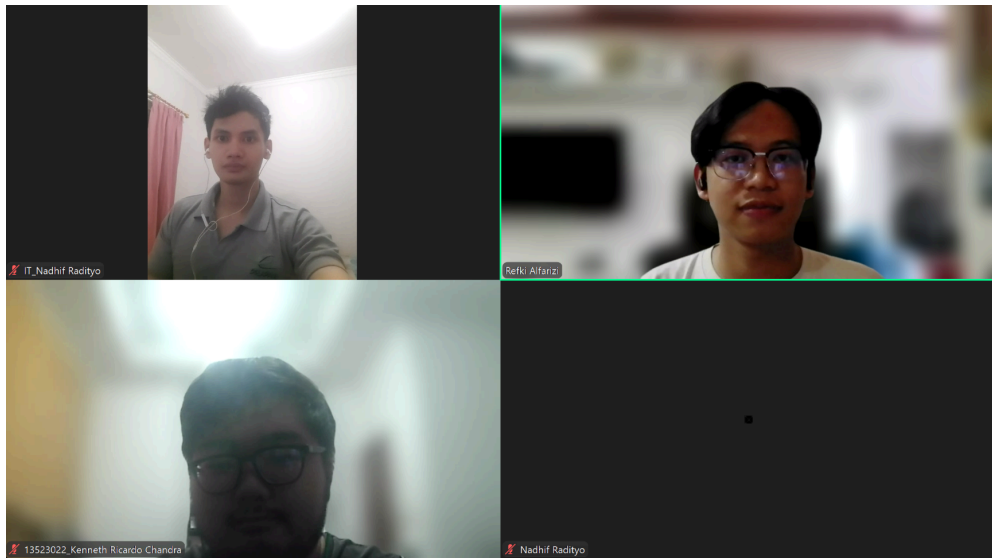


LAPORAN TUGAS BESAR 1

IF2211 Strategi Algoritma

Pemanfaatan Algoritma Greedy dalam Pembuatan Bot Permainan Robocode Tank Royale



Dipersiapkan oleh:

AdekTolongPapaDikejarRudalBalistik

1. Refki Alfarizi / 13523002
2. Kenneth Ricardo Chandra / 13523022
3. Nadhif Radityo Nugroho / 13523045

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

Daftar Isi

Daftar Isi.....	1
Daftar Gambar.....	2
Daftar Tabel.....	3
1 Deskripsi Tugas.....	3
2 Landasan Teori.....	10
2.1 Algoritma Greedy.....	10
2.2 Elemen-elemen Algoritma Greedy.....	10
2.3 Robocode Tank Royale.....	11
2.4 Cara Kerja Program.....	11
3 Aplikasi Strategi Greedy.....	14
3.1 Proses Mapping Persoalan Greedy.....	14
Tabel 3.1.1.1 Mapping Elemen Greedy untuk Menembak.....	14
Tabel 3.1.1.2 Mapping Elemen Greedy untuk Bergerak.....	14
3.2 Eksplorasi Alternatif Solusi.....	15
3.3 Pemilihan Solusi.....	22
4 Implementasi dan Pengujian.....	23
4.1 Implementasi.....	23
4.2 Pengujian.....	33
5 Kesimpulan dan Saran.....	35
6 Lampiran.....	36
6.1 Tautan.....	36
6.2 Tabel.....	36
7 Daftar Pustaka.....	37

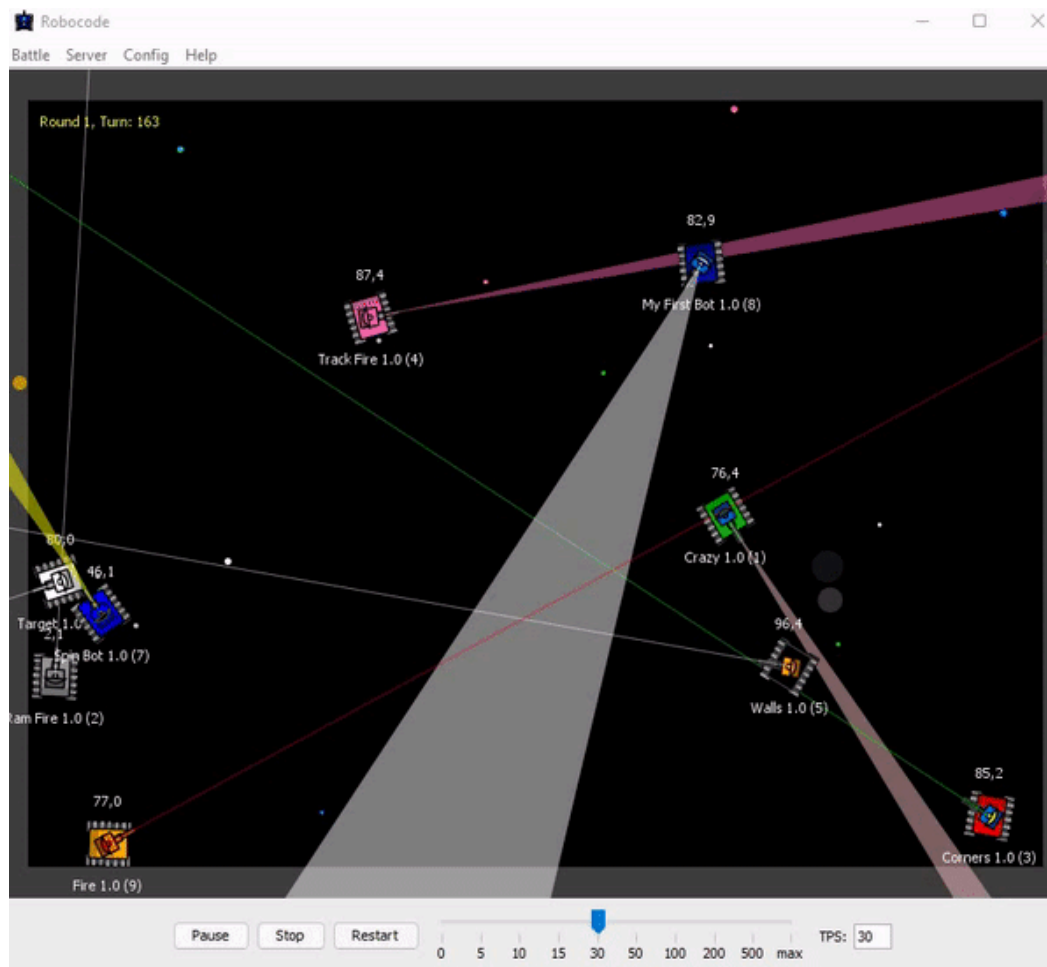
Daftar Gambar

Gambar 1.1 Robocode Tank Royale.....	4
Gambar 1.2 Tubuh Tank.....	7
Gambar 1.3 Sudut Pemindaian Radar.....	8
Gambar 1.4 Sudut Pemindaian Radar Nol.....	8

Daftar Tabel

Tabel 3.1.1.1 Mapping Elemen Greedy untuk Menembak.....	14
Tabel 3.1.1.2 Mapping Elemen Greedy untuk Bergerak.....	14

1 Deskripsi Tugas



Gambar 1.1 Robocode Tank Royale

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

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

Komponen-komponen dari permainan ini antara lain:

1. 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)

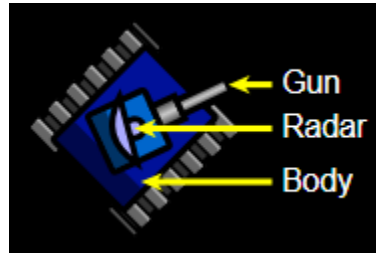
Saat menembakkan peluru, meriam akan menjadi panas. Peluru yang lebih berat menghasilkan lebih banyak panas dibandingkan peluru yang lebih ringan. Ketika meriam terlalu panas, bot tidak dapat menembak hingga suhu meriam turun ke nol. Selain itu, meriam juga sudah dalam keadaan panas di awal round dan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

7. Bagian Tubuh Tank

Tubuh tank terdiri dari 3 bagian:



Gambar 1.2 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.

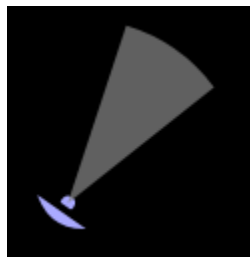
Bagian paling lambat adalah tubuh tank, yang dalam kondisi terbaik dapat berputar hingga 10 derajat per giliran. Namun, ini bergantung pada kecepatan bot saat ini. Semakin cepat bot bergerak, semakin lambat kemampuannya untuk berbelok.

