

IF2211 Strategi Algoritma
Laporan Tugas Besar 1



Disusun oleh:

Kelompok TankTop:

Rafael Marchel Darma Wijaya (13523146)

Muhammad Kinan Arkansyaddad (13523152)

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132
2025**

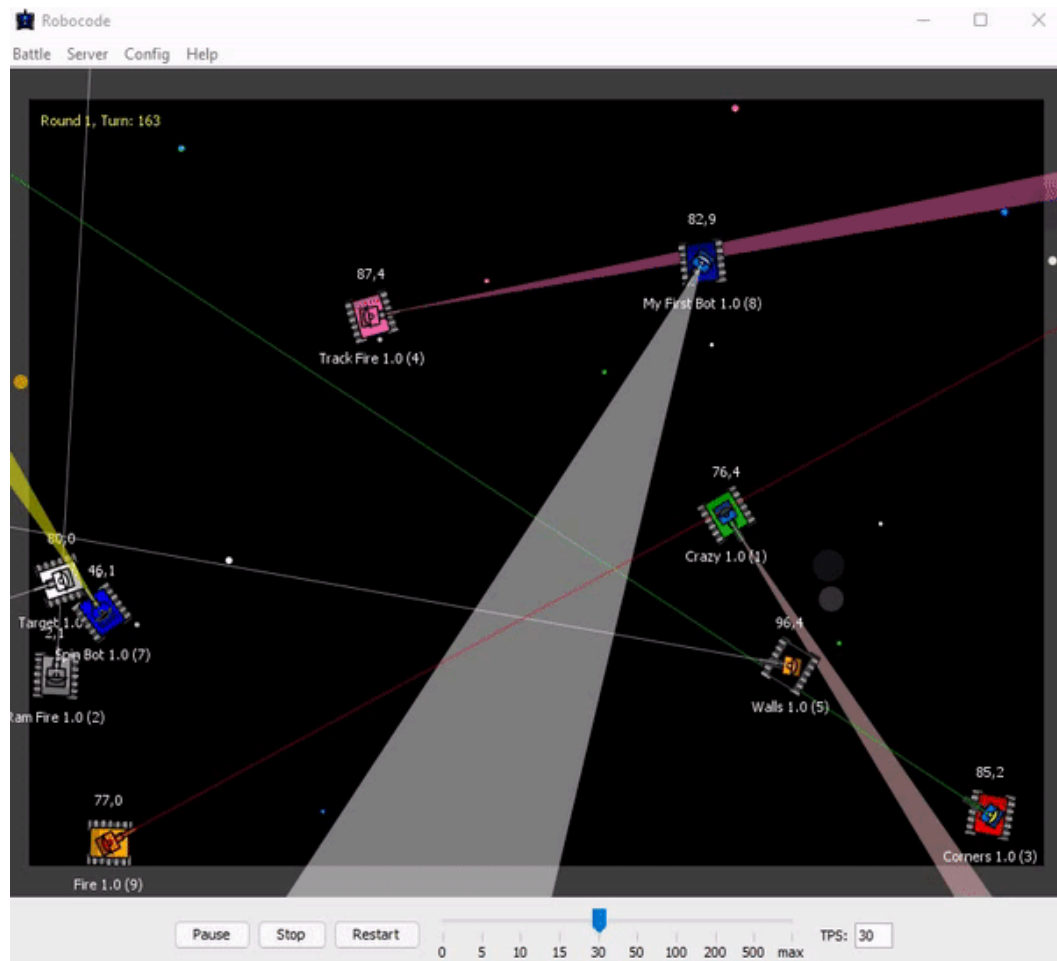
DAFTAR ISI

DAFTAR ISI.....	2
BAB I DESKRIPSI TUGAS.....	3
BAB II LANDASAN TEORI.....	12
2.1. Algoritma Greedy.....	12
2.2. Cara Kerja Program.....	13
BAB III APLIKASI STRATEGI GREEDY.....	18
3.1. Mapping Persoalan Robocode.....	18
3.2. Eksplorasi Algoritma Greedy.....	20
3.2.1. Algoritma Greedy Bot Obi-Wan Tanknobi.....	20
3.2.2. Algoritma Greedy Bot ASS.....	23
3.2.3. Algoritma Greedy Bot ASU.....	28
3.2.4. Algoritma Greedy Bot PNIS.....	30
3.3. Analisis Efisiensi dan Efektivitas Algoritma Greedy.....	33
3.3.1. Analisis Bot Obi-Wan Tanknobi.....	33
3.3.2. Analisis Bot ASS.....	35
3.3.3. Analisis Bot ASU.....	38
3.3.4. Analisis Bot PNIS.....	38
3.4 Algoritma Greedy Terpilih.....	41
BAB IV IMPLEMENTASI DAN PENGUJIAN.....	42
4.1. Implementasi Algoritma Greedy.....	42

4.1.1. Algoritma Greedy Bot Obi-Wan Tanknobi.....	42
4.1.2. Algoritma Greedy Bot ASS.....	45
4.1.3. Algoritma Greedy Bot ASU.....	53
4.1.4. Algoritma Greedy Bot PNIS.....	60
4.2. Hasil Pengujian.....	64
4.3. Analisis Hasil Pengujian.....	66
BAB V KESIMPULAN DAN SARAN.....	67
5.1. Kesimpulan.....	67
5.2. Saran.....	67
LAMPIRAN.....	69
DAFTAR PUSTAKA.....	70

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 melewatkan turn tersebut. Jika bot melewatkan turn, ia tidak akan bisa menyesuaikan gerakannya atau menembakkan senjatanya karena

server tidak menerima perintah tepat waktu sebelum turn berikutnya dimulai.

3. Energi

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

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

4. Peluru

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

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

ringan tidak memberikan banyak poin energi saat mengenai bot musuh.

5. Panas Meriam (Gun Heat)

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

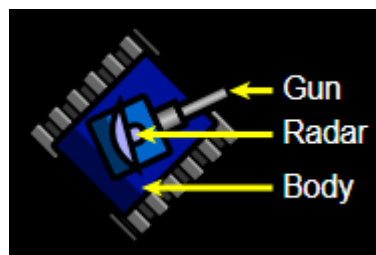
6. Tabrakan

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

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

7. Bagian Tubuh Tank

Tubuh tank terdiri dari 3 bagian:



Gambar 2. Body 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

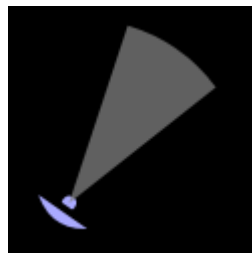
bergantung pada kecepatan bot saat ini. Semakin cepat bot bergerak, semakin lambat kemampuannya untuk berbelok.

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

10. Pemindaian

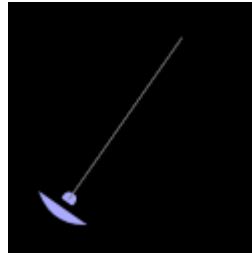
Aspek penting dalam Robocode adalah memindai bot musuh menggunakan radar. Radar dapat mendeteksi bot dalam jangkauan hingga 1200 piksel. Musuh yang berada lebih dari 1200 piksel dari bot tidak dapat terdeteksi atau dipindai oleh radar.

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 3. Radar Tank

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 4. Arah Radar Tank

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 lainya 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.

BAB II

LANDASAN TEORI

2.1. Algoritma Greedy

Algoritma greedy adalah algoritma yang memecahkan persoalan secara langkah demi langkah (*step by step*). Setiap langkah algoritma greedy mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan apa yang akan terjadi ke depannya, prinsip "*take what you can get now!*". Algoritma greedy memilih optimum lokal pada setiap langkahnya dengan harapan berakhir dengan optimum global.

Algoritma greedy memiliki elemen-elemen yang membantu dalam menentukan solusi terbaik. Elemen-elemen algoritma greedy terdiri dari:

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

Dengan menggunakan elemen-elemen di atas, maka dapat dikatakan bahwa algoritma greedy melibatkan pencarian sebuah

himpunan bagian, S , dari himpunan kandidat, C , yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu S menyatakan suatu solusi dan S dioptimasi oleh fungsi objektif.

Algoritma greedy akan menghasilkan beberapa solusi optimum lokal dan dari semua optimum lokal tersebut akan dipilih solusi terbaik yang menjadi optimum global. Optimum global belum tentu solusi optimum (terbaik), bisa jadi hanya merupakan solusi *sub-optimum* atau *pseudo-optimum*. Hal ini terjadi karena algoritma greedy tidak beroperasi secara menyeluruh terhadap semua kemungkinan solusi yang ada, sebagaimana pada metode *exhaustive search*, dan terdapat beberapa fungsi seleksi yang berbeda, sehingga kita harus memilih fungsi yang tepat jika ingin menghasilkan solusi optimum. Jadi, pada sebagian persoalan, algoritma greedy tidak selalu berhasil memberikan solusi yang optimal, namun sub-optimal.

2.2. Cara Kerja Program

1. Menjalankan program dan bot

Download aplikasi robocode tank royale. File yang di-download seharusnya memiliki nama seperti `robocode-tankroyale-gui-x.y.z.jar`. Untuk menjalankan program, lakukan perintah,

```
java -jar robocode-tankroyale-gui-x.y.z.jar
```

pada terminal.

Selanjutnya, pilih menu `Config` lalu `Bot Root Directories`. Kemudian tambahkan direktori tempat menyimpan folder-folder bot. Ketika memilih menu `Battle` lalu `Start Battle`, semua bot yang berada pada direktori *root* akan muncul pada kotak kiri-atas.

Pilih bot yang akan dipertandingkan dan klik tombol *boot*. Bot akan muncul di kotak kanan-atas dan kemudian muncul di kotak kiri-bawah. Setelah melakukan *boot* pada semua bot yang diinginkan, klik *add all* untuk menambahkan semua bot atau *add* untuk hanya menambahkan bot yang dipilih. Lalu klik tombol *Start Battle*.

