing ← BearingTo(e.X, e.Y)
            ADD scannedBots[e.ScannedBotId, distance, bearing] TO scannedBots
        endif

procedure OnHitWall(HitWallEvent e)
{ Procedure yang terpanggil ketika robot menabrak dinding }
    PROGRAM
        Back(30)
        TurnRight(180)
        SetForward(30)

```

```

procedure OnHitBot(HitBotEvent e)
{ Procedure yang terpanggil ketika robot menabrak robot lain }
    PROGRAM
        Fire(3)

```

2. Implementasi Strategi *Greedy* Berdasarkan Jarak Musuh Terdekat (*Nearest Enemy*)

```

Program Strategi Nearest Enemy
{ Implementasi dari SenggolTembakBot, strategi greedy yang mencari robot musuh yang memiliki jarak terdekat. }

```

KAMUS

```

    scannedBots : Set<(integer, ScannedBotInfo)>
    isScanning : boolean
    moved_distance : real

```

PROGRAM UTAMA

```

    // Loop utama selama bot berjalan
    while (IsRunning) do
        // Memulai pemutaran radar
        StartNewScanCycle()

        // Bot akan terus menerus memutar radar selama permainan
        TurnRightRadar(360);

        // Jika radar telah berputar secara penuh
        if ((RadarTurnRemaining = 0) then
            // Panggil fungsi untuk menyerang target
            FireAtClosestTarget()
        endif

        // Targeting telah selesai dan dilanjutkan targeting berikutnya
        FinishScanCycle()
    endwhile

```

FUNGSI DAN PROSEDUR

```

procedure OnRoundedStart(RoundStartEvent e)
{ Prosedur awal ketika round baru dimulai }

```

```

    PROGRAM
        Forward(10)
        scannedBots.Clear()

```

```

procedure StartNewScanCycle()
{ Prosedur ketika akan dimulai pemindaian }

```

```

    PROGRAM
        isScanning ← true
        scannedBots.Clear()

```

```

procedure FinishScanCycle()
{ Prosedur ketika pemindaian telah selesai }

```

```

    PROGRAM
        isScanning ← false
        scannedBots.Clear()

```

```

procedure Move()
{ Prosedur untuk basis gerakan robot }

```

```

    KAMUS LOKAL
        distance : real
    PROGRAM
        distance ← random() * move_distance + 20

```



```

        SetForward(distance)

procedure FireAtClosestTarget()
{ Prosedur untuk menembak target yang telah ditentukan }
    KAMUS LOKAL
        closestID : integer
        Distance : real
    PROGRAM
        closestID ← -1
        distance ← MaxValue
        for traversal [0..len(scannedBots)-1]
            if (scannedBots[i].distance < distance) then
                closestID ← scannedBots[i].id
                distance ← scannedBots[i].distance
            endif

        if (closestID != -1) then
            TurnLeft(scannedBots[closestID].bearing)
            SmartFire(scannedBots[closestID].distance)
            Move()
        endif

procedure SmartFire(input: Real distance)
{ Prosedur untuk menyesuaikan kekuatan tembakan berdasarkan jarak }
    KAMUS LOKAL
        power : real
    PROGRAM
        if (distance > 300) then
            // Tidak menembak
        else if (distance > 200) then
            power ← 1
        else if (distance > 100) then
            power ← 2
        else
            power ← 3
        endif

        Fire(power)

procedure OnScannedBot(ScannedBotEvent e)
{ Procedure yang dipanggil ketika bot berhasil melakukan scan pada bot musuh }
    KAMUS LOKAL
        distance : real
        bearing : real
    PROGRAM
        if (isScanning = true) then
            distance ← DistanceTo(e.X, e.Y)
            bearing ← BearingTo(e.X, e.Y)
            ADD scannedBots[e.ScannedBotId, distance, bearing] TO
scannedBots
        endif

procedure OnHitWall(HitWallEvent e)
{ Procedure yang terpanggil ketika robot menabrak dinding }
    PROGRAM
        Back(30)
        TurnRight(180)
        SetForward(30)

procedure OnHitBot(HitBotEvent e)
{ Procedure yang terpanggil ketika robot menabrak robot lain }

```

```
PROGRAM
Fire(3)
```

3. Implementasi Strategi *Greedy* Berdasarkan Pusat Keramaian (*Crowd Evasion*)

Program Strategi Crowd Evasion

{ Implementasi dari CulunBot, strategi *greedy* yang mengutamakan keselamatan Bot dengan menghindari daerah yang memiliki kepadatan Bot musuh tertinggi. }

KAMUS