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

10. Pemindaian

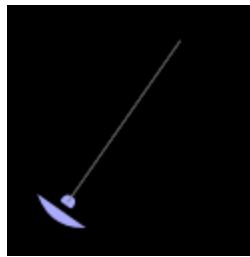
Aspek penting dalam Robocode adalah memindai bot musuh menggunakan radar. Radar dapat mendeteksi bot dalam jangkauan hingga 1200 piksel. Musuh yang berada lebih dari 1200 piksel dari bot tidak dapat terdeteksi atau dipindai oleh radar.

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 1.3 Sudut Pemindaian Radar

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



Gambar 1.4 Sudut Pemindaian Radar Nol

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 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](#).

2 Landasan Teori

2.1 Algoritma Greedy

Algoritma greedy adalah sebuah pendekatan dalam pemecahan masalah secara heuristik yang berfokus pada pengambilan keputusan terbaik secara lokal pada setiap langkah dengan harapan bahwa serangkaian pilihan yang optimal secara lokal tersebut akan menghasilkan solusi global yang optimal. Dalam implementasinya, algoritma ini bekerja dengan cara memilih opsi yang terlihat paling menguntungkan di setiap langkah tanpa mempertimbangkan hasil akhirnya. Sebagai contoh, strategi greedy untuk *travelling salesman problem* adalah mengunjungi setiap kota terdekat yang belum pernah dikunjungi pada setiap langkah perjalanan. Strategi heuristik ini tidak bermaksud untuk menemukan solusi terbaik, tetapi berakhir pada jumlah langkah yang setidaknya masuk akal karena untuk menemukan solusi optimal untuk masalah yang kompleks biasanya membutuhkan banyak langkah yang tidak masuk akal.

Keberhasilan pendekatan greedy sangat bergantung pada dua properti utama yang harus dimiliki oleh permasalahan yang dihadapi. Pertama, masalah tersebut harus memiliki sifat *optimal substructure*, yang berarti solusi optimal untuk permasalahan utama dapat dibangun dari solusi optimal sub-masalah yang lebih kecil. Kedua, harus terdapat *greedy property*, yang mengindikasikan bahwa pilihan terbaik secara lokal pada setiap langkah tidak perlu diperhitungkan kembali pada langkah selanjutnya.

Algoritma greedy memiliki beberapa keterbatasan. Salah satu kekurangannya adalah tidak adanya jaminan bahwa solusi yang dihasilkan merupakan solusi optimum global. Misalnya, pada permasalahan kompleks seperti *travelling salesman problem*, heuristik kota tetangga terdekat dapat menghasilkan solusi yang terburuk, tergantung pada penugasan jarak antar titik.

2.2 Elemen-elemen Algoritma Greedy

Berikut adalah elemen-elemen dalam algoritma greedy:

1. Himpunan Kandidat (C)
Himpunan yang berisi kandidat yang akan dipilih pada setiap langkah seperti simpul/sisi di dalam graf, job, task, koin, benda, karakter, dsb.
2. Himpunan Solusi (S)
Himpunan yang berisi kandidat yang sudah dipilih.
3. Fungsi Solusi
Fungsi untuk menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.
4. Fungsi Seleksi (*selection function*)
Fungsi untuk memilih kandidat berdasarkan strategi greedy tertentu. Strategi ini 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.

2.3 Robocode Tank Royale

Robocode Tank Royale adalah permainan pemrograman yang tujuannya adalah membuat program bot dalam bentuk tank virtual untuk bersaing dengan bot lain di arena pertempuran virtual. Permainan ini menyediakan *Application Programming Interface* (API) yang memungkinkan pengembang untuk mengontrol berbagai aspek bot, seperti pergerakan, pengendalian turret, dan operasi radar. API ini memberikan akses ke fungsi-fungsi penting, seperti penginderaan musuh, penentuan arah, dan eksekusi perintah serangan atau pertahanan, sehingga bot dapat beroperasi secara mandiri sesuai dengan logika yang telah diprogram.

Selain itu, permainan ini menyediakan *Graphical User Interface* yang memudahkan pengguna dalam mengelola pertempuran. Melalui GUI, pengguna dapat menghubungkan ke server lokal atau memulai server baru, mengatur pertempuran dengan bot-bot yang telah dipilih, menetapkan aturan pertempuran, melihat aksi bot secara *real-time* di arena, serta mengatur kecepatan visualisasi, sehingga memungkinkan pengguna untuk mengamati pertempuran dengan kecepatan yang sesuai untuk analisis dan evaluasi strategi.

2.4 Cara Kerja Program

Bot pada Robocode Tank Royale dapat berjalan melalui sistem komunikasi *client-server* berbasis WebSocket. Bot (client) terhubung ke server game (*local* maupun *remote*) yang mengatur simulasi pertempuran. Pada setiap putaran (turn), server mengirimkan data sensor dan informasi lainnya ke bot seperti informasi posisi, energi tersisa, data radar, dan berbagai event seperti tabrakan atau deteksi musuh. Bot kemudian memproses informasi ini sesuai dengan program telah dibuat dan mengirimkan perintah balik ke server seperti perintah gerakan, rotasi turret, atau penembakan. Server dari game mengatur waktu eksekusi setiap bot dengan timeout (biasanya 30-50ms per turn), untuk memastikan semua bot mendapat jatah waktu pemrosesan yang adil. Simulasi fisika dijalankan oleh server untuk menghitung pergerakan, tabrakan, dan efek penembakan, kemudian memperbarui keadaan arena dan mengirimkan data baru ke semua bot pada putaran berikutnya.

2.4.1 Mengimplementasikan Program

Cara mengimplementasikan bot secara lengkap, terutama dengan bahasa C#, dapat diakses pada [tautan resmi](#) berikut dan langkah untuk memulai ada pada [tautan starter pack](#). Sederhananya,

pengembang bisa mengimplementasikan bot dengan membuat sebuah *class* yang *derived* dari *class* API bot yang disediakan oleh Robocode Tank Royale.

Berikut adalah contoh struktur folder yang dapat digunakan:

```
MainBot
├─ MainBot.bat
├─ MainBot.cs
├─ MainBot.csproj
├─ MainBot.json
└─ MainBot.sh
```

Program utama terdapat pada file berekstensi .cs dan terdapat file untuk menyimpan *metadata* dari *class* pada file berekstensi .csproj dan *metadata* dari bot pada file berekstensi .json

File dengan ekstensi .bat dan .sh berisikan *command* untuk melakukan *build* dan bersifat opsional karena hanya untuk memudahkan proses.

Berikut adalah contoh program utama sederhana yang mengimplementasikan fungsi dasar dari API dan dapat digunakan:

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

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

    MyFirstBot() : base(BotInfo.FromFile("MyFirstBot.json")) { }

    public override void Run()
    {
        while (IsRunning)
        {
            SetForward(360);
            SetTurnGunRight(360);
            Go();
        }
    }

    public override void OnScannedBot(ScannedBotEvent evt)
    {
    }
}
```

```

        Fire(1);
    }

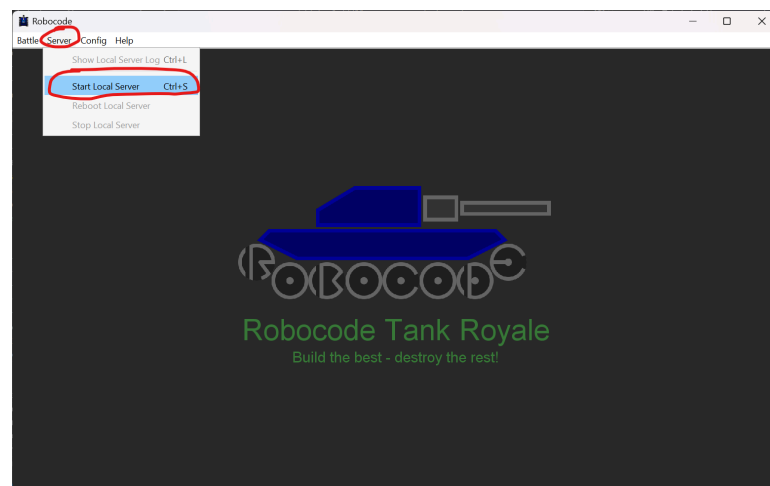
    public override void OnHitByBullet(HitByBulletEvent evt)
    {
        double bearing = CalcBearing(evt.Bullet.Direction);

        TurnLeft(90 - bearing);
    }
}

```

2.4.2 Menjalankan Program

Untuk menjalankan program, pastikan seluruh program dari [tautan resmi](#) dan [tautan starter pack](#) sudah terunduh. Program dapat dijalankan dengan membuka *game engine* dan menjalankan *local server* (keduanya melalui GUI yang disediakan).



Gambar 2.4.2.1 Menu Untuk Menyalakan Local Server

Setelah itu, terdapat *environment variables* yang harus diatur pada komputer yaitu `SERVER_SECRET`. *Server secret* bisa didapatkan pada file `server.properties` dari *starter pack*.