2. Aksi Bot

Setiap ronde permainan dibagi menjadi *turns* yang merupakan unit waktu terkecil. Pada setiap *turn*, bot dapat melakukan beberapa hal yakni, menggerakkan bot, *scan* bot musuh, menembak, atau pun bereaksi terhadap *event* (menabrak dinding, terkena tembakan, etc). Perintah untuk bergerak, berbelok, *scan*, menembak, dan aksi lainnya dikirim ke *server* sebagai *intent* pada *turn* tersebut. Kemudian bot akan mendapatkan informasi dari *server* berdasarkan *intent* yang telah dikirim. Dalam satu *turn* terdapat batas waktu untuk bot mengirimkan *intent* ke *server*. Waktu ini biasanya selama 30-50 ms. Jika dalam satu *turn* bot tidak mengirimkan *intent* ke *server*, maka bot dianggap melewatkan *turn* tersebut dan tidak dapat melakukan aksi yang diharapkan karena *server* tidak menerima aksi tersebut.

3. Implementasi Algoritma *Greedy* ke dalam Bot

Pertama buatlah *folder* yang dinamai dengan nama bot. Buatlah *file json* dengan *fields* sebagai berikut:

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

Fields "name", "version", dan "authors" wajib terisi, sedangkan *fields* lainnya bersifat opsional. Buat *file source code C#* untuk bot. Susun *source code* bot dengan kerangka sebagai berikut,

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

public class BotTemplate : Bot
{
    // The main method starts our bot
    static void Main(string[] args)
    {
        new BotTemplate().Start();
    }

    // Constructor, which loads the bot
    config file
    BotTemplate() :
    base(BotInfo.FromFile("BotTemplate.json"))
    { }

    // Called when a new round is started
    -> initialize and do some movement
    public override void Run()
```



```

{
    BodyColor = Color.FromArgb(0x00,
0x00, 0x00);
    TurretColor = Color.FromArgb(0x00,
0x00, 0x00);
    RadarColor = Color.FromArgb(0x00,
0x00, 0x00);
    BulletColor = Color.FromArgb(0x00,
0x00, 0x00);
    ScanColor = Color.FromArgb(0x00,
0x00, 0x00);
    TracksColor = Color.FromArgb(0x00,
0x00, 0x00);
    GunColor = Color.FromArgb(0x00,
0x00, 0x00);