```
enemyX, enemyY, targetX, targetY : real
savedEnemies : integer
shouldMove : boolean
detectedBots : set
```

PROGRAM UTAMA

```
// Setup awal robot
AdjustGunForBodyTurn ← true
shouldMove ← true

// Loop utama selama bot berjalan
while (IsRunning) do
    // Bot akan terus menerus memutar radar selama permainan
    SetTurnGunRight(∞)
    // shouldMove berarti bot harus bergerak menghindari kerumunan
    if (shouldMove) then
        ResetTracking()
        EscapeFromEnemies()
        shouldMove ← false
    else
        AttackEnemies()
    endif
endwhile
```

FUNGSI dan PROSEDUR

```
procedure OnScannedBot(input : ScannedBotEvent e)
{ Procedure yang dipanggil ketika bot berhasil melakukan scan
pada bot musuh }
```

PROGRAM

```
// Proses mencatat bot dengan id unik
if (e.ScannedBotId ≠ detectedBots) then
    detectedBots ← detectedBots ∪ {e.ScannedBotId}
    enemyX ← enemyX + e.X
    enemyY ← enemyY + e.Y
    savedEnemies ← savedEnemies + 1
endif
```

```
distance ← DistanceTo(e.X, e.Y)
// jika musuh terlalu dekat maka bot akan menjauh
if (distance < 100) then
    shouldMove ← true
endif
// menembak musuh
SmartFire(distance)
Rescan()
```

```
procedure OnBotDeath(input : BotDeathEvent e)
{ Procedure yang dipanggil ketika sebuah bot musuh hancur }
PROGRAM
detectedBots ← detectedBots - {e.VictimId}
```

```

procedure EscapeFromEnemies()
{ Procedure untuk melarikan diri dari musuh yang terdeteksi }
PROGRAM
MaxSpeed ← 100
if (savedEnemies > 0) then
    // Proses penentuan titik kabur bot dengan memanfaatkan
    // pencerminan melalui titik tengah peta //
    targetX ← 800 - (enemyX / savedEnemies)
    targetY ← 600 - (enemyY / savedEnemies)
    TurnToFaceTarget(targetX, targetY)
    Forward(DistanceTo(targetX, targetY) + 5)
endif

procedure AttackEnemies()
{ Procedure untuk melakukan pola gerakan saat menyerang }
PROGRAM
SetTurnLeft(5000)
MaxSpeed ← 5
Forward(5000)

procedure ResetTracking()
{ Procedure untuk mereset data musuh yang terdeteksi }
PROGRAM
enemyX ← 0
enemyY ← 0
savedEnemies ← 0
detectedBots ← {}

procedure SmartFire(input : double distance)
{ Procedure untuk menembakkan peluru dengan kekuatan berdasarkan
  energi bot }
KAMUS LOKAL
power : real

PROGRAM
if (Energy > 50) then
    power ← 3
else if (Energy > 15) then
    power ← 2
else
    power ← 1
endif

Fire(power)

procedure TurnToFaceTarget(input : real x, real y)
{ Procedure untuk memutar bot agar menghadap ke target }
PROGRAM
TurnLeft(BearingTo(x, y))

procedure OnHitByBullet(input : HitByBulletEvent e)
{ Procedure yang dipanggil saat bot terkena peluru }
PROGRAM
EscapeFromEnemies()

procedure OnHitBot(input : HitBotEvent e)
{ Procedure yang dipanggil saat bot bertabrakan dengan bot lain }
PROGRAM
shouldMove ← true

```

```

procedure OnHitWall(input : HitWallEvent e)
  { Procedure yang dipanggil saat bot menabrak dinding }
  PROGRAM
  SetTurnRight(20000)
  MaxSpeed ← 5
  Back(50)

```

4. Implementasi Strategi *Greedy* Berdasarkan Satu Target Musuh (*Single Target Enemy*)

```

Program Strategi Single Target
{ implementasi dari GoBot, strategi greedy yang mengutamakan penyerangan
yang terfokus kepada satu robot yang menjadi target }

KAMUS
  targetId : integer
  lastId : integer
  x : integer
  targetX : real
  targetY : real