```

server.properties
1  bots-secrets=Ud+8bbdffUuAeSla6eRS0/qOr3EZHRHEQiKMnoag
2  controller-secrets=NcHV5GXlAmx1xG/xGCLkn3BzO/sgGCDew8dJRFsNQ/
3

```

Gambar 2.4.2.2 Server Secret

Setelah *server secret* diatur pada [environment variables](#), *battle* dapat dimulai pada GUI.

3 Aplikasi Strategi Greedy

3.1 Proses Mapping Persoalan Greedy

Pertempuran pada Robocode Tank Royale dapat dimenangkan dengan mengumpulkan skor sebanyak-banyaknya. Untuk mengakumulasi skor sebanyak-banyaknya dengan algoritma greedy, diperlukan rancangan yang jelas sehingga bot dapat mengambil langkah paling optimal di setiap langkahnya. Oleh karena itu, pendekatan greedy dapat dilakukan pada dua opsi yang akan dilakukan bot pada setiap langkah yaitu bergerak dan menembak.

3.1.1 Elemen Greedy untuk Menembak

Tabel 3.1.1.1 Mapping Elemen Greedy untuk Menembak

No.	Elemen	Penjelasan
1	Himpunan Kandidat	Seluruh aksi menembak untuk setiap arah pada setiap turn
2	Himpunan Solusi	Aksi yang terpilih
3	Fungsi Solusi	Memeriksa apakah aksi mampu memberikan skor tambahan
4	Fungsi Seleksi	Memilih aksi menembak yang mampu mengenai musuh
5	Fungsi Kelayakan	Memeriksa apakah aksi mungkin untuk mengenai musuh
6	Fungsi Obyektif	Mencari aksi menembak yang menghasilkan skor setinggi-tingginya

3.1.2 Elemen Greedy untuk Bergerak

Tabel 3.1.1.2 Mapping Elemen Greedy untuk Bergerak

No.	Elemen	Penjelasan
1	Himpunan Kandidat	Seluruh gerakan bot pada setiap turn
2	Himpunan Solusi	Gerakan yang terpilih
3	Fungsi Solusi	Memeriksa apakah gerakan mampu memberikan skor tambahan
4	Fungsi Seleksi	Memilih gerakan yang mampu meningkatkan <i>survivalibility</i> yaitu dengan menghindari tembakan musuh, meningkatkan kemungkinan tembakan mengenai musuh, atau

		meningkatkan kemungkinan untuk melakukan <i>ramming</i>
5	Fungsi Kelayakan	Memeriksa apakah gerakan mungkin untuk dilakukan
6	Fungsi Obyektif	Mencari gerakan yang menghasilkan skor setinggi-tingginya

3.2 Eksplorasi Alternatif Solusi

Berdasarkan diskusi dan riset yang dilakukan kelompok kami, terdapat empat alternatif solusi algoritma berbasis greedy untuk permasalahan ini. Beberapa alternatif solusi yang kami pilih adalah sebagai berikut:

3.2.1 CIV-1L Mauler

CIV-1L Mauler adalah bot dengan pendekatan greedy yang adaptif dengan jumlah musuh pada arena sehingga bot ini memiliki dua mode, yaitu mode *targetting* dan mode *roaming*. Pemisahan mode tersebut dikarenakan meningkatnya gangguan atau kejadian yang sulit atau bahkan mustahil untuk diperhitungkan seiring meningkatnya jumlah pemain.

Mode *targetting* aktif ketika jumlah lawan pada arena kurang dari 5. Pada mode ini, bot akan menarget salah satu lawan dan melakukan gerakan osilasi melingkari lawan. Strategi ini dilakukan untuk **mengurangi kemungkinan terkena tembakan dan meningkatkan kemungkinan tembakan mengenai lawan.**

Berikut adalah beberapa teknik yang digunakan pada bot ini dalam mode *targetting*:

1. Oscillating Movement

Bot ini mengimplementasikan gerakan osilasi di sekitar lawan dengan jarak yang selalu menyesuaikan pergerakan lawan. Hal yang menarik pada gerakan ini adalah adanya pendekatan heuristik dalam bergerak yaitu pada saat energi lawan/target berkurang, kemungkinan besar lawan melakukan tembakan sehingga bot ini menggunakan saat kejadian tersebut terjadi untuk memutar arah osilasi. Perputaran arah tersebut juga ditambahkan *randomness* agar pola pergerakan arah semakin tidak terprediksi sehingga lawan yang menggunakan prediksi pergerakan dalam targeting akan kesulitan untuk mengenai tembakannya.

2. Guess Factor

Saat menentukan arah penembakan, bot akan melakukan penambahan sudut dari posisi lawan saat ini secara *random*. Penambahan sudut tersebut dibatasi oleh perhitungan *max escape angle*, yaitu sudut maksimum bot lawan dari arah posisi saat terdeteksi untuk

bergerak dengan kecepatan maksimal dengan waktu yang dibutuhkan peluru untuk tiba di posisi lawan.

Mode *roaming* aktif ketika jumlah lawan pada arena lebih dari 4. Pada mode ini bot akan berputar-putar sepanjang mode ini aktif dan melakukan *targetting* pada salah satu lawan. Algoritma *targetting* pada mode ini sama dengan mode *targetting*, yaitu dengan menggunakan Guess Factor. Mode ini juga sangat adaptif terhadap gangguan/serangan yang dilakukan oleh musuh yang sangat mungkin terjadi saat bot di arena relatif banyak. Oleh karena itu, strategi ini **mengurangi kemungkinan terkena serangan (tembakan maupun *ramming*) dan meningkatkan kemungkinan tembakan mengenai lawan.**

Berikut adalah beberapa teknik yang digunakan pada bot ini dalam mode *roaming*:

1. Adaptive Move

Selain hanya berputar sepanjang mode ini aktif, bot juga akan menyesuaikan pergerakan sesuai dengan kejadian pada setiap turn. Berikut adalah penjelasan setiap kejadian.

- Ditabrak oleh lawan
Jika bot lawan memiliki energi yang rendah dan berada didepan hadapan, bot akan melakukan tembakan maksimum (*hit chance* nya sangat tinggi). Jika tidak, bot akan mundur dari lawan.
- Ditembak oleh lawan
Bot akan mengganti arah putar agar gerakan semakin sulit untuk diprediksi.

2. Move Away

Setiap beberapa *turn* (sekitar 240 *turn*), bot akan bergerak menuju sisi dari arena yang terdapat tepat dihadapannya dengan sedikit *offset* agar tidak menabrak tembok. Hal tersebut dilakukan sebagai salah satu cara heuristik karena saat terdapat banyak bot pada arena, bot memiliki kemungkinan tinggi untuk terperangkap pada satu daerah atau dekat tembok sehingga strategi ini akan membantu mengatasi hal tersebut.

Berikut adalah analisis dari alternatif solusi ini:

a. Elemen Greedy

Himpunan Kandidat:

Kandidat pergerakan adalah seluruh gerakan bot pada turn dan kandidat penembakan adalah seluruh aksi tembak tiap turn.

Himpunan Solusi:

Tembakan dan gerakan yang dipilih.

Fungsi Solusi:

Memeriksa apakah aksi tembakan dan gerakan mampu memberikan skor tambahan.

Fungsi Seleksi:

Memilih arah tembakan yang mendekati target/lawan.

Untuk mode *targetting*, pergerakan yang dipilih adalah gerakan yang mengosilasi lawan pada jarak tertentu. Untuk mode *roaming*, pergerakan yang dipilih adalah gerakan berputar dan mampu menghindari lawan yang menabrak.

Fungsi Kelayakan:

Memeriksa apakah pergerakan dan penembakan mungkin untuk dilakukan.

Fungsi Objektif:

Mencari gerakan yang mampu menghasilkan skor setinggi-tingginya dengan menghindari serangan lawan. Mencari arah tembakan yang mampu menghasilkan skor setinggi-tingginya dengan menentukan target.

b. Analisis Efisiensi Solusi

Efisiensi bot ini (pada kedua mode) sudah bagus dalam pergerakan karena setiap gerakan memiliki tujuan yang jelas tetapi dalam penembakan masih kurang efisien karena bot hanya melakukan kalkulasi posisi tanpa melakukan perhitungan kemungkinan tembakan akan mengenai musuh atau tidak sehingga target akan selalu ditembak.

c. Analisis Efektivitas Solusi

Bot ini mampu menghindari peluru lawan dengan baik karena gerakannya yang sulit diprediksi. Akan tetapi, penembakan lawan pada bot ini tidak efektif pada lawan yang bergerak lurus secara tegak lurus dari bot karena penggunaan Guess Factor yang menempatkan tembakan pada area potensi lawan berada sedangkan pada kondisi ini lawan akan berada pada ujung dari area potensi tersebut.

3.2.2 Hecate

Hecate adalah bot yang menggunakan pendekatan greedy sesuai dengan konsumsi energi yang digunakan serta persediaan energi yang tersisa pada bot. Apabila bot memiliki energi yang masih banyak atau diatas 70, maka bot berada dalam mode agresif atau menggunakan energi yang lebih banyak untuk menembak peluru sehingga damage yang dikeluarkan lebih besar. Saat bot memiliki energi sedang atau di antara 70 dan 30, Saat bot memiliki energi yang rendah atau dibawah 30, maka bot akan mengeluarkan peluru dengan energi yang lebih sedikit. Selain itu, saat bot memiliki energi dibawah 30, maka bot akan otomatis mencari lokasi yang aman dengan pergi ke suatu ujung atau corner dari arena. Apabila energi masih diatas 30, maka bot akan terus bertarung hingga energinya dibawah 30. Strategi ini **mengoptimalkan penggunaan energi agar dapat mendapatkan poin yang paling banyak.**

Berikut adalah analisis dari alternatif solusi ini:

a. Elemen Greedy

Himpunan Kandidat:

Himpunan kandidat yang dipilih adalah energi yang dimiliki oleh bot pada saat itu untuk menentukan aksi yang akan dipilih oleh bot.

Himpunan Solusi:

Himpunan solusi yang dipilih adalah aksi yang dilakukan oleh bot, berdasarkan energi yang dimilikinya. Bila energi diatas 70, maka bot akan menembakan peluru dengan energi besar. Bila energi di antara 70 dan 30, bot akan menembakan peluru dengan energi sedang dan jika energi dibawah 30, maka bot akan menembakan peluru dengan energi rendah. Bila energi diatas 30, maka bot akan terus menyerang musuh, namun bila dibawah 30, maka bot akan mulai bergerak menuju lokasi yang aman yaitu di ujung atau corner.

Fungsi Solusi:

Fungsi solusi pada strategi ini adalah aksi yang dilakukan untuk menjaga energi dari bot agar tidak habis dengan cepat dan dapat digunakan seefektif mungkin untuk mendapat poin yang paling besar.

Fungsi Seleksi:

Fungsi seleksi pada strategi ini adalah aksi yang dipilih untuk menjaga efisiensi energi tetap terjaga dengan baik dan memastikan bahwa aksi yang dilakukan adalah aksi terbaik yang dapat dilakukan saat berada dalam kondisi energi tertentu

Fungsi Kelayakan:

Fungsi kelayakan pada strategi ini adalah memilih aksi untuk dilakukan, apakah aksi tersebut dapat mengoptimalkan penggunaan energi dan menjaga efisiensi energi untuk mendapat poin yang paling banyak.

Fungsi Obyektif:

Fungsi obyektif pada strategi ini adalah memilih aksi dengan efisiensi energi yang paling tinggi untuk mendapatkan poin yang setinggi-tingginya.

b. Analisis Efisiensi Solusi

Efisiensi solusi sudah cukup baik karena dengan melakukan strategi ini, energi yang digunakan tidak cepat terkuras sehingga kemungkinan survivabilitynya lebih lama lebih tinggi, dibanding jika terus-terusan menghabiskan energi dengan besar. Namun secara penembakan dan pergerakan masih kurang efisien karena tidak berfokus pada dua hal tersebut.

c. Analisis Efektivitas Solusi

Efektivitas solusi ini cukup baik menghadapi bot yang menggunakan energi lebih banyak atau bot yang lebih agresif karena bot ini mengoptimalkan penggunaan energinya dengan cukup baik. Namun apabila menghadapi bot yang mementingkan kehidupannya, maka bot ini kurang efektif karena akan kehabisan energi dalam menghadapinya. Apabila bot ini bertemu dengan bot yang dapat melakukan tracking dan menembak dengan akurat, maka bot ini akan sangat terancam dan kemungkinan besar akan langsung masuk ke mode energi rendah yaitu menembak dengan energi rendah dan pergi ke area yang lebih aman

3.2.3 DantolBot

Bot ini dikembangkan menggunakan algoritma *greedy* untuk pengambilan keputusan secara optimal berdasarkan informasi historis dan analisis *crowd* dari pergerakan serta interaksi bot lain dalam arena. Bot mengumpulkan data dari berbagai *events* dalam permainan untuk membangun strategi serangan yang adaptif dan prediktif.

Berikut adalah analisis dari alternatif solusi ini:

a. Elemen Greedy

Himpunan Kandidat:

BotStateLoggingBotTask dan **IntentLoggingBotTask** mengumpulkan informasi keadaan bot dan intent tembakan. Informasi ini menyusun dasar dari himpunan kandidat, yaitu data historis pergerakan lawan dan pola serangan.

CrowdObserverBotTask membangun density map yang menyediakan informasi spasial juga temporal, sehingga kandidat juga dapat berupa wilayah dengan nilai kepadatan rendah (untuk pergerakan defensif) atau tinggi (untuk agresi) dalam turn number tertentu.

Himpunan Solusi:

Dari data logging, bot dapat menyaring kandidat (misalnya, posisi lawan atau jalur yang telah dianalisis) yang menunjukkan probabilitas serangan atau pertahanan terbaik. Penggabungan data fire intent dan crowd density menghasilkan solusi yang dapat dipilih untuk menyerang target atau menghindari.

Fungsi Solusi:

Keputusan tembakan atau pergerakan dapat dianggap sebagai fungsi solusi yang mengkonsolidasikan informasi dari event-event (seperti bullet hit, scanned bot, dll.) dan mengubahnya menjadi aksi (misalnya, memanggil fungsi Go() dengan parameter tertentu). Pada tingkat kode, pengiriman perintah melalui method Step() di setiap BotTask mengindikasikan realisasi solusi.

Fungsi Seleksi:

Pemilihan kandidat terbaik terjadi secara lokal pada setiap langkah. Misalnya, dalam **CrowdObserverBotTask**, fungsi `GetCrowdedValue()` menghitung nilai kepadatan pada sektor tertentu dan memilih daerah dengan probabilitas serangan atau pertahanan yang optimal. Pemilihan juga terjadi di **IntentLoggingBotTask**, di mana resolusi tembakan (hit wall, hit bot, hit bullet) menentukan apakah intent tembakan tersebut menguntungkan atau tidak.

Fungsi Kelayakan:

Validasi pada setiap event (misalnya, pengecekan status bot hidup/mati, validitas data posisi, atau kesesuaian fire intent terhadap data peluru) merupakan bentuk fungsi kelayakan. Metode seperti `EqualTo()` di `FireIntentState` mengecek apakah data peluru yang masuk sesuai dengan intent yang sudah dicatat.

Fungsi Objektif:

Maksimisasi peluang menghantam target dengan menggunakan data bullet hit dan resolusi intent untuk menilai efektivitas serangan. Minimisasi risiko dengan meminimalkan paparan terhadap daerah dengan kepadatan lawan yang tinggi, yang dihitung melalui density map. Juga perhitungan `GetTotalCrowdedValue()` dan `GetCrowdValue()` pada **CrowdObserverBotTask** secara tidak langsung membantu dalam mengoptimasi fungsi objektif dengan menilai distribusi kerumunan di arena.

b. Analisis Efisiensi Solusi

Implementasi menggunakan algoritma greedy yang bersifat lokal dan iterative, sehingga keputusan dapat diambil secara cepat tanpa perlu pencarian global yang berat. Penyimpanan state per round memberikan kesempatan untuk analisis temporal, sehingga bot dapat mengadaptasi strateginya dari round ke round berdasarkan pola pergerakan lawan. Meskipun algoritma greedy memiliki keunggulan dalam kecepatan pengambilan keputusan, solusi greedy cenderung hanya optimal secara lokal. Penambahan komponen pembelajaran atau evaluasi global (misalnya, dengan machine learning) bisa meningkatkan efektivitas dalam jangka panjang.

c. Analisis Efektivitas Solusi

Dengan menggabungkan data historis dari berbagai event (seperti bot hit, bullet hit, intent logging), bot dapat membuat keputusan yang lebih terinformasi dan responsif terhadap dinamika pertempuran. Fungsi seleksi dan objektif yang berbasis perhitungan probabilitas (melalui `densityMap` dan evaluasi fire intent) memungkinkan bot untuk memilih jalur serangan/pertahanan yang optimal.

Efektivitas dari algoritma greedy sangat bergantung pada akurasi data log. Jika terdapat noise atau data yang tidak konsisten, keputusan greedy yang diambil bisa kurang optimal. Evaluasi hasil dari resolusi fire intent (misalnya, membedakan antara hit wall, hit bot,

atau hit bullet) perlu dilakukan untuk mengkalibrasi fungsi objektif sehingga lebih tepat dalam menilai potensi risiko dan keuntungan.

Meskipun dengan analisis intensif ini, DantolBot tidak dapat membuktikan keefektifitasannya karena data historis yang terdapat sangat dependen terhadap radius radar yang digunakan. Meskipun data historis yang digunakan telah akurat, penggunaan informasi-informasi tersebut dalam strategi penyerangan masih dinilai minim. Sehingga, DantolBot masih dapat dikalahkan dengan mudah. Harapan kedepannya agar penggunaan data historis ini dapat digunakan untuk menebak pattern dari musuh atau dimasukkan ke metode-metode analisis lainnya yang dapat bermanfaat. Untuk saat ini, penulis implementasi masih belum kepikiran untuk strategi penyerangan yang terbaik.

3.2.4 Freedom

Freedom adalah bot yang mengimplementasikan pendekatan greedy dengan strategi adalah bergerak berputar di ujung arena sehingga objektif utama bot ini adalah **memaksimalkan peluang bertahan dari serangan peluru lawan**.

Dengan bergerak berputar, pergerakan akan semakin sulit diprediksi dan sulit ditembak.

Dengan bergerak di ujung arena, lawan hanya akan berada sekitar 90 derajat dari sudut pandang bot sehingga meminimalisir ekspos terhadap lawan.

Untuk penembakan, bot ini melakukan *target lock* agar tembakan mengarah kepada posisi lawan selagi masih terlihat atau hidup.

Berikut adalah analisis dari alternatif solusi ini:

a. Elemen Greedy

Himpunan Kandidat:

Kandidat pergerakan adalah seluruh gerakan bot pada turn dan kandidat penembakan adalah seluruh aksi tembak tiap turn.

Himpunan Solusi:

Tembakan dan gerakan yang dipilih.

Fungsi Solusi:

Memeriksa apakah aksi tembakan dan gerakan mampu memberikan skor tambahan.

Fungsi Seleksi:

Memilih arah tembakan yang sesuai dengan target/lawan saat dideteksi.

Memilih gerakan dengan resiko terendah (pada kasus ini ujung arena terdekat dari *spawn*)

Fungsi Kelayakan:

Memeriksa apakah pergerakan dan penembakan mungkin untuk dilakukan.

Fungsi Objektif:

Mencari gerakan yang mampu menghasilkan skor setinggi-tingginya dengan menghindari serangan lawan. Mencari arah tembakan yang mampu menghasilkan skor setinggi-tingginya dengan menentukan target.

b. Analisis Efisiensi Solusi

Efisiensi bot ini sudah bagus dalam pergerakan karena setiap gerakan memiliki tujuan yang jelas tetapi dalam penembakan masih kurang efisien karena bot hanya melakukan kalkulasi posisi tanpa melakukan perhitungan kemungkinan tembakan akan mengenai musuh atau tidak sehingga target akan selalu ditembak.

c. Analisis Efektivitas Solusi

Pergerakan pada bot ini tidak mampu beradaptasi dengan kondisi pada arena dengan baik sehingga rentan terhadap gangguan oleh lawan.

3.3 Pemilihan Solusi

Berdasarkan solusi-solusi alternatif tersebut, solusi alternatif yang dipilih adalah CIV-1L Mauler. Pemilihan tersebut didasari oleh pendekatan greedy pada bot tersebut yang **mengurangi kemungkinan terkena tembakan dan meningkatkan kemungkinan tembakan mengenai lawan**. Penembakan lawan merupakan salah satu cara “paling mudah” dalam mengakumulasi skor dan bot ini memiliki kemampuan yang baik untuk menembak musuh berdasarkan hasil pengujian yang telah kami lakukan.

4 Implementasi dan Pengujian

4.1 Implementasi

4.1.1 Implementasi CIV-1L Mauler Bot