    // Repeat while the bot is running
    while (IsRunning)
    {
        // Write your bot logic
    }
}
}

```

Ganti nama BotTemplate sesuai dengan nama bot yang dibuat. Fungsi Run adalah fungsi yang dipanggil setiap awal ronde. Ada beberapa fungsi lain yang dapat di-*override* untuk bereaksi terhadap suatu *event*. Contohnya adalah OnHitByBullet adalah suatu fungsi yang dapat di-*override* untuk menjalankan suatu algoritma ketika terkena tembakan. Selain *event-event* yang sudah tersedia, dapat juga dibuat *event* baru dengan membuat *class* baru yang merupakan *child class* dari *class* Condition. Lalu, *event* ini akan dideteksi oleh fungsi OnCustomEvent.

Buatlah *file* .csproj yang lalu dinamai oleh nama bot. *File* .csproj harus memiliki isi sebagai berikut

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

Perhatikan syarat *fields file* .csproj berikut:

1. **RootNamespace** berisi nama *Class* dari bot
2. **TargetFramework** berisi versi .Net version yang diperlukan bot untuk dijalankan, yaitu versi .Net dari bot.
3. **LangVersion** berisi versi bahasa C# untuk menjalankan bot.
4. **PackageReference** berisi versi Robocode.TankRoyale.BotApi yang digunakan. versi yang digunakan adalah "0.30.0"

Terakhir, buat *file* .cmd dan .sh yang bernama sesuai dengan nama bot. *File* ini berisi perintah cmd atau bash untuk melakukan *build* terhadap *file* cs yang telah dibuat.

BAB III

APLIKASI STRATEGI GREEDY

3.1. Mapping Persoalan Robocode

Mapping persoalan Robocode Tank Royale ke elemen-elemen algoritma greedy diperlukan untuk mengidentifikasi bagaimana bot dapat mengambil keputusan optimal berdasarkan kondisi saat ini. Dengan memahami setiap elemen greedy, kita dapat merancang strategi yang efisien. Berikut adalah pemetaan elemen-elemen greedy dalam Robocode Tank Royale:

1. Himpunan kandidat

Himpunan kandidat dalam persoalan Robocode adalah semua tindakan/aksi yang dapat dilakukan oleh bot-nya dalam satu tick permainan. Aksi-aksi ini termasuk bergerak maju atau mundur, memutar turret, menembak, atau bahkan tetap diam. Setiap aksi ini menjadi opsi yang dapat dipilih oleh algoritma greedy berdasarkan kondisi saat itu dalam permainan.

2. Himpunan solusi

Himpunan solusi terdiri dari rangkaian aksi yang telah dipilih oleh bot selama permainan untuk mencapai tujuan tertentu, seperti bertahan lebih lama atau mengalahkan lawan. Dalam pendekatan greedy, bot akan terus melakukan aksi dan menambahkan aksi tersebut ke dalam solusi berdasarkan evaluasi lokal terbaik tanpa mempertimbangkan efek jangka panjang.

3. Fungsi solusi

Fungsi solusi menentukan kapan strategi yang dijalankan telah mencapai tujuan tertentu. Dalam Robocode, ini berarti memastikan bahwa bot tetap hidup lebih lama dari lawan, mencapai jumlah *kill* tertentu, atau berhasil mempertahankan posisi strategis.

4. Fungsi seleksi(*selection function*)

Fungsi seleksi bertanggung jawab untuk memilih aksi terbaik dari himpunan kandidat berdasarkan strategi greedy dan *heuristic* yang dipilih. Pemilihan ini dilakukan berdasarkan metrik tertentu, seperti jumlah nyawa musuh, peluang mengenai target, atau risiko terkena serangan. Fungsi seleksi tiap strategi akan dijelaskan lebih mendalam di bagian 3.2 Eksplorasi Algoritma Greedy.

5. Fungsi kelayakan(*feasible*)

Fungsi kelayakan memastikan bahwa aksi yang dipilih tetap valid dan tidak menyebabkan situasi yang merugikan. Misalnya, jika bot mempertimbangkan untuk menembak, fungsi kelayakan akan memeriksa apakah bot memiliki cukup energi dan apakah sudut tembakan memungkinkan peluru mengenai target. Dalam navigasi, fungsi kelayakan juga mengecek apakah bot dapat bergerak ke arah yang diinginkan tanpa menabrak dinding atau masuk ke posisi berbahaya.

6. Fungsi objektif

Fungsi objektif dalam algoritma greedy bertanggung jawab untuk memaksimumkan atau meminimumkan suatu nilai agar bot dapat mencapai tujuannya secara optimal. Dalam konteks Robocode Tank Royale, fungsi ini digunakan

untuk mengarahkan strategi bot dengan memilih aksi yang memberikan keuntungan terbesar dalam jangka pendek.

3.2. Eksplorasi Algoritma Greedy

3.2.1. Algoritma Greedy Bot Obi-Wan Tanknobi

Ide dari strategi algoritma greedy ini adalah algoritma greedy heuristik berbasis posisi dan reaksi serta memprioritaskan *survivability* (daya bertahan hidup) dari bot dengan cara memposisikan bot di “high ground” atau lokasi yang lebih strategis dalam arena, memberikan keuntungan posisi relatif terhadap musuh. Dengan memilih posisi yang menguntungkan, bot dapat menghindari tembakan musuh lebih efektif dan lebih mudah menargetkan lawan. Selain itu, algoritma ini mengutamakan reaksi terhadap perubahan situasi, yaitu mendeteksi perubahan energi pada musuh yang diprediksi akan mengarah pada tembakan, dan memilih untuk melakukan gerakan menghindar dengan cepat. Strategi ini mengandalkan keputusan berbasis situasi saat itu juga, di mana bot bergerak secara reaktif untuk menghindari kerusakan sambil tetap menjaga posisi yang memungkinkan serangan efektif. Bot ini juga menyerang secara langsung saat musuh terdeteksi, tanpa memprediksi pergerakan musuh selanjutnya.

a. Himpunan kandidat

Himpunan kandidat algoritma bot Obi-Wan Tanknobi terdiri dari berbagai tindakan yang dapat dilakukan oleh bot. Ini mencakup pergerakan ke posisi yang lebih ideal di arena, menghindari serangan lawan, melakukan serangan terhadap lawan, serta melakukan pemindaian untuk mendeteksi posisi musuh.

Kandidat-kandidat ini selalu tersedia bagi bot untuk dipilih dalam menentukan langkah berikutnya.

b. Himpunan solusi

Himpunan solusi adalah kumpulan tindakan yang telah diambil oleh bot selama pertandingan. Ini termasuk posisi yang sudah dicapai (misalnya, berhasil mencapai bagian atas arena sebagai "high ground"), gerakan menghindar yang telah dilakukan, dan tembakan yang telah dilakukan ke arah musuh. Setiap tindakan yang diambil menambah elemen ke dalam himpunan solusi, dengan tujuan mempertahankan keuntungan strategis.

c. Fungsi solusi

Fungsi solusi menentukan apakah himpunan solusi yang dibangun sudah mencapai kondisi yang optimal atau cukup baik untuk memenuhi tujuan bot. Dalam hal ini, bot dianggap telah mencapai solusi sementara jika berhasil menjaga posisinya di "high ground" / posisi yang ideal di arena, tetap hidup lebih lama dari musuh, dan memberikan damage yang cukup melalui tembakan. Bot tidak memiliki strategi jangka panjang, sehingga fungsi solusi lebih bersifat lokal dalam konteks keputusan-keputusan kecil yang diambil setiap saat.

d. Fungsi seleksi

Fungsi seleksi dalam bot ini didasarkan pada strategi greedy yang menggunakan heuristik berbasis posisi dan reaksi. Dalam *positioning*, bot menggunakan

heuristik “high ground advantage”, di mana bot memilih untuk langsung menuju posisi ideal karena menganggap posisi tersebut memberikan keuntungan taktis, seperti kontrol lebih baik terhadap *vulnerable area* dari bot ini. Selain itu, dalam menghadapi serangan musuh, bot memilih untuk segera menghindari begitu mendeteksi penurunan energi lawan, yang menunjukkan adanya tembakan, tanpa mempertimbangkan pola tembakan jangka panjang. Dalam hal menyerang, bot langsung menembak begitu mendeteksi musuh tanpa melakukan prediksi pergerakan lawan, mengutamakan aksi cepat dibandingkan akurasi jangka panjang.

e. Fungsi kelayakan

Fungsi kelayakan memastikan bahwa setiap tindakan yang dipilih layak dilakukan tanpa melanggar batasan yang ada. Sebagai contoh, jika bot ingin bergerak ke posisi atas, bot memastikan bahwa gerakan tersebut tidak menyebabkan tabrakan dengan dinding arena atau bot lain. Demikian pula, saat melakukan penghindaran, bot memilih untuk maju atau mundur sejauh 300 unit sesuai dengan posisi bot saat ini.

f. Fungsi objektif

Fungsi objektif dalam algoritma greedy ini adalah memaksimalkan keuntungan posisi dan kelangsungan hidup bot dalam pertempuran. Dengan selalu memilih posisi lebih ideal menempel di tembok, bot meningkatkan kemungkinan bertahan lebih lama dan

memiliki sudut serangan yang lebih baik. Selain itu, fungsi objektif juga dapat dikaitkan dengan memaksimalkan jumlah tembakan yang mengenai lawan serta meminimalkan kemungkinan terkena tembakan dengan cepat menghindar setelah mendeteksi serangan musuh.

3.2.2. Algoritma Greedy Bot ASS

Bot ASS (*Aggressive Seeker Sharpshooter*) adalah bot yang mengaplikasikan algoritma *greedy* berorientasi serangan dan penentuan target. Bot ini mula-mula akan melakukan *radar scan* 360 derajat untuk mengumpulkan informasi dari semua musuh. setelah itu bot akan menentukan target yang akan ditembak. Penentuan target ini berdasarkan jarak target dan banyaknya energi target. Bot ini akan menghitung nilai dari setiap tank musuh berdasarkan rumus,

$$V = \frac{dw}{d} + \frac{Ew}{E}$$

V = value

dw = distance weight (seberapa penting jarak musuh)

d = jarak musuh

Ew = energy weight (seberapa penting energi musuh)

E = energi musuh

Kemudian bot akan menargetkan musuh dengan nilai tertinggi.

Selain strategi *targeting*, bot juga melakukan perhitungan terhadap besarnya kekuatan tembakan yang akan dilakukan. Bot akan menghitung setiap kemungkinan kekuatan tembakan (dari 0.1 hingga 3 dengan *increment* sebesar 0.1) dan memberikan nilai. Nilai dari setiap kekuatan tembakan yang diuji dihitung berdasarkan tiga aspek yaitu, *hit probability*, *damage*, dan *energy cost* dengan persamaan,

$$V = P(\text{hit}) \times D - C$$

V = value

$P(\text{hit})$ = hit probability

D = damage

C = energy cost

Untuk menentukan *hit probability*, diperlukan beberapa parameter yakni, jarak target, stabilitas target, kecepatan target, dan waktu tempuh peluru. Ketidakteraturan target (*erraticness*) dihitung berdasarkan perbandingan kecepatan dan arah dari dua informasi tank musuh yang didapat dari dua kali *radar scan*. Setiap parameter ini dihitung dengan persamaan,

$$d_f = 1 - \frac{d}{d_{\max}}$$

$$\eta = \frac{\frac{|\theta_1 - \theta_0|}{180^\circ} + \frac{|v_1 - v_2|}{v_{\max}}}{2}$$

$$\xi = 1 - \eta$$

$$v_f = 1 - \frac{v}{v_{\max}}$$

$$t_f = e^{(-\lambda \times t_{\text{bullet}})}$$

$$P(\text{hit}) = d_f \times \xi \times v_f \times t_f$$

$P(\text{hit})$ = hit probability

d_f = distance factor

d = distance to target

d_{max} = maximum distance

η = erraticness

θ = target direction

ξ = stability factor

v_f = target speed factor

v = target speed

v_{max} = maximum speed

t_f = bullet travel time factor

λ = decay factor

t_{bullet} = bullet travel time

Kemudian untuk menghitung *damage* yang diberikan peluru dapat dihitung dengan persamaan,

$$D(x) = \begin{cases} 4x & , x \leq 1 \\ 4x + 2(x - 1) & , x > 1 \end{cases}$$

$D(x)$ = damage

x = firepower

Energy cost dari suatu kekuatan peluru sama dengan kekuatan peluru tersebut. Setelah menghitung nilai dari semua kemungkinan kekuatan peluru, bot akan menembak target dengan kekuatan peluru dengan nilai tertinggi.

a. Himpunan Kandidat