PROGRAM UTAMA
  targetId ← -1
  lastId ← -1
  x ← 1

  // setup awal robot
  AdjustRadarForGunTurn ← true
  AdjustGunForBodyTurn ← false
  MaxRadarTurnRate ← 100

  // while loop setiap turn
  while (IsRunning) do
    if (targetId = -1) then
      // Jika belum ada target, terus putar radar untuk mencari
      musuh
      SetTurnRadarRight(45)
    else
      // Kunci radar pada target
      SetTurnRadarRight((BearingTo(targetX, targetY) -
      RadarDirection) * x)

      // Kejar target
      TurnToFaceTarget(targetX, targetY)
      SetForward(DistanceTo(targetX, targetY) - 50)
    Go()

FUNGSI dan PROSEDUR
procedure OnScannedBot(input : ScannedBotEvent e)
  {Procedure yang dipanggil ketika bot berhasil melakukan scan pada
  bot musuh}
  PROGRAM
  if (targetId = -1 or targetId = lastId) then
    targetId ← e.ScannedBotId
    Output("Target ditemukan: " + targetId)
  endif

  if (e.ScannedBotId = targetId) then
    targetX ← e.X

```

```

        targetY ← e.Y
        { Menembak terus-menerus }
        SmartFire(DistanceTo(targetX, targetY))
        x ← x * 1
    endif
    Rescan();

procedure OnBotDeath(input : BotDeathEvent e)
    { Procedure yang dipanggil ketika sebuah bot musuh hancur }
    PROGRAM
    lastId ← targetId
    if (e.VictimId = targetId) then
        targetId ← -1
        Output("Target hancur, mencari target baru...")
    endif

procedure TurnToFaceTarget(input : real x, real y)
    { Procedure untuk memutar bot agar menghadap ke target }
    PROGRAM
    SetTurnLeft(BearingTo(x, y))

procedure SmartFire(input : real distance)
    { Procedure untuk menembakkan peluru dengan kekuatan yang
      disesuaikan berdasarkan energi bot}
    KAMUS LOKAL
    power : real

    PROGRAM
    if (Energy > 40) then
        power ← 3
    else if (Energy > 10) then
        power ← 2
    else if (Energy > 5) then
        power ← 1
    else
        power ← 0.1
    endif