```
type State:
  combatState: CombatState

type Enemy:
  Id: integer
  X, Y: real
  Energy, Bearing, Direction, Speed: real

type CombatState:
  { Melee: Ketika masih ada banyak musuh }
  { By1: Ketika tersisa satu musuh }
  Melee, By1

{ Bot Attributes }
oldTurn: integer <- 0
turn: real <- 2
turnDir: integer <- 1
moveDir: integer <- 1
oldEnergy: real <- 100
cornerRadius: real <- 50
state: State
target: Enemy
random: Random

procedure Run()
{ALGORITMA}
  Set bot color properties
  Adjust gun and radar movement behavior

  while (IsRunning) do
    UpdateState()
    target <- null
    SetTurnRadarLeft(1000) { Sweep radar untuk mencari musuh }

    if (state.combatState == Melee) then
      MaxSpeed <- 5
      MaxTurnRate <- MAX_TURN_RATE
      if (TurnNumber % 400 == 0) then
        distance <- DistanceToWall() - 20
        MaxSpeed <- 8
        Forward(distance)
      else
        SetTurnLeft(INFINITY * turnDir)
        SetForward(INFINITY)
    Go()

procedure UpdateState()
{ALGORITMA}
  if (EnemyCount > 3) then
    state.combatState <- Melee
  else
```



```

state.combatState <- By1

procedure OnScannedBot(e: ScannedBotEvent)
{KAMUS}
    absBearing, bulletPower, randomGuessFactor, maxEscapeAngle, randomAngle,
    firingAngle: real
    enemy: Enemy

{ALGORITMA}
    enemy <- Create Enemy(e.ScannedBotId, e.X, e.Y, e.Energy, BearingTo(e.X, e.Y),
    e.Direction, e.Speed)

    if (target == null or target.Id == e.ScannedBotId or DistanceTo(e.X, e.Y) <
    DistanceTo(target.X, target.Y)) then
        target <- enemy

    UpdateState()
    absBearing <- BearingTo(e.X, e.Y) + Direction

    if (state.combatState == By1) then
        { Menyesuaikan radar dan gerakan bot saat bertarung 1v1 }
        SetTurnRadarLeft(NormalizeRelativeAngle(absBearing - RadarDirection) * 2)
        AdjustOscillatingMovement(e)

        bulletPower <- CalculateBulletFirePower(e.Energy, Energy)
        firingAngle <- CalculateFiringAngle(absBearing, bulletPower, e.Speed)
        SetTurnGunLeft(NormalizeRelativeAngle(firingAngle))

        if (GunHeat <= 0.1) then
            SetFire(bulletPower)

    else if (state.combatState == Melee) then
        { Menyesuaikan strategi untuk pertarungan banyak musuh }
        absBearing <- BearingTo(target.X, target.Y) + Direction
        SetTurnRadarLeft(NormalizeRelativeAngle(absBearing - RadarDirection) * 2)

        bulletPower <- CalculateBulletFirePower_Melee(target)
        firingAngle <- CalculateFiringAngle(absBearing, bulletPower, target.Speed)
        SetTurnGunLeft(NormalizeRelativeAngle(firingAngle))

        if (TurnNumber % 60 == 0) then
            target <- null
            SetTurnRadarLeft(360)
        else if (GunHeat <= 0.1) then
            SetFire(bulletPower)

    oldTurn <- TurnNumber
    ClearEvents()

procedure OnHitBot(e: HitBotEvent)
{KAMUS}
    bearing: real

{ALGORITMA}
    bearing <- BearingTo(e.X, e.Y)

    if (bearing > -10 and bearing < 10) then
        Fire(3)
    else
        target <- Create Enemy(e.VictimId, e.X, e.Y, e.Energy, BearingTo(e.X, e.Y),

```