Himpunan kandidat mencakup semua tindakan yang dapat diambil oleh bot dalam setiap *turn* permainan, yaitu bot dapat melakukan 360 *scan*, bot dapat menentukan setiap *value* dari semua bot musuh yang di-*scan*, bot dapat menentukan arah tembakan yang mungkin mengenai musuh, dan bot dapat menentukan setiap *value* dari setiap kandidat besar kekuatan tembakan.

b. Himpunan Solusi

Himpunan solusi adalah kumpulan tindakan yang telah diambil oleh bot selama pertandingan. Ini termasuk menentukan target, bergerak mendekat ke arah musuh, dan melakukan tembakan.

c. Fungsi Solusi

Fungsi solusi menentukan apakah himpunan solusi yang dibangun sudah mencapai kondisi yang optimal atau cukup baik untuk memenuhi tujuan bot. Dalam hal ini, bot dianggap telah mencapai solusi sementara jika berhasil menarget musuh yang dekat dan berenergi rendah serta menembak dengan hasil maksimal dan risiko minimal. Bot tidak memiliki strategi jangka panjang, sehingga fungsi solusi lebih

bersifat lokal dalam konteks keputusan-keputusan kecil yang diambil setiap saat.

d. Fungsi Seleksi

Fungsi seleksi dalam bot ini didasarkan pada strategi greedy yang menggunakan heuristik berbasis *targeting* dan *best firepower*. Dalam *targeting*, bot menggunakan taktik menentukan target dengan jarak terdekat dan energi terendah. Setiap faktor jarak dan energi juga dipengaruhi seberapa penting faktor tersebut. Dalam hal menembak, bot akan menentukan arah tembakan dengan *linear targeting* dan menentukan besar kekuatan tembakan. Taktik yang digunakan adalah menghitung *risk and reward* dari setiap kemungkinan kekuatan tembakan dan memilih yang paling bagus.

e. Fungsi Kelayakan

Fungsi kelayakan memastikan bahwa setiap tindakan yang dipilih layak dilakukan tanpa melanggar batasan yang ada. Bot akan selalu mendekat ke arah target yang sudah dipilih agar memperbesar kemungkinan tembakan mengenai musuh. Namun, jika dirasa jarak terlalu dekat, bot akan mundur beberapa unit jarak.

f. Fungsi Objektif

Fungsi objektif dalam algoritma greedy ini adalah memaksimalkan membunuh musuh dengan memilih musuh berenergi paling rendah dan juga

paling dekat. Selain itu, fungsi objektif juga dapat dikaitkan dengan memaksimalkan keputusan besar kekuatan penembakan agar tembakan yang dirasa kecil kemungkinan mengenai musuh tidak akan mengurangi begitu banyak energi.

3.2.3. Algoritma Greedy Bot ASU

Bot ASU (*Active Stealth Unit*) adalah bot dengan pendekatan *greedy* untuk strategi defensif. Bot ini mula-mula akan membagi arena menjadi suatu *grid* berukuran 5x5. Bot kemudian akan menghitung *danger value* dari setiap zona berdasarkan posisi musuh dan banyaknya energi musuh. Untuk setiap musuh, bot akan menambahkan nilai *danger* sebesar jumlah energi musuh ke zona yang ditempati oleh musuh tersebut. Nilai *danger* yang diberikan dengan cara ini tidaklah permanen dan selalu dihitung ulang karena posisi musuh yang selalu berpindah-pindah. Namun, terdapat cara penambahan nilai *danger* yang permanen terhadap suatu zona. Jika bot ini terkena tembakan, bot akan menambahkan nilai *danger* permanen ke zona tempat bot ini berada.

Bot ASU kemudian akan berpindah tempat dengan memilih zona yang memiliki *danger value* terkecil. Pada implementasi algoritmanya, bot ini akan berpindah ke zona yang dipilih dengan memilih koordinat acak yang berada di dalam zona tersebut. Strategi penembakan dari bot ini sangatlah sederhana, yakni langsung menembak jika berhasil *scan* bot musuh.

a. Himpunan Kandidat

Himpunan kandidat mencakup semua tindakan yang dapat diambil oleh bot dalam setiap

turn permainan, yaitu bot dapat mendeteksi semua musuh, memperhitungkan area aman, dan belajar dari pengalaman dengan menandai area tempat terkena tembakan.

b. Himpunan Solusi

Himpunan solusi adalah kumpulan tindakan yang telah diambil oleh bot selama pertandingan. Ini termasuk membagi arena menjadi *grid* 5x5, memperhitungkan area yang memiliki musuh sebagai zona bahaya, menandai area tempat terkena tembakan, dan langsung menembak musuh jika berhasil di-*scan*.

c. Fungsi Solusi

Fungsi solusi menentukan apakah himpunan solusi yang dibangun sudah mencapai kondisi yang optimal atau cukup baik untuk memenuhi tujuan bot. Dalam hal ini, bot dianggap telah mencapai solusi sementara jika berhasil pergi ke tempat yang tidak berbahaya dan jarang terkena tembakan. Solusi ini juga dipengaruhi oleh faktor bahwa bot bergerak ke tempat acak pada zona yang dianggap aman.

d. Fungsi Seleksi

Fungsi seleksi dalam bot ini didasarkan pada strategi *greedy* yang menggunakan heuristik berbasis *safest area*. Dalam hal ini, bot akan membagi arena menjadi banyak zona dan menentukan zona bahaya dan zona aman berdasarkan banyak musuh, musuh berenergi tinggi,

dan zona tempat terkena tembakan. Bot akan mencari dan pergi ke area yang dianggap paling aman.

e. Fungsi Kelayakan

Fungsi kelayakan memastikan bahwa setiap tindakan yang dipilih layak dilakukan tanpa melanggar batasan yang ada. Bot melakukan sedikit manuver dan berbelok agar sulit ditembak saat ingin pergi ke area aman. Agar perhitungan zona bahaya semakin baik, bot juga memperbaharui area-area tempat bot terkena tembakan.

f. Fungsi Objektif

Fungsi objektif dalam algoritma greedy ini adalah meminimalkan berada di tempat dengan banyak bot dan tempat yang sering terkena tembakan.

3.2.4. Algoritma Greedy Bot PNIS

Strategi algoritma greedy dalam bot PNIS, Predictive enemy Neutralization & Intelligent Shooting, ini berfokus pada tembakan prediktif dan pergerakan adaptif yang dibantu oleh *radar lock*, di mana keputusan diambil berdasarkan kondisi saat ini tanpa mempertimbangkan dampak jangka panjang. Bot menggunakan prediksi linear untuk menentukan posisi musuh di masa depan. Perhitungan dilakukan dengan rumus:

$$PredictedX = e.X + \cos(enemyDirection) \times enemySpeed \times timeToHit$$

$$PredictedY = e.Y + \sin(enemyDirection) \times enemySpeed \times timeToHit$$

dimana *timeToHit* dihitung sebagai:

$$timeToHit = \frac{distance}{bulletSpeed}$$

dengan *bulletSpeed* ditentukan oleh game engine, yaitu:

$$bulletSpeed = 20 - (3 \times firePower)$$

Bot secara *greedy* menembak ke titik yang diprediksi sebagai lokasi musuh berdasarkan kecepatan dan arah musuh saat ini, tanpa mempertimbangkan kemungkinan perubahan arah atau kecepatan lawan. Keputusan ini lokal optimal karena memilih posisi terbaik berdasarkan informasi saat ini, tetapi tidak memperhitungkan strategi lawan.

Dalam *AdaptiveMovement*, bot membuat keputusan berdasarkan jarak ke musuh. Jika jarak lebih dari 500 unit, musuh dianggap jauh, bot bergerak maju ke arah musuh dengan sudut tertentu untuk menghindari tembakan langsung. Jika jarak kurang dari 100 unit, musuh dianggap dekat, bot melakukan *strafing* (gerakan menyamping) secara acak. Sedangkan, ketika bot dianggap *midrange* atau di pertengahan, bot tidak langsung maju atau mundur, tetapi bergerak menyamping dengan kemungkinan mengubah arah secara acak.

Dalam setiap langkahnya, bot memilih tindakan yang terbaik secara lokal, seperti menghindari serangan dengan pergerakan lateral, menembak dengan prediksi posisi musuh, dan menyesuaikan arah radar agar tetap mengunci musuh. Berikut adalah pemetaan strategi bot ini terhadap elemen-elemen algoritma *greedy*:

a. Himpunan kandidat

Himpunan kandidat mencakup semua tindakan yang dapat diambil oleh bot dalam setiap *turn*

permainan, yaitu bot dapat mengarahkan radar ke posisi musuh (*radar lock*), bot akan melakukan 360 scan ketika tidak ada musuh yang ter-*lock*, bot dapat memperkirakan posisi musuh berdasarkan kecepatan dan arah gerak musuh dan menembak musuh tersebut, bot juga dapat melakukan pergerakan sesuai dengan jarak bot terhadap bot yang di-*lock* oleh radar.

b. Himpunan solusi

Himpunan solusi adalah kumpulan dari kandidat yang telah dipilih oleh bot, seperti posisi dan pergerakan bot yang berdasarkan jarak ke musuh dan arah radar yang mengunci musuh.

c. Fungsi solusi

Fungsi solusi menentukan apakah bot sudah mencapai tujuan utamanya, yaitu memaksimalkan *survivability* dan efektivitas serangan. Solusi dikatakan valid jika bot tidak tertembak terlalu sering dan habis energi, bot terus mengunci target dengan radar dan tidak kehilangan informasi tentang posisi musuh, dan serangan bot mengenai musuh.

d. Fungsi seleksi

Fungsi seleksi dalam algoritma ini memilih kandidat terbaik berdasarkan heuristik berbasis keadaan musuh yang di-*lock* oleh radar. Bot memilih untuk menembak ke prediksi posisi musuh berdasarkan posisi, kecepatan dan arah gerak musuh saat ini. Bot juga akan menyesuaikan pergerakan berdasarkan jarak

bot ke musuh. Bot juga menyesuaikan *firePower* sesuai jarak ke musuh.

e. Fungsi kelayakan

Fungsi ini memastikan bahwa setiap tindakan yang dipilih masih memungkinkan untuk dilakukan dalam batasan permainan. Contohnya, dalam kasus bot ini adalah ketika bot menabrak tembok atau bot lain, bot akan merubah gerakan arahnya ke sebaliknya. Bot juga hanya menembak musuh ketika ter-*scan* oleh radar.

f. Fungsi objektif

Fungsi objektif dalam algoritma ini adalah memaksimalkan peluang bertahan hidup dengan bergerak berdasarkan jarak ke posisi musuh dan akurasi serangan dengan *radar lock* dan prediksi posisi musuh sebelum mulai menembak berdasarkan keadaan saat ini.

3.3. Analisis Efisiensi dan Efektivitas Algoritma Greedy

3.3.1. Analisis Bot Obi-Wan Tanknobi

Bot Obi-Wan Tanknobi menggunakan pendekatan greedy dengan prioritas utama mempertahankan “high ground” di arena, yakni menempati bagian atas. Bot ini bertujuan untuk mengambil keuntungan dari posisinya dalam pertarungan.

Selain itu, bot ini mengandalkan reaksi cepat terhadap perubahan energi lawan tanpa mempertimbangkan langkah jangka panjang atau memprediksi pergerakan lawan.

a. Keunggulan Strategi

1. Responsif dan Adaptif

Bot ini responsif dan adaptif terhadap perubahan dalam pertempuran, terutama pada perubahan energi lawan. Dengan mendeteksi penurunan energi lawan, bot dapat segera melakukan gerakan menghindar tanpa memerlukan sistem prediksi yang kompleks. Kecepatan reaksi ini membuat bot sulit ditembak oleh lawan yang mengandalkan strategi statis atau pola tembakan tetap.

2. Keunggulan Posisi

Penempatan bot di bagian atas arena memberi bot kendali terhadap area pertempuran. Bot dapat mengurangi kemungkinan diserang dari berbagai arah.

3. Efektif pada Awal Permainan

Bot ini sangat efektif dalam early game. Dengan segera mendapatkan posisi dan menembak setiap musuh yang terlihat, bot dapat memberikan tekanan kepada lawan yang belum memiliki strategi defensif yang baik.

b. Kelemahan Strategi

1. Tidak Memiliki Mekanisme Prediksi

Bot ini tidak memprediksi pergerakan lawan, sehingga banyak tembakannya bisa meleset terhadap musuh yang bergerak cepat atau memiliki pola pergerakan yang tidak terduga.

2. Pergerakan Menghindar yang Terbatas dan Terprediksi

Pergerakan menghindar mudah diprediksi karena hanya kanan dan kiri serta mekanisme menghindar hanya berdasarkan deteksi penurunan energi lawan. Jika lawan memiliki mekanisme prediksi terhadap pola gerakan bot ini, mereka dapat menembak ke posisi yang akan dituju saat bot melakukan pergerakan, sehingga strategi menghindar menjadi tidak efektif.