    if (distance < 200) then
        Fire(power)
    endif

```

4.2 Struktur Data, Fungsi, dan Prosedur Strategi Greedy yang Dipilih (*Crowd Evasion*)

1. Struktur Data

Dalam pengimplementasian Strategi Greedy Berdasarkan Pusat Keramaian (*Crowd Evasion*), digunakan struktur data untuk mendukung proses deteksi musuh secara optimal. Adapun struktur data yang digunakan adalah sebagai berikut:

- detectedBots (set of integer)

Struktur data ini digunakan untuk menyimpan ID bot musuh yang telah terdeteksi oleh radar. Dengan menggunakan Set, setiap musuh yang terdeteksi hanya akan dicatat sekali, sehingga menghindari duplikasi data. Selain itu, operasi penambahan dan pengecekan keberadaan dalam Set memiliki kompleksitas $O(1)$, yang jauh lebih efisien dibandingkan array atau list yang memerlukan pencarian linear. Penggunaan Set juga memungkinkan bot untuk

dengan cepat mengetahui apakah musuh yang baru terdeteksi adalah musuh yang sama dengan yang sebelumnya telah dipindai.

Selain struktur data utama, terdapat beberapa variabel dengan tipe data primitif yang digunakan dalam implementasi strategi ini untuk menyimpan informasi penting terkait posisi musuh dan pergerakan bot. Adapun tipe data primitif yang digunakan adalah sebagai berikut:

- enemyX, enemyY (double)

Dua variabel ini digunakan untuk menyimpan total koordinat X dan Y dari seluruh musuh yang terdeteksi. Dengan menjumlahkan posisi semua musuh, kita dapat menghitung rata-rata koordinat mereka, yang nantinya digunakan untuk menentukan pusat keramaian di arena. Perhitungan ini dilakukan setiap kali bot memindai musuh baru, sehingga posisi pusat keramaian selalu diperbarui secara dinamis.

- targetX, targetY (double)

Variabel ini menyimpan koordinat titik kabur yang dihitung sebagai pencerminan pusat keramaian terhadap tengah peta. Dengan pendekatan ini, bot akan selalu bergerak menjauh dari area dengan kepadatan musuh tertinggi, meningkatkan peluang bertahan lebih lama. Perhitungan titik kabur hanya membutuhkan operasi aritmatika sederhana, sehingga tetap efisien dalam lingkungan real-time.

- savedEnemies (integer)

Variabel ini berfungsi sebagai penghitung jumlah musuh yang telah terdeteksi dalam satu siklus pemindaian radar. Nilai ini sangat penting dalam proses perhitungan rata-rata koordinat musuh, karena digunakan sebagai pembagi saat menentukan pusat keramaian. Jika tidak ada musuh yang terdeteksi, variabel ini akan bernilai nol, sehingga bot dapat menyesuaikan strategi pergerakan yang lebih fleksibel.

- shouldMove (boolean)

Variabel ini berfungsi sebagai penanda apakah bot perlu berpindah posisi atau tetap menyerang dari tempatnya saat ini. Jika musuh berada terlalu dekat atau bot terkena serangan, nilai shouldMove akan diatur menjadi true, sehingga bot segera melakukan pergerakan untuk menjauh dari bahaya. Sebaliknya, jika situasi masih aman, bot akan tetap berada di posisinya dan fokus menyerang musuh dari kejauhan.

2. Fungsi dan Prosedur

Dalam pengimplementasian Strategi Greedy Berdasarkan Pusat Keramaian (*Crowd Evasion*) terdapat beberapa fungsi dan prosedur digunakan untuk mengatur pergerakan bot, mendeteksi musuh, serta menyesuaikan strategi serangan dan