```

NormalizeAbsoluteAngle(Direction + BearingTo(e.X, e.Y) + 180), 0)
    Back(100)

    if (e.IsRammed) then
        turnDir <- -turnDir
        TurnLeft(10 * turnDir)

procedure OnHitByBullet(e: HitByBulletEvent)
{ALGORITMA}
    turnDir <- -turnDir
    TurnLeft(10 * turnDir)

```

Menggunakan strategi **greedy** berdasarkan **kondisi pertempuran** (Melee atau 1v1). Strategi ini mencakup **posisi, pergerakan, dan penargetan musuh** untuk bertahan dan menyerang secara optimal.

A. Mode Melee (Banyak Musuh, ≥ 4)

Strategi Utama dengan menghindari pertempuran langsung dengan tetap bergerak di pinggiran arena dan memantau situasi. Jika bot mencapai dinding, ia akan maju sejauh mungkin agar tetap di pinggir. Jika tidak, bot akan berputar tak terbatas dengan gerakan maju otomatis.

Bot tidak fokus pada musuh tertentu, tetapi tetap memindai area sekitar dengan radar terus berputar. Menembak musuh dengan daya tembak yang disesuaikan dengan jarak musuh.

B. Mode 1v1 (Hanya Sedikit Musuh Tersisa)

Menggunakan gerakan oscillating (berayun) untuk menghindari tembakan musuh dan menyerang dengan presisi. Jika tidak bergerak, bot akan mengganti arah dengan bergerak maju/mundur sejauh 185 unit. Bot juga menyesuaikan arah berdasarkan posisi musuh, bergerak zig-zag/osilasi untuk menghindari tembakan. Bot akan selalu memilih musuh terdekat atau musuh yang sudah menjadi target sebelumnya. Jika musuh kehilangan energi secara tiba-tiba (indikasi mereka menembak), bot akan mengubah arah gerakannya untuk menghindari peluru. Menghitung sudut pelarian maksimum musuh (escape angle) dan menggunakan random guess factor untuk memperkirakan tembakan. Radar selalu diarahkan ke musuh untuk mempertahankan target. Senjata akan berputar ke arah musuh dengan memperkirakan gerakannya sebelum menembak.

4.1.2 Implementasi HecateBot

```

constant MAX_SPEED_HIGH = 10
constant MAX_SPEED_MEDIUM = 6
constant MAX_SPEED_LOW = 4
constant MIN_ENERGY_SAFE = 30
constant MAX_ENERGY_AGGRESSIVE = 70
constant BULLET_POWER_LOW = 1.0

```

```

constant BULLET_POWER_HIGH = 3.0

type Enemy:
  <Id : integer,
    X, Y, Energy, Bearing, Direction, Speed : double>

{VARIABLES}
turnDir : integer <- 1 // Direction for turning
moveDir : integer <- 1 // Direction for moving
oldEnergy : double <- 100
target : Enemy <- null
random : RandomGenerator

procedure Run()
{ALGORITMA}
  while IsRunning do
    SetTurnRadarLeft(360) // Continuously scan the environment

    if Energy < MIN_ENERGY_SAFE then
      MoveToCorner()
      MaxSpeed <- MAX_SPEED_LOW
    else if Energy > MAX_ENERGY_AGGRESSIVE then
      MaxSpeed <- MAX_SPEED_HIGH
      if target ≠ null then
        MoveTowardEnemy(target)
      else
        Patrol()
    else
      MaxSpeed <- MAX_SPEED_MEDIUM
      Patrol()

    Go() // Execute actions

procedure OnScannedBot(e : ScannedBotEvent)
{KAMUS}
  enemy : Enemy
  absBearing, bulletPower, enemySpeed, futureX, futureY, futureBearing : double

{ALGORITMA}
  enemy <- CreateEnemyFromEvent(e)

  if target = null OR enemy.Energy < target.Energy OR DistanceTo(enemy.X, enemy.Y) <
  DistanceTo(target.X, target.Y) then
    target <- enemy

  absBearing <- BearingTo(e.X, e.Y) + Direction
  bulletPower <- CalculateBulletFirePower(Energy)

  enemySpeed <- e.Speed
  (futureX, futureY) <- PredictEnemyFuturePosition(e.X, e.Y, e.Direction, enemySpeed,
5)
  futureBearing <- BearingTo(futureX, futureY) + Direction

  SetTurnGunLeft(NormalizeRelativeAngle(futureBearing - GunDirection))

  if GunHeat = 0 AND Energy > bulletPower * 2 then
    SetFire(bulletPower)

  if EnemyHasFired(e.Energy, oldEnergy) then
    moveDir <- moveDir * -1

```

```

        SetForward(100 * moveDir)

        oldEnergy <- e.Energy

    procedure MoveToCorner()
    {KAMUS}
        distanceToWall : double

    {ALGORITMA}
        distanceToWall <- DistanceToWall()
        if distanceToWall > 50 then
            SetForward(distanceToWall - 20)
            SetTurnLeft(45 * turnDir)
        else
            SetTurnLeft(90 * turnDir)

    procedure Patrol()
    {ALGORITMA}
        if DistanceRemaining < 10 then
            moveDir <- moveDir * -1
            SetForward(150 * moveDir)
            SetTurnLeft(30 * turnDir)

    procedure MoveTowardEnemy(enemy : Enemy)
    {KAMUS}
        bearing : double

    {ALGORITMA}
        bearing <- BearingTo(enemy.X, enemy.Y)
        SetTurnLeft(bearing)
        SetForward(DistanceTo(enemy.X, enemy.Y) * 0.7)

    procedure OnHitByBullet(e : HitByBulletEvent)
    {ALGORITMA}
        if Energy < 40 then
            Back(100)
        else
            turnDir <- turnDir * -1
            TurnLeft(30 * turnDir)

    procedure OnHitBot(e : HitBotEvent)
    {ALGORITMA}
        if Energy > e.Energy + 10 then
            Fire(3)
            Forward(50)
        else
            Back(100)

```

Bot terus memutar radar untuk mendeteksi musuh dan menentukan target terbaik. Target dipilih berdasarkan energi lebih rendah atau jarak lebih dekat (greedy approach). Jika energi tinggi (>70), bot bergerak agresif mendekati musuh. Jika energi rendah (<30), bot bergerak ke sudut arena untuk menghindari konfrontasi langsung. Jika energi sedang, bot bergerak secara acak (patrolling). Bot memprediksi posisi masa depan musuh berdasarkan kecepatan dan arah gerakan lawan. Kekuatan tembakan disesuaikan berdasarkan energi bot. Jika musuh baru saja menembak (deteksi perubahan energi), bot akan segera berubah arah untuk menghindari tembakan.

4.1.3 Implementasi DantolBot

```
type BotStateSnapshot:
  id : integer
  snapshotTurn : integer
  energy : EnergyBotStateDiff (nullable)
  direction : DirectionBotStateDiff (nullable)
  speed : SpeedBotStateDiff (nullable)
  position : PositionBotStateDiff (nullable)
  dead : DeadBotStateDiff (nullable)

type BotState:
  id : integer
  roundNumber : integer
  getTurnNumber : function() -> integer
  diffs : list of BotStateDiff
  Diffs : readonly list of BotStateDiff

  procedure UpdateEnergy(energy: double)
  procedure UpdateDirection(direction: double)
  procedure UpdateSpeed(speed: double)
  procedure UpdatePosition(positionX: double, positionY: double)
  procedure SetDead(dead: boolean)

  function Snapshot(): BotStateSnapshot
  function Snapshot(snapshotTurn: integer): BotStateSnapshot

type FireIntent:
  Firepower : double
  TargetId : integer (nullable)
  TargetX : double (nullable)
  TargetY : double (nullable)

type CrowdObserverBotTask:
  densityMap : 2D array of double
  debugBitmap : Bitmap
  const densityResolution = 1.5
  const sigma = 50
  const lambda = 1.2

  procedure DrawDensityRegion(positionX: double, positionY: double, intensity:
double)
  procedure DrawTemporalDecay()
  function GetTotalCrowdedValue(): double
  function GetCrowdValue(positionX: double, positionY: double, directionStart:
double, directionEnd: double): double

type CrowdSweepAttackStrategyBotTask:
  moveDirSign : integer

  procedure FindMostCrowdedDirection(OUTPUT crowdedDirection: double, OUTPUT
directionRange: double)
  function NormalizeBearing(angle: double): double

procedure Step():
{ALGORITHM}
  FindMostCrowdedDirection(crowdedDirection, directionRange)
  SetTurnRight(NormalizeBearing(crowdedDirection - Direction))
```