3. Ketergantungan pada High Ground

Strategi greedy yang terlalu fokus pada high ground membuat bot kurang adaptif terhadap kondisi lain dalam pertempuran. Jika ada bot lain yang berada di atas juga, bot ini lebih rentan terkena serangan.

4. Kurang Efektif untuk Jangka Panjang

Bot ini tidak memiliki strategi pengelolaan energi atau pemilihan target. Dalam pertarungan yang lebih panjang, bot ini cenderung menghabiskan energi dengan menembak sembarangan, sehingga dapat dikalahkan oleh lawan yang lebih efisien dalam mengatur penggunaan energi mereka.

3.3.2. Analisis Bot ASS

Bot ASS menggunakan pendekatan *greedy* dengan memprioritaskan musuh-musuh yang dekat dan memiliki energi terendah. Bot ini bertujuan untuk bertindak agresif

mengejar musuh dan dengan harapan dapat membunuh musuh berenergi rendah.

a. Keunggulan Strategi

1. Pemilihan Target yang Cepat

Bot dapat memilih target secara cepat dan mengutamakan bot berjarak dekat dan berenergi rendah. Hal ini membuat bot tidak perlu pergi ke tempat yang jauh karena fokus pada pertarungan jarak dekat. Kemudian, karena menargetkan bot berenergi rendah, bot ini mempunyai kemungkinan tinggi untuk melakukan *last hit* dan dihitung sebagai membunuh bot.

2. Penyesuaian Firepower

Bot menyesuaikan *firepower*-nya sesuai dengan berbagai faktor. Hal ini tentunya dapat menguntungkan bot dalam menghemat energinya dan tidak membuang-buang energi untuk tembakan yang kecil kemungkinan mengenai musuh. Jika jarak cukup dekat dan kemungkinan kena tinggi, bot akan mengeluarkan serangan dengan *damage* tinggi.

3. 360 Radar Scan

Bot melakukan *radar scan* 360 derajat setiap kali akan melakukan sesuatu. Hal ini membuat bot dapat melakukan perhitungan optimal

setelah mengetahui seluruh informasi dalam *battlefield*.

b. Kelemahan Strategi

1. Ketergantungan Parameter

Bot ini sangat bergantung pada berbagai parameter yang sangat memengaruhi perhitungannya. Jika tidak disesuaikan dengan baik, parameter ini akan membuat perhitungan bot tidak optimal dan menyebabkan pengambilan keputusan yang terkadang menjadi aneh. Keadaan kondisi *battlefield* yang selalu berubah-ubah menyebabkan sulitnya menentukan parameter yang paling optimal. Parameter-parameter ini juga tidak dinamis sehingga bot tidak dapat beradaptasi dengan kondisi *battlefield*.

2. Risiko Pergerakan Agresif

Bot ini sangat agresif karena akan selalu mengejar target agar kemungkinan tembakan mengenai musuh menjadi tinggi. Namun, hal ini juga menjadi kelemahan karena bot menjadi target yang juga mudah untuk ditembak. Selain itu, karena pergerakannya yang tidak kompleks membuat bot ini mudah terkena tembakan.

3. Kompleksitas Perhitungan

Perhitungan prediksi dan penentuan *firepower* yang cukup kompleks dapat menyebabkan bot perlu waktu dalam memroses tindakan

selanjutnya. Hal ini menyebabkan bot menjadi statis setelah melakukan *radar scan* dan membuat menjadi sasaran yang mudah untuk ditembak. Terkadang bot juga langsung berhenti dan tidak berbuat apa-apa ketika melewati gilirannya.

3.3.3. Analisis Bot ASU

Bot ASU dirancang untuk bergerak secara *stealthy* di arena pertempuran. Bot ini secara terus-menerus memindai musuh dan membangun "peta bahaya" (*danger map*) dengan cara membagi arena menjadi *grid* 5x5. Pada tiap zona, nilai bahaya (*danger score*) ditentukan dari kehadiran musuh yakni, energi musuh yang berada di zona tersebut ditambahkan sebagai nilai bahaya dan tembakan yang mengenai (*bullet hits*) yakni, saat terkena peluru, zona pada posisi bot akan mendapatkan tambahan nilai bahaya tetap.

Setelah menghitung nilai bahaya dari tiap zona, bot akan memilih zona dengan nilai bahaya terendah dan kemudian bergerak ke titik acak di dalam zona tersebut (dengan *offset* acak) untuk membuat pergerakan lebih tidak terduga.

a. Keunggulan Strategi

1. Adaptasi Dinamis

Bot selalu memperbarui peta bahaya berdasarkan informasi musuh dan tempat terkena tembakan, sehingga bisa menavigasi ke wilayah yang relatif aman.

2. Pemanfaatan Grid

Pembagian arena menjadi 5x5 *grid* memudahkan bot dalam melakukan evaluasi cepat terhadap keseluruhan area sehingga pengambilan keputusan menjadi lebih efisien. Dibandingkan jika perlu mengevaluasi tiap titik koordinat, metode ini jauh lebih cepat dan efisien.

3. Stealth & Evasiveness

Strategi memilih wilayah dengan nilai bahaya terendah membuat bot cenderung menghindari area yang banyak musuh atau area dengan tingkat kerusakan tinggi akibat peluru, mengurangi kemungkinan terkena tembakan lawan.

b. Kelemahan Strategi

1. Keterlambatan Update Data

Jika musuh bergerak sangat dinamis dan cepat, peta bahaya berbasis *grid* menjadi kurang akurat karena data yang dihitung dengan cepat menjadi tidak sesuai dengan keadaan *battlefield* yang sebenarnya.

2. Kecenderungan Pergerakan Statis

Bot cenderung berada di satu zona terus menerus. Walaupun terdapat *random offset*, tetap saja rata-rata bot akan berada di suatu zona dalam jangka

waktu yang panjang. Lalu, jika ada zona dengan nilai yang sama, bot akan cenderung bergerak ke zona paling pojok kiri atas karena proses perhitungan *grid* dimulai dari pojok kiri atas ke pojok kanan bawah.

3. Reaksi yang Berisiko

Reaksi bot terhadap peristiwa terkena tembakan sangatlah berisiko. Bot perlu terkena tembakan terlebih dahulu lalu baru menambah nilai bahaya ke zona tempat terkena tembakan.

4. Boros Energi

Bot akan secara langsung menembak dengan *firepower* maksimal jika mendeteksi musuh. Hal ini akan sangat boros energi jika tembakan ditembak dari jauh atau musuh memiliki pergerakan dinamis. Semakin sering bot ini gagal mengenai musuh, semakin boros penggunaan energinya.

3.3.4. Analisis Bot PNIS

Bot PNIS berfokus pada tembakan prediktif dan pergerakan adaptif yang dibantu oleh radar lock, di mana keputusan diambil berdasarkan kondisi saat ini tanpa mempertimbangkan dampak jangka panjang.

a. Keunggulan Strategi

1. Predictive Aiming

PNIS menggunakan predictive aiming di mana ia tidak hanya menargetkan musuh saat ini, tetapi memprediksi posisi musuh di masa depan berdasarkan kecepatan dan arah musuh.

2. Dynamic Firepower

PNIS menyesuaikan firepower berdasarkan jarak bot ke musuh sehingga menyeimbangkan damage output dan efisiensi energi.

3. Radar Lock

PNIS memiliki mekanisme radar yang selalu disesuaikan agar tetap terkunci pada musuh. Hal ini mengurangi kemungkinan kehilangan target di tengah pertempuran.

4. Adaptive Movement

Pergerakan PNIS berubah tergantung pada jarak musuh. Jika jarak jauh, PNIS akan mendekati musuh dengan sudut acak agar tidak terkena tembakan musuh. Pada jarak menengah, PNIS bergerak secara defensif dengan strafing. Pada jarak dekat, PNIS menghindar dan bergerak secara agresif.

5. Collision Handling

Ketika PNIS bertabrakan dengan dinding dan bot lain, PNIS langsung melakukan perubahan arah sehingga PNIS menjadi target yang sulit ditembak.

6. Enemy Data Tracking

PNIS menyimpan data musuh dalam dictionary, termasuk posisi, kecepatan, arah, dan energi.

b. Kelemahan Strategi

1. Lemah terhadap Pergerakan Tidak Terduga

Sistem targeting berbasis linear prediction, jika musuh sering mengubah arah atau kecepatan, prediksi bisa meleset.

2. Kurang Optimal Menghadapi Banyak Musuh

PNIS fokus pada satu musuh saja dan tidak memiliki mekanisme yang baik untuk berpindah target atau mempertahankan kesadaran terhadap banyak lawan. Radar PNIS pun hanya menyesuaikan dengan target yang sedang dikunci, PNIS berisiko kehilangan informasi tentang musuh lain yang bisa menyerangnya dari arah berbeda.

3. Tidak ada Strategi Bertahan

Tidak ada strategi untuk retreat atau bertahan ketika dalam kondisi berbahaya, seperti saat energi rendah atau dikejar oleh bot lain.

4. Tidak Mempertimbangkan Energi

PNIS terus menyerang tanpa mempertimbangkan sisa energi dan keadaan lawan, sehingga bisa kehabisan tenaga lebih cepat di pertandingan yang panjang.

5. Kurang Adaptif terhadap Serangan Lawan

PNIS hanya bereaksi terhadap posisi dan jarak musuh, tetapi tidak menganalisis pola tembakan atau strategi lawan untuk menghindari serangan secara lebih efektif.

3.4 Algoritma Greedy Terpilih

Berdasarkan analisis empat bot dalam implementasi algoritma greedy, **PNIS Bot** terpilih sebagai yang terbaik dalam pendekatan greedy karena memiliki keseimbangan antara prediksi tembakan, pergerakan adaptif, efisiensi energi, dan sistem radar yang baik. Dibandingkan bot lain yang lebih fokus pada aspek tertentu seperti posisi, agresivitas, atau stealth, PNIS Bot memiliki fleksibilitas dalam menghadapi berbagai situasi di battlefield, menjadikannya lebih unggul secara teoretis. PNIS Bot juga memiliki sistem serang terbaik dengan SmartFire dan Radar Lock sehingga memungkinkan mendapatkan poin banyak dari *bullet damage* dan PNIS Bot memiliki AdaptiveMovement dan Strafing sehingga membuat PNIS Bot dapat bertahan hidup lebih lama.

Obi-Wan Tanknobi menggunakan strategi greedy berbasis penguasaan posisi high ground, tetapi tidak memiliki mekanisme prediksi musuh atau adaptasi pergerakan yang kompleks. Bot ini hanya bereaksi terhadap perubahan energi lawan dan cenderung memiliki pola gerakan yang dapat diprediksi. Sebaliknya, PNIS Bot memanfaatkan predictive aiming dan adaptive movement, sehingga lebih mampu menghadapi lawan yang bergerak dinamis dan tidak bergantung pada satu kondisi spesifik di arena.

ASSBot menerapkan strategi greedy yang sangat agresif dengan memprioritaskan lawan berenergi rendah di sekitar dan melakukan serangan dengan firepower optimal. Meskipun memiliki keunggulan dalam menyerang musuh dengan cepat, bot ini kurang memiliki strategi defensif dan pergerakannya mudah diprediksi oleh lawan. Sebaliknya, PNIS Bot menggunakan pendekatan yang lebih seimbang antara offense dan defense melalui radar lock, adaptive movement, dan collision handling, sehingga lebih sulit untuk dikalahkan dibandingkan ASSBot yang cenderung mengabaikan pertahanan.

ASUBot memiliki pendekatan stealth melalui danger map, di mana ia menghindari zona berbahaya dan mencari tempat yang lebih aman. Meskipun strategi ini efektif dalam menghindari serangan, ASUBot memiliki kelemahan dalam inisiatif menyerang, terutama karena reaksi terhadap serangan lawan sering kali terlambat. PNIS Bot, dengan predictive aiming dan dynamic firepower, mampu menyerang secara efisien tanpa membuang banyak energi, sehingga memiliki keunggulan dalam pertempuran jangka panjang dibandingkan ASUBot yang terlalu pasif.

Dari perbandingan ini, PNIS Bot unggul karena memiliki keseimbangan terbaik antara ofensif dan defensif, serta mampu menyesuaikan strategi berdasarkan kondisi arena. Dengan kombinasi predictive aiming, radar lock, adaptive movement, dan enemy tracking, PNIS Bot dapat lebih efektif dalam bertahan dan menyerang dibandingkan bot lain yang hanya mengandalkan satu aspek strategi greedy. Oleh karena itu, secara teoretis, PNIS Bot merupakan

bot terbaik dalam implementasi algoritma greedy di antara keempat bot.

BAB IV IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi Algoritma Greedy

4.1.1. Algoritma Greedy Bot Obi-Wan Tanknobi