penghindaran secara dinamis. Adapun fungsi dan prosedur yang digunakan adalah sebagai berikut:

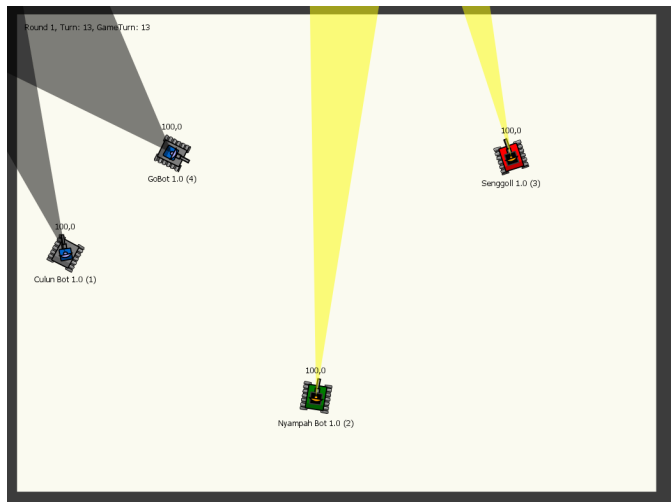
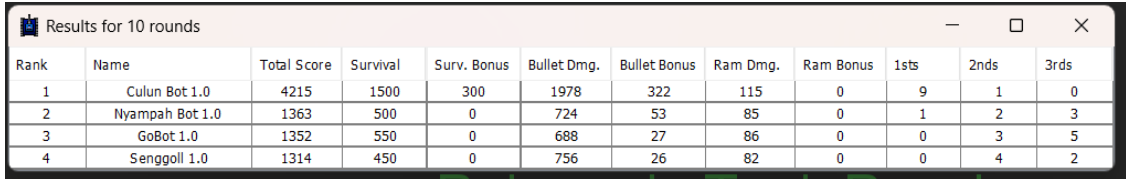
Nama Fungsi/Prosedur	Deskripsi	Kompleksitas Waktu
<code>OnScannedBot (e: ScannedBotEvent)</code>	Prosedur yang dipanggil setiap kali radar mendeteksi musuh. Menyimpan posisi musuh, memperbarui pusat keramaian, dan menentukan apakah bot perlu menghindar.	$O(1)$
<code>EscapeFromEnemies ()</code>	Prosedur yang menghitung titik kabur berdasarkan pencerminan pusat keramaian terhadap tengah peta, lalu menggerakkan bot ke arah tersebut untuk menjauh dari musuh.	$O(1)$
<code>AttackEnemies ()</code>	Prosedur yang mengatur pola pergerakan bot saat menyerang, menggunakan pola melingkar agar lebih sulit ditembak musuh.	$O(1)$
<code>SmartFire (distance: double)</code>	Fungsi yang menentukan kekuatan tembakan berdasarkan energi bot agar serangan tetap efisien dan tidak boros energi.	$O(1)$
<code>OnHitByBullet (e: HitByBulletEvent)</code>	Prosedur yang dijalankan saat bot terkena peluru, sehingga bot segera menjauh dari musuh dengan menjalankan <code>EscapeFromEnemies ()</code> .	$O(1)$
<code>OnBotDeath (e: BotDeathEvent)</code>	Prosedur yang dipanggil saat musuh mati, menghapus ID musuh dari <code>detectedBots</code> agar tidak lagi dihitung dalam perhitungan pusat keramaian.	$O(1)$
<code>ResetTracking ()</code>	Prosedur yang mengosongkan daftar musuh yang terdeteksi dan mereset nilai koordinat pusat keramaian untuk memulai pemindaian ulang.	$O(1)$
<code>TurnToFaceTarget (x: double, y: double)</code>	Prosedur yang memutar bot agar menghadap ke titik kabur atau target yang akan diserang.	$O(1)$
<code>OnHitWall (e:</code>	Prosedur yang dijalankan saat bot	$O(1)$

HitWallEvent)	menabrak dinding, membuat bot berputar dan mundur untuk menghindari posisi berbahaya.	
---------------	---	--

4.3 Analisis dan Pengujian

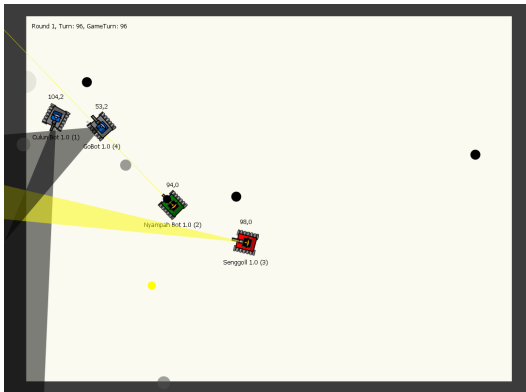
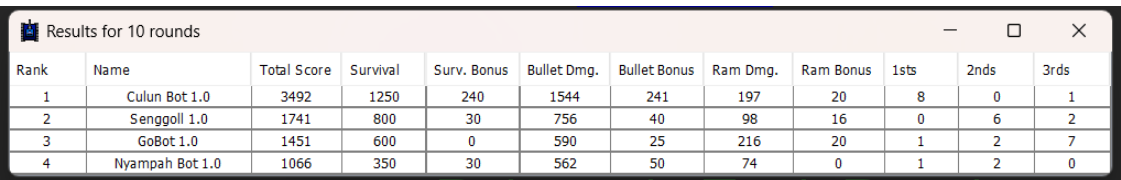
Pengujian dilakukan untuk menentukan keefektifan Bot dalam permainan Robocode Tank Royale dan menentukan algoritma yang paling mungkin menghasilkan skor maksimal. Pengujian menggunakan empat bot yang memiliki algoritma greedy yang berbeda yaitu *Greedy by Lowest Energy Enemy*, *Greedy by Nearest Enemy*, *Greedy by Crowd Evasion*, dan *Greedy by Single Target*. Pengujian dilakukan tiga kali dengan cara mempertandingkan keempat bot tersebut dalam 2 permainan yang terdiri dari 10 ronde dan 1 permainan yang terdiri dari 4 ronde.

Tabel 4.3.1 Hasil Pengujian 1

No.	Hasil dan Penjelasan
	 
1.	Culun Bot (<i>Crowd Evasion</i>) memiliki poin terbesar diantara ketiga bot, menandakan bahwa strategi <i>greedy</i> yang dimilikinya yang paling mangkus. Jika dilihat, survival poin yang dimiliki Culun Bot sangat jauh dibandingkan bot yang lain, yang berarti bahwa ia selalu menjadi bot terakhir yang berdiri di setiap rondonya.
2.	Nyampah Bot (<i>Lowest Energy Enemy</i>) menempati posisi kedua dalam perolehan