```

    if DistanceRemaining == 0 then
        moveDirSign <- moveDirSign * -1
        SetForward(250 * moveDirSign)

    SetTurnRadarRight(360)
    await Go()
    return IsRunning

procedure OnScannedBot(scannedBotEvent):
{ALGORITHM}
    enemyBearing <- scannedBotEvent.Bearing
    enemyDistance <- scannedBotEvent.Distance
    fireDirection <- NormalizeBearing(Direction + enemyBearing)

    SetTurnGunRight(NormalizeBearing(fireDirection - GunDirection))

    firePower <- CalculateBulletFirePower(enemyDistance)
    await Fire(firePower)

procedure OnHitBot(botHitBotEvent):
{ALGORITHM}
    moveDirSign <- moveDirSign * -1
    SetBack(150 * moveDirSign)
    await Go()

procedure OnHitByBullet(hitByBulletEvent):
{ALGORITHM}
    moveDirSign <- moveDirSign * -1
    SetForward(250 * moveDirSign)
    await Go()

procedure FindMostCrowdedDirection(OUTPUT crowdedDirection: double, OUTPUT
directionRange: double)
{ALGORITHM}
    baseDirection <- Direction
    directionSplit <- 180
    while directionSplit > 22.5 do
        crowdLeft <- crowdObserver.GetCrowdValue(X, Y, baseDirection, baseDirection +
directionSplit)
        crowdRight <- crowdObserver.GetCrowdValue(X, Y, baseDirection, baseDirection -
directionSplit)
        if crowdLeft >= crowdRight then
            baseDirection <- baseDirection + directionSplit / 2
        else
            baseDirection <- baseDirection - directionSplit / 2
        directionSplit <- directionSplit / 2
    crowdedDirection <- baseDirection
    directionRange <- directionSplit

procedure OnTick(tickEvent):
{ALGORITHM}
    currentTurn <- TurnNumber
    snapshots <- empty list

    while botStateUpdateConsumer has data do
        snapshot <- botStateUpdateConsumer.dequeue().Snapshot()
        append snapshot to snapshots

    DrawTemporalDecay()

```

```

for each snapshot in snapshots do
  if snapshot.Dead == false then
    DrawDensityRegion(snapshot.PositionX, snapshot.PositionY, Exp(-(currentTurn
- snapshot.PositionTurn) / 30))
  else
    DrawDensityRegion(snapshot.PositionX, snapshot.PositionY,
-Exp(-(currentTurn - snapshot.DeadTurn) / 30))

procedure DrawDensityRegion(positionX: double, positionY: double, intensity: double)
{ALGORITHM}
  radius <- Constants.BoundingCircleRadius
  positionX <- positionX * densityResolution
  positionY <- positionY * densityResolution
  for each (x, y) in densityMap do
    distance <- EuclideanDistance(x, y, positionX, positionY)
    if distance <= radius then
      density <- 1
    else
      density <- Exp(-((distance - radius)^2) / (2 * sigma^2))
    if intensity >= 0 then
      densityMap[x, y] <- max(densityMap[x, y], density * intensity)
    else
      densityMap[x, y] <- min(densityMap[x, y], (1 - density) * -intensity)

```

Bot dalam Robocode ini menggunakan pendekatan greedy dengan crowd-aware movement, serta fire intent tracking. BotStateLoggingBotTask Mencatat perubahan kondisi bot lain, termasuk energi, posisi, arah, status hidup/mati. Data ini digunakan untuk melacak pergerakan musuh seiring waktu. IntentLoggingBotTask Melacak fire intent bot ini sendiri, termasuk target yang ingin ditembak, kekuatan peluru, dan hasil dari tembakan (kena bot lain, tembok, atau peluru lain). CrowdObserverBotTask Menghitung crowd density di arena berdasarkan posisi bot lain. Menggunakan kernel-based density estimation dengan fungsi gaussian smoothing untuk memperkirakan area yang padat.

Bot menggunakan strategi CrowdSweepAttackStrategyBotTask. FindMostCrowdedDirection mencari arah paling padat, dengan menggunakan binary search untuk mempersempit sudut pengamatan hingga menemukan titik dengan kepadatan tertinggi. Jika bot masih diam, ia bergerak maju sejauh 250 unit dalam satu arah. Setiap kali bot bertabrakan atau ditembak, ia membalik arah gerakannya (zig-zag avoidance pattern). Radar berputar 360 derajat setiap langkah untuk mendapatkan informasi terbaru tentang lawan.

4.1.4 Implementasi FreedomBot

```

{ KONSTANTA }
  TankHeight <- 36
  TankWidth <- 36
  TankRadius <- 18
  CornerOffset <- 100
  radius <- 20

```

```

{ ENUMERASI }
    type Side = (Left, Right, Top, Bottom)
    type Corner = (TopLeft, TopRight, BottomLeft, BottomRight)

{ VARIABEL GLOBAL }
    cornerX, cornerY : float // Posisi pusat rotasi di pojok arena
    direction : float <- 1 // Arah rotasi (1 = searah jarum jam, -1 = berlawanan)
    CurrentSide : Side
    CurrentCorner : Corner

{ ALGORITMA }
procedure Main()
    new FreedomBot() -> Start()

procedure FreedomBot()
    INHERIT Bot(BotInfo.FromFile("FreedomBot.json"))

procedure Run()
    // Inisialisasi warna robot
    SetRobotColors()

    // Menentukan posisi awal robot di sudut arena
    InitiatePosition()

    // Konfigurasi orientasi radar dan turret
    AdjustGunForBodyTurn <- true
    AdjustRadarForGunTurn <- true

    while (IsRunning) do
        DoMove()
        SetTurnRadarLeft(1000) // Radar terus berputar untuk mendeteksi musuh
        Go()

procedure OnScannedBot(enemy : ScannedBotEvent)
    e <- Enemy(enemy)
    DoScan(e)
    DoFire(e)

procedure DoScan(enemy : Enemy)
    if (enemy == null) then
        SetTurnRadarLeft(360) // Putar radar 360 derajat jika tidak ada musuh
    else
        absBearing <- BearingTo(enemy.X, enemy.Y) + Direction
        turn <- NormalizeRelativeAngle(absBearing - RadarDirection)
        SetTurnRadarLeft(turn * 2) // Fokus radar ke posisi musuh

procedure DoFire(enemy : Enemy)
    if (enemy != null) then
        absBearing <- BearingTo(enemy.X, enemy.Y) + Direction
        bulletPower <- CalculateBulletFirePower(DistanceTo(enemy.X, enemy.Y))

        predictedX, predictedY <- PredictEnemyPosition(enemy, bulletPower)
        theta <- NormalizeAbsoluteAngle(atan2(predictedY - Y, predictedX - X))

        SetTurnGunLeft(NormalizeRelativeAngle(theta - GunDirection + absBearing))
        SetFire(bulletPower)

procedure DoMove()
    if (DistanceRemaining == 0 AND TurnRemaining == 0) then

```



```

        SetForward( $\infty$ ) // Bergerak maju terus

        angularVelocity <- Speed / radius * (180 /  $\pi$ )
        TurnRate <- angularVelocity * direction // Rotasi mengikuti sudut melingkar

    procedure OnHitBot(e : HitBotEvent)
        bearing <- BearingTo(e.X, e.Y)
        if (-10 < bearing < 10) then
            Fire(3) // Jika musuh berada di depan langsung tembak
        else
            Back(100) // Jika tidak, mundur

        if (e.IsRammed) then
            TurnLeft(10) // Jika ditabrak, belok kiri sedikit

    procedure InitiatePosition()
        MaxSpeed <- 8
        CurrentCorner <- DetermineClosestCorner()
        CurrentSide <- GetSideFromCorner(CurrentCorner)