```
public override void Run()
```

```
procedure Run()  
{ Loop utama bot }
```

KAMUS LOKAL

```
AdjustRadarForBodyTurn : bool  
AdjustGunForBodyTurn : bool  
AdjustRadarForGunTurn : bool  
IsRunning: bool
```

ALGORITMA

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

```
while (IsRunning) do  
    Strategy()
```

```
public override void OnScannedBot(ScannedBotEvent e)
```

```
procedure OnScannedBot(e : ScannedBotEvent)  
{ Prosedur yang dilakukan ketika bot ter-scan oleh radar}
```

KAMUS LOKAL

```
enemyId : string  
enemies : dictionary<string, EnemyInfo>  
lastEnergy : dictionary<string, double>
```

ALGORITMA

```
Fire(1)  
enemyId ← e.ScannedBotId
```

```
if (enemyId is in enemies) then
    lastEnergy[enemyId] ← enemies[enemyId].Energy

enemies[enemyId] ← EnemyInfo(e.X, e.Y, e.Energy)
```

private void Strategy()

```
procedure Strategy()
{ Prosedur strategi greedy }
```

KAMUS LOKAL

ALGORITMA

```
SetupPosition()

if (ShouldDodge()) then
    Evade()
```

private void SetupPosition()

```
procedure SetupPosition()
{ Prosedur untuk setup posisi bot }
```

KAMUS LOKAL

```
targetY: real
currentBearing: real
targetAngle: real
ArenaHeight: real
Y: real
Direction: real
```

ALGORITMA

```
if (Y > ArenaHeight * 0.9 and Direction = 0) then
    ->
    targetY ← ArenaHeight
    currentBearing ← NormalizeBearing(Direction)

if (currentBearing ≠ 90) then
    SetTurnLeft(NormalizeBearing(90 -
```



```

currentBearing))
    WaitFor(new TurnCompleCondition(this))

if (Y < targetY) then
    SetForward(targetY - Y)
    WaitFor(new MoveCompleteCondition(this))

targetAngle ← 0
if (NormalizeBearing(Direction) ≠ targetAngle) then
    SetTurnLeft(NormalizeBearing(targetAngle -
Direction))
    WaitFor(new TurnCompleteCondition(this))

```

private bool ShouldDodge()

```

function ShouldDodge() → boolean
{ Fungsi untuk menentukan apakah Bot harus
menghindar }

```

KAMUS LOKAL

```

enemyId: String
enemyShot: String
turnRadarLeft: bool

```

ALGORITMA

```

if (turnRadarLeft) then
    TurnGunLeft(180)
    turnRadarLeft ← false
else
    TurnGunRight(180)
    turnRadarLeft ← true

for enemy in enemies do
    enemyId ← enemy.key
    if (enemyId is in lastEnergy and enemyId is in
enemies) then
        if (enemies[enemyId].Energy <
lastEnergy[enemyId]) then
            enemyShot ← enemyId
            → true

```

→ false

```
procedure evade(enemyId: String)
{ Prosedur menghindari dari musuh }
KAMUS LOKAL
X: real

ALGORITMA
if (enemyId not in enemies) or (enemyId not in
lastEnergy) then
    →

if (X < ArenaWidth / 2) then
    SetForward(300)
else
    SetBack(300)
```

4.1.2. Algoritma Greedy Bot ASS

```
public override void Run()

procedure Run()
{ Loop utama bot }

ALGORITMA
MAX_DISTANCE = Math.Sqrt(Math.Pow(ArenaHeight, 2) +
Math.Pow(ArenaWidth, 2));
AdjustGunForBodyTurn = true;
AdjustRadarForBodyTurn = true;
AdjustRadarForGunTurn = true;
AddCustomEvent(new FullRadarScan(this));
TurnRadarRight(360);
```

```
public override void OnCustomEvent(CustomEvent evt)
```

```
procedure OnCustomEvent(evt : CustomEvent)  
{ Algoritma yang dipanggil saat ada custom event  
yang terjadi. Custom event di sini adalah ketika  
bot selesai melakukan radar scan 360 derajat}
```

ALGORITMA

```
if (evt.Condition is FullRadarScan)  
{  
    EnemyInfo target = FindTarget();  
  
    if (target != null)  
    {  
        double distance =  
DistanceTo(target.XPosition, target.YPosition);  
        double firepower = ChooseFirepower(target,  
distance);  
        double travelTime = distance / (20 - 3 *  
firepower); // Based on game rule  
  
        double enemyRadians = target.Direction *  
DEGREE_TO_RADIANS;  
        double predictedX = target.XPosition +  
target.Speed * Math.Cos(enemyRadians) * travelTime;  
        double predictedY = target.YPosition +  
target.Speed * Math.Sin(enemyRadians) * travelTime;  
  
        double gunAngle =  
RelativeEnemyAngle(predictedX, predictedY, true);  
  
TurnGunLeft(NormalizeRelativeAngle(gunAngle));  
        Fire(firepower);  
  
        double tankAngle =  
RelativeEnemyAngle(predictedX, predictedY);  
  
TurnLeft(NormalizeRelativeAngle(tankAngle));
```

```

        TurnRate = randomGenerator.NextDouble() * 6
- 3; // random from -3 to 3
        if (distance - DISTANCE_THRESHOLD > 0)
        {
            Forward(distance - DISTANCE_THRESHOLD);
        }
        else
        {
            Back(DISTANCE_THRESHOLD);
        }
    }
    TurnRadarRight(360);
}

```

private double ChooseFirepower(EnemyInfo enemy, double distance)

function double ChooseFirepower(EnemyInfo enemy, double distance)
 { Algoritma kalkulasi untuk menentukan firepower yang terbaik berdasarkan kondisi battle }

ALGORITMA

```

double bestValue = double.MinValue;
double bestFirepower = LOWEST_FIREPOWER;

double maxCandidate = Math.Min(MAX_ENERGY, Energy);
double pDistance = Math.Max(0, 1 - distance /
MAX_DISTANCE);
double stability = 1 - enemy.Erraticness;
double speedFactor = Math.Max(0.1, 1 - enemy.Speed
/ MAX_VELOCITY);

for (double candidate = LOWEST_FIREPOWER; candidate
<= maxCandidate; candidate += FIREPOWER_INCREMENT)
{
    double bulletSpeed = 20 - 3 * candidate; //
based on game rule

```

```

        double travelTime = distance / bulletSpeed;

        double pTravel = Math.Exp(-LAMDA * travelTime);

        double hitProbability = stability * pDistance *
pTravel * speedFactor;

        // based on game rule
        double damage = (candidate <= 1) ? (4 *
candidate) : (4 * candidate + 2 * (candidate - 1));

        double value = hitProbability * damage -
candidate;

        if (value > bestValue)
        {
            bestValue = value;
            bestFirepower = candidate;
        }
    }

    return bestFirepower;

```

```

public override void OnScannedBot(ScannedBotEvent
scannedBot)

```

```

procedure OnScannedBot(ScannedBotEvent scannedBot)
{ Algoritma yang dijalankan ketika ada bot yang
terdeteksi radar }

```

ALGORITMA

```

int id = scannedBot.ScannedBotId;
EnemyInfo enemy;
if (!enemies.ContainsKey(id))
{
    enemy = new EnemyInfo

```

```

        {
            Id = scannedBot.ScannedBotId,
            XPosition = scannedBot.X,
            YPosition = scannedBot.Y,
            Energy = scannedBot.Energy,
            Direction = scannedBot.Direction,
            Speed = scannedBot.Speed
        };
    }
else
{
    enemy = enemies[id];
    double directionDiff =
Math.Abs(NormalizeRelativeAngle(scannedBot.Direction
n - enemy.Direction));
    double directionFactor = directionDiff / 180.0;
    double speedDiff = Math.Abs(scannedBot.Speed -
enemy.Speed);
    double speedFactor = speedDiff / MAX_VELOCITY;
    double Erraticness = (directionFactor +
speedFactor) / 2.0; // average
    enemy.Erraticness = (enemy.Erraticness +
Erraticness) / 2.0; // average to smooth the
erraticness
    enemy.XPosition = scannedBot.X;
    enemy.YPosition = scannedBot.Y;
    enemy.Direction = scannedBot.Direction;
    enemy.Speed = scannedBot.Speed;
    enemy.Energy = scannedBot.Energy;
}

enemies[id] = enemy;
Rescan();

```

```
public override void OnBotDeath(BotDeathEvent deadBot)
```

```
procedure OnBotDeath(BotDeathEvent deadBot)  
{ Algoritma yang dijalankan ketika ada bot yang  
mati }
```

ALGORITMA

```
if (enemies.ContainsKey(deadBot.VictimId))  
{  
    enemies.Remove(deadBot.VictimId);  
}  
Rescan();
```

```
private double RelativeEnemyAngle(double BotX, double BotY,  
bool MoveGun = false)
```

```
function RelativeEnemyAngle(double BotX, double  
BotY, bool MoveGun)  
{ Algoritma untuk menentukan sudut relatif body  
tank terhadap suatu bot, jika MoveGun = true, sudut  
relatif yang dicari untuk gun angle}
```

ALGORITMA

```
double deltaX = BotX - X;  
double deltaY = BotY - Y;  
double headingRadians = Math.Atan2(deltaY, deltaX);  
double headingAngle = headingRadians *  
RADIANS_TO_DEGREE;  
double deltaAngle;  
  
if (MoveGun)  
{
```

```
        deltaAngle = headingAngle - GunDirection;
        return deltaAngle;
    }

    deltaAngle = headingAngle - Direction;
    return deltaAngle;
```

private EnemyInfo FindTarget()

function FindTarget()
{ Algoritma untuk menentukan target terdekat dan dengan energi terendah}

ALGORITMA

```
double bestValue = 0;
EnemyInfo bestTarget = null;
foreach (var enemy in enemies.Values)
{
    double distance = DistanceTo(enemy.XPosition,
    enemy.YPosition);
    double score = (DISTANCE_WEIGHT / (distance +
    1)) + (ENERGY_WEIGHT / enemy.Energy);
    if (score > bestValue)
    {
        bestValue = score;
        bestTarget = enemy;
    }
}

return bestTarget;
```

```
class EnemyInfo
```



```
class EnemyInfo  
{ ADT untuk menyimpan informasi musuh }
```

ALGORITMA

```
public int Id { get; set; }  
public double XPosition { get; set; }  
public double YPosition { get; set; }  
public double Energy { get; set; }  
public double Direction { get; set; }  
public double Speed { get; set; }  
public double Erraticness { get; set; } = 0;
```

class FullRadarScan : Condition

```
class FullRadarScan  
{ class custom event yang menandakan jika bot sudah  
melakukan 360 radar scan}
```

ALGORITMA