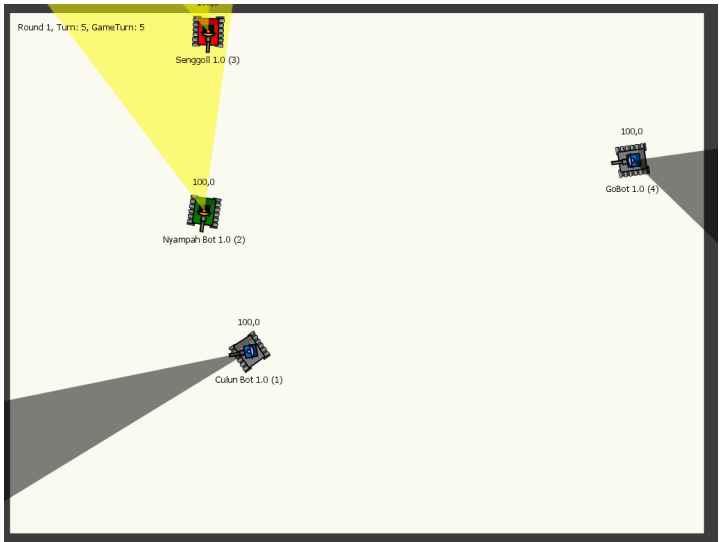
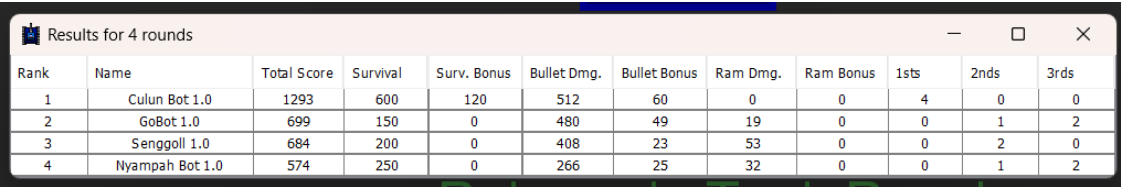
	poin di antara ketiga bot lainnya, menunjukkan bahwa menyerang lawan dengan energi terendah bisa menjadi strategi yang efektif dalam situasi tertentu, terutama saat musuh terdekat memiliki energi paling sedikit.
3.	GoBot (<i>Single Target</i>) berada di posisi ketiga. Hal ini menunjukkan bahwa strategi yang digunakan kurang efektif untuk melawan ketiga bot yang lain. Hal ini terlihat dari skor <i>Bullet Damage</i> yang paling rendah, menunjukkan bahwa banyak tembakan GoBot tidak mengenai target dengan akurat.
4.	Senggoll (<i>Nearest Enemy</i>) berada di posisi ke empat. Hal ini menyesuaikan dengan posisi awal bot-bot tersebut. Jika di awal permainan tidak ada robot yang dekat dengannya, maka akan dibutuhkan waktu untuk menyesuaikan dengan robot-robot lain.

Tabel 4.3.2 Hasil Pengujian 2

No.	Hasil dan Penjelasan
	 
1.	Culun Bot (<i>Crowd Evasion</i>) memperoleh skor tertinggi di antara ketiga bot, menunjukkan bahwa strategi greedy yang diterapkannya adalah yang paling efektif. Jika diperhatikan, poin bertahan hidup yang dimiliki Culun Bot jauh melampaui bot lainnya, menandakan bahwa ia selalu menjadi bot terakhir yang bertahan di setiap ronde.
2.	Senggoll (<i>Nearest Enemy</i>) berada di posisi kedua. Hal ini menyesuaikan dengan posisi awal bot-bot tersebut. Jika di awal permainan telah ada musuh yang dekat dengannya, maka robot dapat langsung menyerang musuh tanpa menunggu waktu.

3.	GoBot (<i>Single Target</i>) berada di posisi ketiga. Hal ini menunjukkan bahwa strategi yang digunakan kurang efektif untuk melawan ketiga bot yang lain. Hal ini terlihat dari skor Bullet Damage yang termasuk paling rendah, menunjukkan bahwa banyak tembakan GoBot tidak mengenai target dengan akurat.
4.	Nyampah Bot (<i>Lowest Energy Enemy</i>) berada di peringkat keempat dalam perolehan poin di antara tiga bot lainnya, menunjukkan bahwa menyerang lawan dengan energi terendah dapat menjadi kurang efektif di beberapa kondisi tertentu.

Tabel 4.3.3 Hasil Pengujian 3

No.	Hasil dan Penjelasan
	 
1.	Culun Bot (<i>Crowd Evasion</i>) mencatatkan skor tertinggi di antara ketiga bot, mengindikasikan bahwa strategi greedy yang digunakannya adalah yang paling berhasil. Jika diperhatikan, poin ketahanan yang dimiliki Culun Bot jauh lebih tinggi dibandingkan bot lainnya, menunjukkan bahwa ia konsisten menjadi bot terakhir yang bertahan di setiap ronde.
2.	GoBot (<i>Single Target</i>) berada di posisi kedua. Hal ini menunjukkan bahwa strategi yang digunakan efektif untuk melawan ketiga bot yang lain. Hal ini terlihat dari skor Bullet Damage yang termasuk paling tinggi, menunjukkan bahwa banyak tembakan GoBot mengenai target dengan akurat.

3.	Senggoll (<i>Nearest Enemy</i>) berada di posisi ketiga. Hal ini menyesuaikan dengan posisi awal bot-bot tersebut. Jika di awal permainan tidak ada musuh yang dekat dengannya, maka robot akan membutuhkan waktu lebih untuk mencari musuh terdekat dan kemudian menyerangnya.