        MoveToCorner(CurrentCorner, CornerOffset + radius + 2)

        angle <- GetInitialGunAngle(CurrentCorner)
        SetTurnGunLeft(NormalizeRelativeAngle(angle - GunDirection + 180))
        TurnLeft(NormalizeRelativeAngle(angle - 90))

        (cornerX, cornerY) <- GetCornerCoordinates(CurrentCorner)

        MaxSpeed <- 5 // Kurangi kecepatan setelah sampai di pojok

    function DetermineClosestCorner() -> Corner
        topLeft <- DistanceTo(0, ArenaHeight)
        topRight <- DistanceTo(ArenaWidth, ArenaHeight)
        bottomLeft <- DistanceTo(0, 0)
        bottomRight <- DistanceTo(ArenaWidth, 0)

        minDistance <- Min(topLeft, topRight, bottomLeft, bottomRight)

        if (minDistance == topLeft) then -> Corner.TopLeft
        if (minDistance == topRight) then -> Corner.TopRight
        if (minDistance == bottomLeft) then -> Corner.BottomLeft
        if (minDistance == bottomRight) then -> Corner.BottomRight

    procedure MoveToCorner(corner : Corner, offset : float)
        (cornerX, cornerY) <- GetCornerPosition(corner, offset)

        TurnLeft(NormalizeRelativeAngle(BearingTo(cornerX, cornerY)))
        Forward(DistanceTo(cornerX, cornerY))

        CurrentCorner <- corner

    procedure MoveToSide(side : Side)
        closestDistance <- GetDistanceToSide(side)
        angle <- GetSideBearing(side)

        isBackward <- |angle| > 90
        angleRadians <- ConvertToRadians(angle)

        if (isBackward) then
            angleRadians <-  $\pi$  - angleRadians

```

```

distance <- closestDistance / cos(angleRadians)

if (isBackward) then
  Back(distance - 1)
else
  Forward(distance - 1)

direction <- GetGunDirectionForSide(CurrentCorner, side)
SetTurnGunLeft(NormalizeRelativeAngle(direction - GunDirection))

Go()
CurrentSide <- side

function HitChance(distance : float, anglediff : float) -> float
  anglediffRad <- ConvertToRadians(anglediff)
  if (distance > 200) then
    chance <- Max(cos(anglediffRad), 0) * Max(1 - (distance / 900)^(1.5), 0.1)
  else
    chance <- Max(1 - (distance / 900)^(1.5), 0.1)
  -> chance


```

Bot **FreedomBot** menggunakan strategi **greedy berbasis corner** dalam permainan. Saat permainan dimulai, bot mencari sudut terdekat dari arena. Bot bergerak langsung ke sudut tersebut dan menyesuaikan arah gerakan serta senjatanya agar optimal untuk menembak lawan. Setelah mencapai sudut, bot bergerak dalam lintasan melingkar dengan radius tetap. Gerakan melingkar ini memungkinkan bot menghindari tembakan lawan secara pasif dan mempertahankan posisi yang sulit dijangkau.

Saat mendeteksi lawan, bot menggunakan strategi **Circular Targeting** untuk memperkirakan posisi musuh berdasarkan arah dan kecepatan gerakannya. Bot menyesuaikan sudut laras senjatanya untuk meningkatkan akurasi tembakan. Bot menghitung kekuatan peluru berdasarkan jarak musuh agar serangan lebih efektif. Radar bot terus berputar untuk mencari lawan. Jika lawan terdeteksi, radar mengunci area sekitar lawan agar tetap dalam pengawasan.

4.2 Pengujian

4.2.1 Pengujian 1

 Results for 10 rounds — □ ×

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Freedom Bot 1.0	1444	650	60	655	67	11	0	2	3	1
2	CIV-1L Mauler 1.0	1392	500	90	694	88	19	0	3	1	0
3	Dantol Bot 1.0	585	350	0	212	18	5	0	1	1	2
4	Hecate 1.0	552	250	0	278	4	19	0	0	1	3

4.2.2 Pengujian 2

Results for 10 rounds											
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	CIV-1L Mauler 1.0	1899	750	120	917	106	6	0	5	1	0
2	Freedom Bot 1.0	1196	550	30	547	58	10	0	1	4	1
3	Hecate 1.0	705	200	30	432	34	8	0	1	0	3
4	Dantol Bot 1.0	661	450	0	196	5	10	0	0	2	3

4.2.3 Pengujian 3

Results for 10 rounds											
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	CIV-1L Mauler 1.0	1479	600	90	677	99	13	0	3	1	2
2	Freedom Bot 1.0	1164	600	0	538	19	6	0	2	4	0
3	Dantol Bot 1.0	811	500	60	212	27	11	0	1	1	3
4	Hecate 1.0	171	50	0	105	0	16	0	0	0	1

4.2.4 Analisis Hasil Pengujian

Dari hasil pengujian tiga kali, hasil yang didapatkan berupa kemenangan dari CIV-1L Mauler dan juga Freedom bot. CIV-1L Mauler bot berhasil meraih poin yang tinggi dari bullet damage yang berarti bot tersebut berhasil menggunakan strategi greedynya yaitu untuk menembak saat sedang bergerak untuk menghindari. Strategi greedy dari Freedom bot juga lumayan berhasil karena bot tersebut bergerak melingkar di bagian ujung sehingga mudah untuk menghindari tembakan dan melancarkan tembakan kembali. Strategy dari Dantol dan Hecate kurang baik karena Dantol memerlukan beberapa ronde terlebih dahulu sebelum benar-benar dapat melakukan strateginya dengan baik dan Hecate kalah karena melawan bot yang dapat menembak dan menghindari dengan akurat, sehingga penggunaan energinya akan cepat habis.

5 Kesimpulan dan Saran

5.1.1 Kesimpulan

Kesimpulan yang kami dapatkan dari melakukan beberapa pendekatan strategi dengan algoritma greedy adalah strategi terbaik untuk memenangkan Robocode adalah dengan menggunakan strategi yang dilakukan oleh bot CIV-1L Mauler karena dari pengetesan yang telah dilakukan, bot tersebut yang paling sering mendapatkan poin tertinggi dan cenderung memenangkan pertandingan. Strategi yang diterapkan pada bot CIVIL-Mauler adalah strategi yang ditentukan berdasarkan jumlah musuh, yang mana saat masih terdapat banyak musuh adalah untuk tidak terlalu agresif agar tidak menjadi mangsa, namun saat musuh sedikit akan bergerak dengan banyak dan menembak dengan berputar agar tidak mudah diserang oleh musuh. Solusi ini mungkin bukan solusi yang paling optimal untuk memenangkan Robocode, namun solusi ini dapat menjadi salah satu kandidatnya untuk solusi paling optimal.

5.1.2 Saran

Saran yang dapat kami berikan adalah untuk terus melakukan perbaikan dan pengembangan pada algoritma dan strategi yang telah dibuat, dengan berfokus pada perbaikan targeting dan penembakan pada musuh agar mengurangi tembakan yang kurang akurat. Selain itu, penggunaan energi juga harus diperhatikan agar bot tidak mudah kalah. Pergerakan juga harus diperhatikan agar bot dapat terus bergerak namun dapat menembak dengan akurat. Uji melawan bot-bot lain juga harus dilakukan terus-menerus agar dapat dilihat hasil dari perubahan yang dilakukan. Selain itu, saran lain yang dapat kami berikan adalah untuk tidak menjadi deadliner karena deadliner akan berbahaya untuk tugas-tugas kedepannya.

6 Lampiran

6.1 Tautan

- GitHub
https://github.com/I0stplains/Tubes1_AdekTolongPapaDikejarRudalBalistik
- Video
<https://youtu.be/9CfVcgZbBDI>

6.2 Tabel

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.	✓	

7 Daftar Pustaka

- Munir, Rinaldi. (2025). “Algoritma Greedy (Bagian 1)”. Diakses pada 22 Maret 2025, dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf)
- Munir, Rinaldi. (2025). “Algoritma Greedy (Bagian 2)”. Diakses pada 22 Maret 2025, dari [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)”. Diakses pada 22 Maret 2025, dari [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 Dev. (2025). “Robocode Tank Royale Docs”. Diakses pada 22 Maret 2025, dari <https://robocode-dev.github.io/tank-royale/>