```
private ASSBot bot;  
private double previousRadarDirection;  
private double cumulativeRotation;  
  
public FullRadarScan(ASSBot bot)  
{  
    this.bot = bot;  
    previousRadarDirection = bot.RadarDirection;  
    cumulativeRotation = 0;  
}  
  
public override bool Test()  
{
```

```

        double delta =
NormalizeRelativeAngle(bot.RadarDirection -
previousRadarDirection);
        cumulativeRotation += Math.Abs(delta);
        previousRadarDirection = bot.RadarDirection;

        if (cumulativeRotation >= 360)
        {
            cumulativeRotation = 0;
            return true;
        }
        return false;
    }

    private double NormalizeRelativeAngle(double angle)
    {
        while (angle > 180) angle -= 360;
        while (angle < -180) angle += 360;
        return angle;
    }

```

4.1.3. Algoritma Greedy Bot ASU

```
public override void Run()
```

```
procedure Run()  
{ Prosedur main loop bot }
```

ALGORITMA

```
cellWidth = ArenaWidth / GRID_COLS;  
cellHeight = ArenaHeight / GRID_ROWS;  
  
GunTurnRate = GUN_ROTATION_SPEED;  
while (IsRunning)  
{  
    PointF safePoint = FindSafestCoordinate();  
    MoveTo(safePoint.X, safePoint.Y);  
}
```

```
public override void OnScannedBot(ScannedBotEvent  
scannedBot)
```

```
procedure OnScannedBot(ScannedBotEvent scannedBot)  
{ Algoritma yang dijalankan ketika ada bot yang  
terdeteksi radar }
```

ALGORITMA

```
Fire(FIREPOWER);  
int id = scannedBot.ScannedBotId;  
EnemyInfo enemy;  
if (!enemies.TryGetValue(id, out enemy))  
{  
    enemy = new EnemyInfo();  
}  
enemy.Id = id;  
enemy.XPosition = scannedBot.X;  
enemy.YPosition = scannedBot.Y;
```

```
enemy.Energy = scannedBot.Energy;
enemy.Direction = scannedBot.Direction;
enemy.Speed = scannedBot.Speed;
enemy.Distance = DistanceTo(enemy.XPosition,
enemy.YPosition);
enemies[id] = enemy;

Rescan();
```

```
public override void OnBotDeath(BotDeathEvent deadBot)
```

```
procedure OnBotDeath(BotDeathEvent deadBot)
{ Algoritma yang dijalankan ketika ada bot yang
mati }
```

ALGORITMA

```
if (enemies.ContainsKey(deadBot.VictimId))
{
    enemies.Remove(deadBot.VictimId);
}
Rescan();
```

```
public override void OnRoundStarted(RoundStartedEvent e)
```

```
procedure OnRoundStarted(RoundStartedEvent e)
{ Algoritma yang dijalankan ketika ronde baru
dimulai}
```

ALGORITMA

```
for (int i = 0; i < GRID_ROWS; i++)
{
    for (int j = 0; j < GRID_COLS; j++)
    {
        dangerGrid[i, j] = 0;
    }
}
```

```
public override void OnHitByBullet(HitByBulletEvent bullet)
```

```
procedure OnHitByBullet(HitByBulletEvent bullet)
{ Algoritma yang dijalankan ketika bot terkena
tembakan musuh }
```

ALGORITMA

```
int cellX = Math.Min((int)(X / cellWidth),
GRID_COLS - 1);
int cellY = Math.Min((int)(Y / cellHeight),
GRID_ROWS - 1);

dangerGrid[cellY, cellX] += BULLET_DANGER_LEVEL;
```

```
private PointF FindSafestCoordinate()
```

```
function FindSafestCoordinate()  
{ Fungsi untuk mengembalikan koordinat aman  
berdasarkan danger zone }
```

ALGORITMA

```
double[,] gridDanger = new double[GRID_ROWS,  
GRID_COLS];  
for (int i = 0; i < GRID_ROWS; i++)  
{  
    for (int j = 0; j < GRID_COLS; j++)  
    {  
        gridDanger[i, j] = dangerGrid[i, j];  
    }  
}  
  
foreach (var enemy in enemies.Values)  
{  
    int cellX = Math.Min((int)(enemy.XPosition /  
cellWidth), GRID_COLS - 1);  
    int cellY = Math.Min((int)(enemy.YPosition /  
cellHeight), GRID_ROWS - 1);  
    gridDanger[cellY, cellX] += enemy.Energy;  
}  
  
double minDanger = double.MaxValue;  
int bestRow = 0;  
int bestCol = 0;  
for (int i = 0; i < GRID_ROWS; i++)  
{  
    for (int j = 0; j < GRID_COLS; j++)  
    {  
        if (gridDanger[i, j] < minDanger)  
        {  
            minDanger = gridDanger[i, j];  
        }  
    }  
}
```

```

        bestRow = i;
        bestCol = j;
    }
}

float safeX = bestCol * cellWidth + cellWidth / 2;
float safeY = bestRow * cellHeight + cellHeight / 2;

safeX = safeX +
(float)(randomGenerator.NextDouble() * cellWidth /
2 - cellWidth / 4);
safeY = safeY +
(float)(randomGenerator.NextDouble() * cellHeight /
2 - cellHeight / 4);

return new PointF(safeX, safeY);

```

private void MoveTo(double x, double y)

procedure MoveTo(double x, double y)
{ Prosedur untuk bergerak ke suatu titik koordinat }

ALGORITMA

```

double angleToEnemy = RelativeEnemyAngle(x, y);
TurnLeft(NormalizeRelativeAngle(angleToEnemy));
TurnRate = randomGenerator.NextDouble() * 6 - 3; //
random from -3 to 3
TargetSpeed = SPEED;
Forward(DistanceTo(x, y));

```

class EnemyInfo

```
class EnemyInfo  
{ ADT untuk menyimpan informasi musuh }
```

ALGORITMA

```
public int Id { get; set; }  
public double XPosition { get; set; }  
public double YPosition { get; set; }  
public double Energy { get; set; }  
public double Direction { get; set; }  
public double Speed { get; set; }  
public double Distance { get; set; }
```

private double RelativeEnemyAngle(double BotX, double BotY)

```
function RelativeEnemyAngle(double BotX, double BotY)  
{ Fungsi untuk mengembalikan arah untuk memutar  
body tank menghadap musuh }
```

ALGORITMA

```
double deltaX = BotX - X;  
double deltaY = BotY - Y;  
double headingRadians = Math.Atan2(deltaY, deltaX);  
double headingAngle = headingRadians *  
RADIANS_TO_DEGREE;  
double deltaAngle;  
  
deltaAngle = headingAngle - Direction;  
return deltaAngle;
```


4.1.4. Algoritma Greedy Bot PNIS

```
public override void Run()
```

```
procedure Run()  
{ Loop utama bot }
```

KAMUS LOKAL

AdjustRadarForGunTurn: bool

IsRunning: bool

RadarTurnRemaining: real

ALGORITMA

AdjustRadarForGunTurn \leftarrow true

TurnRadarRight(360) {scan pertama}

while (IsRunning) do

 Strafe()

if (RadarTurnRemaining = 0) then

 TurnRadarRight(360)

```
public override void OnScannedBot(ScannedBotEvent e)
```

```
procedure OnScannedBot(e : ScannedBotEvent)  
{ Prosedur yang dilakukan ketika bot ter-scan oleh  
radar}
```

KAMUS LOKAL

distance: real

bearing: real

gunBearing: real

enemyId: String

ALGORITMA

distance \leftarrow DistanceTo(e.X, e.Y)

bearing \leftarrow BearingTo(e.X, e.Y)

gunBearing \leftarrow GunBearingTo(e.X, e.Y)

SetTurnLeft(bearing + 90)

SetTurnGunLeft(gunBearing)

```

RotateRadar(e.X, e.Y) {Prosedur untuk Radar Lock}

enemyId ← e.ScannedBotId.ToString()
enemies[enemyId] ← new EnemyInfo(e.X, e.Y,
e.Energy, e.Speed, e.Direction)

SmartFire(enemies[enemyId])
AdaptiveMovement(enemies[enemyId])

```

```

private void SmartFire(EnemyInfo e)

```

```

procedure SmartFire(e: EnemyInfo)
{ Prosedur untuk memprediksi posisi dan menembak
musuh }

```

KAMUS LOKAL

```

bulletSpeed: real
distance: real
timeToHit: real
predictedX: real
predictedY: real
gunBearing: real
firePower: real

```

ALGORITMA

```

bulletSpeed ← 20 - (3 * 2)
distance ← DistanceTo(e.X, e.Y)
timeToHit ← distance / bulletSpeed

predictedX ← e.X + cos(e.Direction * PI/180) *
e.speed * timeToHit
predictedY ← e.Y + sin(e.Direction * Pi/180) *
e.speed * timeToHit

gunBearing ← GunBearingTo(predictedX, predictedY)
SetTurnGunLeft(gunBearing)

if (distance < 200) then

```

```

        firePower ← 3
    else if (distance < 500) then
        firePower ← 2
    else
        firePower ← 1

    SetFire(firePower)

```

private void AdaptiveMovement(EnemyInfo e)

```

procedure AdaptiveMovement(e: EnemyInfo)
{ Prosedur untuk pergerakan bot berdasarkan jarak
ke musuh }

```

KAMUS LOKAL

```

distance: real
enemyBearing: real
moveForward: int

```

ALGORITMA

```

distance ← DistanceTo(e.X, e.Y)
enemyBearing ← BearingTo(e.X, e.Y)

if (distance > 500) then
    SetTurnLeft(enemyBearing + (Random(0,1) > 0.5 ?
30 : -30))
    SetForward(200)
else if (distance < 100) then
    moveForward ← (Random(0,1) > 0.5 ? 1 : -1)
    SetTurnLeft(enemyBearing + 90 * moveForward)
    SetForward(150 * moveForward)
else
    if (Random(0,1) > 0.7) then
        moveForward ← -moveForward
    SetTurnLeft(enemyBearing + 90)
    SetForward(100 * moveForward)

```

```
private void RotateRadar(double x, double y)
```

```
procedure RotateRadar(x: real, y: real)  
{ Prosedur untuk radar lock }
```

KAMUS LOKAL

```
radarBearing: real  
margin: real
```

ALGORITMA

```
radarBearing ← RadarBearingTo(x, y)  
margin ← 5  
  
if radarBearing > 0 then  
    SetTurnRadarLeft(radarBearing + margin)  
else  
    SetTurnRadarRight(-radarBearing + margin)
```