4.	Nyampah Bot (<i>Lowest Energy Enemy</i>) berada di peringkat keempat dalam perolehan poin di antara tiga bot lainnya, menunjukkan bahwa menyerang lawan dengan energi terendah dapat menjadi kurang efektif di beberapa kondisi tertentu.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dalam Tugas Besar 1 IF2211 Strategi Algoritma, kami telah mengembangkan bot untuk Robocode Tank Royale dengan menerapkan berbagai strategi *greedy*. Pemilihan strategi dilakukan dengan membandingkan kinerja bot melalui tiga kali pertandingan dengan format (1 vs 1 vs 1 vs 1).

Dari hasil pengujian, strategi *Greedy by Crowd Evasion* (berdasarkan tingkat keramaian) terbukti paling efektif dibandingkan *Greedy by Lowest Energy Enemy* (berdasarkan energi musuh terendah), *Greedy by Nearest Enemy* (berdasarkan jarak musuh terdekat), dan *Greedy by Single Target Enemy* (berfokus pada satu target). Strategi ini unggul karena memungkinkan bot bertahan lebih lama dengan menghindari keramaian sambil tetap menyerang secara strategis. Sementara itu, strategi lain memiliki keunggulan masing-masing. *Greedy by Lowest Energy Enemy* efektif dalam menghabisi musuh lemah tetapi tidak mempertimbangkan posisi, *Greedy by Nearest Enemy* cepat dalam menemukan target tetapi kurang akurat terhadap musuh yang lincah, dan *Greedy by Single Target Enemy* kuat dalam duel tetapi rentan terhadap serangan dari musuh lain.

Secara keseluruhan, strategi *Greedy by Crowd Evasion* memberikan keseimbangan terbaik antara bertahan dan menyerang. Namun, efektivitasnya tetap dipengaruhi oleh kondisi arena, pola pergerakan musuh, serta faktor keberuntungan dalam *spawn* posisi bot dan musuh.

5.2 Saran

Dalam pelaksanaan tugas besar ini, terdapat beberapa saran yang dapat dipertimbangkan untuk pengembangan program di masa depan:

1. Tidak menunda-nunda dalam pengerjaan Tugas Besar agar dapat lebih banyak mengeksplorasi alternatif strategi *greedy* yang optimal.
2. Membaca dan memahami dokumentasi API secara lengkap sebelum mengerjakan tugas.
3. Melakukan sparing bot dengan kelompok lain untuk mengetahui kekurangan bot satu sama lain.

LAMPIRAN

Tautan Repository Github

Berikut tautan repository github kelompok kami untuk Tugas Besar 1 IF2211 Strategi Algoritma :

https://github.com/AlfianHanifFY/Tubes1_akuAdalahStima

Tautan Video

Berikut tautan video kelompok kami untuk Tugas Besar 1 IF2211 Strategi Algoritma :

https://youtu.be/glu1popGK_U?si=gKq12YkErLPFTIIF

Tabel Kelengkapan Spesifikasi

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)*.
[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) (Diakses pada 20 Maret 2025).
- Munir, Rinaldi. 2025. *Algoritma Greedy (Bagian 2)*.
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/05-Algoritma-Greedy-\(2025\)-Bag2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/05-Algoritma-Greedy-(2025)-Bag2.pdf) (Diakses pada 20 Maret 2025).
- Munir, Rinaldi. 2025. *Algoritma Greedy (Bagian 3)*.
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/06-Algoritma-Greedy-\(2025\)-Bag3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/06-Algoritma-Greedy-(2025)-Bag3.pdf) (Diakses pada 20 Maret 2025).