```
private void Strafe()
```

```
procedure Strafe()  
{ Prosedur bergerak terus-menerus }
```

KAMUS LOKAL

```
moveForward: int
```

ALGORITMA

```
SetForward(10000 * moveForward)
```

```
public override void OnHitWall(HitWallEvent e)
```

```
public override void OnHitBot(HitBotEvent e)
```

```
procedure OnHitWall(e: HitWallEvent)  
{ Prosedur ketika bot menabrak tembok }
```

KAMUS LOKAL

```
moveForward: int
```

ALGORITMA

```

moveForward ← -moveForward

procedure OnHitBot(e: HitWallEvent)
{ Prosedur ketika bot menabrak bot lain }
KAMUS LOKAL
moveForward: int

ALGORITMA
moveForward ← -moveForward

```

4.2. Hasil Pengujian

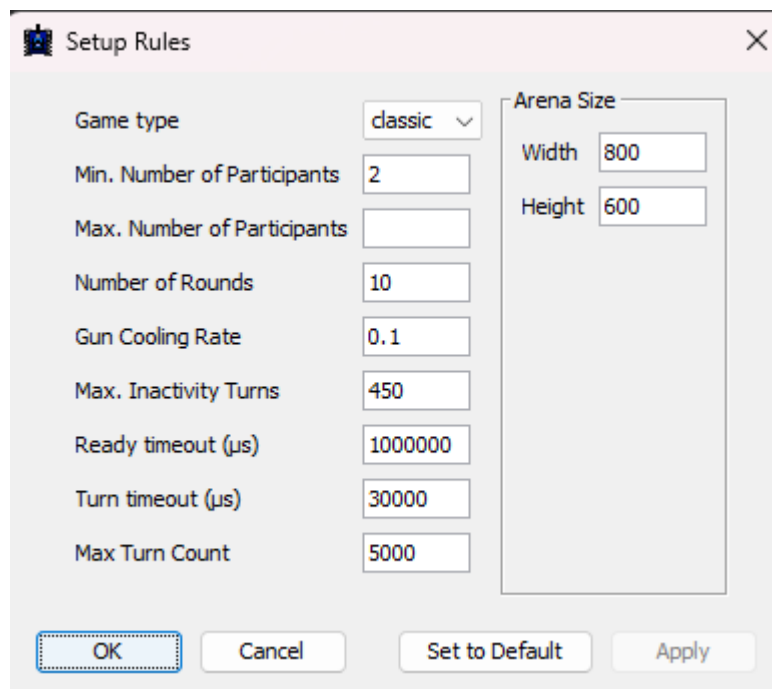
1. Pengujian 1

Gambar 4.2.1 Setup Rules 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	PNIS Bot 1.0	3087	1050	120	1644	220	52	0	4	5	1
2	ASUBot 1.0	2814	1200	180	1248	182	4	0	6	2	2
3	ObiWanTanknobi 1.0	986	600	0	360	6	19	0	0	3	4
4	ASSBot 1.0	310	150	0	156	3	1	0	0	0	3

Gambar 4.2.2 Hasil Pengujian 1

2. Pengujian 2



Setup Rules

Game type: classic

Min. Number of Participants: 2

Max. Number of Participants:

Number of Rounds: 10

Gun Cooling Rate: 0.1

Max. Inactivity Turns: 450

Ready timeout (µs): 1000000

Turn timeout (µs): 30000

Max Turn Count: 5000

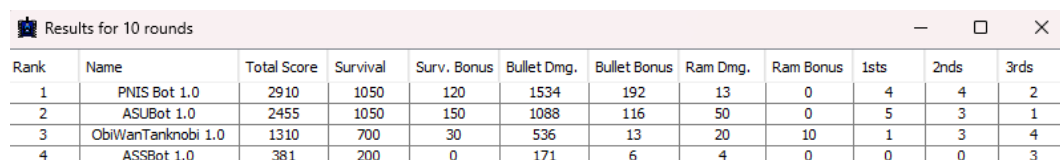
Arena Size

Width: 800

Height: 600

OK Cancel Set to Default Apply

Gambar 4.2.3 Setup Rules Pengujian 2



Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	PNIS Bot 1.0	2910	1050	120	1534	192	13	0	4	4	2
2	ASUBot 1.0	2455	1050	150	1088	116	50	0	5	3	1
3	ObiWanTanknobi 1.0	1310	700	30	536	13	20	10	1	3	4
4	ASSBot 1.0	381	200	0	171	6	4	0	0	0	3

Gambar 4.2.4 Hasil Pengujian 2

3. Pengujian 3

Setup Rules

Game type: classic

Min. Number of Participants: 2

Max. Number of Participants:

Number of Rounds: 10

Gun Cooling Rate: 0.1

Max. Inactivity Turns: 450

Ready timeout (μs): 1000000

Turn timeout (μs): 30000

Max Turn Count: 5000

Arena Size

Width: 1600

Height: 1200

OK Cancel Set to Default Apply

Gambar 4.2.5 Setup Rules Pengujian 3

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	PNIS Bot 1.0	1397	500	60	736	96	5	0	3	1	2
2	ASUBot 1.0	1326	450	60	720	83	13	0	3	2	1
3	ObiWanTanknobi 1.0	565	350	30	168	9	7	0	0	2	2
4	ASSBot 1.0	297	200	0	95	0	2	0	0	1	1

Gambar 4.2.6 Hasil Pengujian 3

4.3. Analisis Hasil Pengujian

Berdasarkan hasil pengujian, dapat dilihat bahwa ranking setiap bot sama. PNIS Bot unggul dalam setiap kondisi pengujian disusul langsung oleh ASU Bot. Berdasarkan pengujian satu dan dua, ASU Bot unggul dalam *survivability score*. Namun, PNIS Bot unggul dalam *bullet damage score*. Dapat ditarik kesimpulan bahwa ASU

Bot unggul dalam bertahan hidup paling lama. Hal ini dapat dijelaskan karena strategi ASU Bot yang mengutamakan berada di area-area aman. Di sisi lain, PNIS Bot sangat akurat dalam menembak. Hal ini juga dapat dijelaskan karena strategi PNIS Bot yang menggunakan sistem *predictive aiming* dengan sistem *linear targeting*. Selain itu, PNIS Bot juga hanya fokus pada satu musuh saja. PNIS Bot juga memiliki movement yang sulit diprediksi sehingga sulit untuk ditembak.

Pada pengujian ketiga, terlihat perbedaan nilai ASU Bot dan PNIS Bot menjadi sangat kecil. Hal ini mungkin disebabkan arena yang besar membuat ASU Bot menjadi lebih aman tetapi, hal ini membuat ASU Bot menjadi semakin jarang menyerang musuh. Pada pengujian ketiga ini, sering kali pada akhir ronde tersisa ASU Bot dan PNIS Bot dalam keadaan berjauhan menyebabkan mereka tidak bisa saling menyerang karena tidak terdeteksi radar.

ObiWanTankNobi paling unggul berada pada pengujian kedua. Hal ini mungkin karena bot ini mendapatkan *advantage* ketika waktu satu ronde sangat cepat. Hal ini sesuai dengan hasil analisis dimana Obi-Wan Tanknobi cocok untuk early game, tetapi tidak cocok untuk late game.

ASS Bot terlihat kalah dalam setiap pengujian. Dalam tiga kali pengujian, ASS Bot hanya berhasil meraih poin dari *bullet damage* dan sedikit *ram damage*, hal ini menunjukkan bahwa strategi ASS Bot dalam menyerang musuh sudah berjalan, tetapi strategi yang dilakukan oleh ASS Bot sangatlah berisiko yang menyebabkan ASS Bot menjadi sering terkena tembakan sehingga memiliki

survivability score yang sangat rendah. Selain itu, ASS Bot juga tidak memiliki pergerakan ketika terserang. ASS Bot cocok di keadaan dimana ASS Bot tidak ditarget oleh bot manapun sehingga ASS Bot dapat menjalankan strateginya dengan baik.

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Algoritma greedy dalam Robocode Tank Royale merupakan pendekatan yang berfokus pada pengambilan keputusan berdasarkan keuntungan jangka pendek yang paling optimal. Dengan memanfaatkan heuristik yang tepat, strategi ini memungkinkan bot untuk bertahan lebih lama, menyerang lebih efektif, dan menghindari bahaya secara langsung. Pendekatan greedy memiliki keunggulan dalam kesederhanaan implementasi dan efisiensi komputasi karena tidak memerlukan perhitungan kompleks. Dalam banyak skenario, bot yang menggunakan strategi greedy mampu bertahan lebih lama, menyerang lebih efektif, dan menghindari bahaya.

Namun, algoritma greedy juga memiliki beberapa keterbatasan. Salah satu kelemahannya adalah kurang optimal dalam skenario jangka panjang di mana keputusan terbaik saat ini belum tentu mengarah pada hasil terbaik secara keseluruhan. Selain itu, bot yang menggunakan strategi greedy rentan terhadap eksploitasi jika musuh dapat mengenali pola pergerakan dan serangannya. Hal ini membuat bot dengan pendekatan greedy lebih mudah dikalahkan oleh lawan yang mampu beradaptasi. Selain itu, keputusan yang hanya mempertimbangkan keuntungan sesaat dapat menyebabkan bot terjebak dalam situasi suboptimal, seperti terus mengejar musuh tanpa mempertimbangkan bahaya dari bot lain di sekitar.

5.2. Saran

Efektivitas bot berbasis algoritma greedy dalam Robocode Tank Royale dapat ditingkatkan dengan beberapa hal. Pertama,

meskipun pendekatan greedy cenderung memilih keputusan terbaik dalam satu langkah, integrasi dengan mekanisme prediksi dapat meningkatkan kinerja bot. Kedua, penggunaan heuristik yang lebih kompleks dapat membantu bot membuat keputusan lebih baik dalam situasi yang tidak pasti. Ketiga, pengujian dan simulasi dalam berbagai skenario sangat penting untuk mengoptimalkan algoritma. Sebaiknya bot diuji melawan berbagai jenis lawan dengan strategi yang berbeda-beda. Terakhir, pengembangan bot sebaiknya dilakukan secara iteratif, dengan memperhitungkan trade-off antara kesederhanaan dan efektivitas strategi. Dengan terus melakukan eksperimen dan iterasi, bot yang dikembangkan akan semakin efektif.

LAMPIRAN

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.		✓

Repository Github: https://github.com/V-Kleio/Tubes1_Tank-Top

DAFTAR PUSTAKA

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