TUGAS BESAR 1
IF2211 STRATEGI ALGORITMA

Dipersiapkan oleh:

Kelompok 11

Clarissa Nethania Tambunan	13523016
Mayla Yaffa Ludmilla	13523050
Noumisyifa Nabila Nareswari	13523058

Dosen Pengampu:

Dr. Nur Ulfa Maulidevi, S.T, M.Sc.
Dr. Ir. Rinaldi Munir, M.T.
Menterico Adrian, S.T, M.T.

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

2025

Daftar Isi

Bab 1
Deskripsi Tugas

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

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

1. Rounds dan Turns

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

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

   Pada setiap turn, sebuah bot dapat:

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

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

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

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

2. Batas Waktu Giliran

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

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

3. Energi

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

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

4. Peluru

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

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

5. Panas Meriam (Gun Heat)

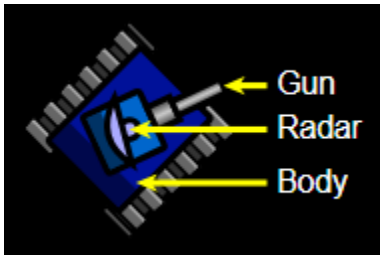
   Saat menembakkan peluru, meriam akan menjadi panas. Peluru yang lebih berat menghasilkan lebih banyak panas dibandingkan peluru yang lebih ringan. Ketika meriam terlalu panas, bot tidak dapat menembak hingga suhu meriam turun ke nol. Selain itu, meriam juga sudah dalam keadaan panas di awal round dan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

6. Tabrakan

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

7. Bagian Tubuh Tank

   Tubuh tank terdiri dari 3 bagian:



   Body adalah bagian utama dari tank yang digunakan untuk menggerakkan tank.
   Gun digunakan untuk menembakkan peluru dan dapat berputar bersama *body* atau independen dari *body*.
   Radar digunakan untuk memindai posisi musuh dan dapat berputar bersama *body* atau independen dari *body*.
8. Pergerakan

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

9. Berbelok

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

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

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

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

10. Pemindaian

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

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

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

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

11. Skor

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

- **Bullet Damage:** Bot mendapatkan **poin sebesar *damage*** yang dibuat kepada bot musuh menggunakan peluru.
- **Bullet Damage Bonus:** Apabila peluru berhasil membunuh bot musuh, bot mendapatkan **poin sebesar 20% dari *damage*** yang dibuat kepada musuh yang terbunuh.
- **Survival Score:** Setiap ada bot yang mati, bot 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 2
Landasan Teori

2.1 Dasar Teori Algoritma *Greedy*

Algoritma *greedy* merupakan pendekatan yang sering digunakan dalam menyelesaikan masalah optimasi, di mana keputusan terbaik diambil pada setiap langkah tanpa mempertimbangkan konsekuensi panjang yang umumnya persoalan optimasi ini terbagi menjadi dua jenis, yaitu maksimasi (*maximization*) dan minimasi (*minimization*). Pendekatan ini bertujuan untuk mencapai solusi optimal secara keseluruhan dengan membuat pilihan optimal lokal pada setiap tahap. Beberapa elemen utama dalam algoritma *greedy* yaitu,

- Himpunan Kandidat (C) adalah kumpulan elemen yang akan dipertimbangkan untuk dimasukkan ke dalam solusi.
- Himpunan Solusi (S) adalah kumpulan elemen yang telah dipilih dan membentuk solusi sementara.
- Fungsi Solusi menentukan apakah himpunan solusi saat ini memenuhi kriteria sebagai solusi yang valid.
- Fungsi Seleksi untuk memilih elemen berikutnya dari himpunan kandidat berdasarkan strategi *greedy* tertentu.
- Fungsi Kelayakan untuk memeriksa apakah elemen yang dipilih layak atau tidak untuk dimasukkan ke dalam himpunan solusi.
- Fungsi Objektif untuk mengukur kualitas solusi saat ini, biasanya dengan ujian memaksimalkan atau meminimalkan nilai tertentu.

Contoh permasalahan yang dapat diselesaikan dengan algoritma *greedy* yaitu masalah penukaran uang (*Coin Exchange Problem*) dengan menentukan kombinasi koin untuk mencapai nilai tertentu dengan jumlah koin seminimal mungkin, masalah pemilihan aktivitas (*Activity Selection Problem*) dengan memilih sejumlah aktivitas yang tidak saling bertentangan dalam hal waktu pelaksanaan untuk memaksimalkan jumlah aktivitas yang dapat dilakukan, masalah *knapsack* dengan memilih sejumlah objek dengan berat dan nilai tertentu untuk dimasukkan ke dalam tas dengan kapasitas terbatas sehingga nilai total yang didapat semaksimal mungkin, pohon merentang minimum (*Minimum Spanning Tree*) dengan menemukan pohon yang menghubungkan semua titik dalam graf dengan total bobot sisi minimum, masalah lintasan terpendek (*Shortest Path*) dengan menemukan jalur terpendek antara dua titik dalam graf, masalah pengkodean Huffman (*Huffman Code*), dan pecahan Mesir (*Egyptian Friction*) Pada beberapa permasalahan seperti *Activity Selection Problem* membuktikan bahwa algoritma *greedy* dapat menghasilkan solusi optimal yaitu jumlah aktivitas yang dipilih adalah maksimal. Pembuktian ini didasarkan fakta bahwa memilih aktivitas selanjutnya sehingga memungkinkan pemilihan lebih banyak aktivitas selanjutnya sehingga memunginkan pemilihan lebih banyak aktivitas secara keseluruhan.

Meskipun algoritma *greedy* efektif untuk beberapa permasalahan, karena pendekatan ini merupakan pendekatan heuristik, pendekatan ini tidak selalu menghasilkan solusi yang paling optimal untuk semua jenis masalah. Walaupun tidak selalu menghasilkan solusi paling optimal, setidaknya pendekatan ini kemungkinan besar menghasilkan solusi yang mendekati paling optimal untuk menghasilkan program dengan algoritma yang lebih murah secara komputasional bila dibandingkan dengan *brute force* yang pasti menghasilkan solusi paling optimal. Oleh karena itu, penting untuk memahami karakteristik, tujuan, dan prioritas dari setiap permasalahan sebelum menerapkan algoritma *greedy*.

2.2 Cara Kerja Program Robocode

Pada tugas ini, bot dibangun menggunakan bahasa pemrograman C# dan juga API dari Robocode itu sendiri. Secara umum, setiap bot dibangun dari beberapa file yang berada dalam satu folder dengan ketentuan file sebagai berikut:

- **Bot.cmd**: Sebuah script batch digunakan untuk membangun sekaligus menyambungkan bot ke server lalu menjalankannya menggunakan runtime .NET untuk sistem berbasis Windows.
- **Bot.cs**: Script utama dimana algoritma dari bot tersebut diimplementasikan. Harus dipastikan file ini telah terhubung dengan API Robocode agar dapat memodifikasi bot.
- **Bot.csproj**: File proyek .NET yang menentukan bagaimana bot tersebut dibangun. Memberitahu bagaimana .NET dapat mengkompilasi bot tersebut.
- **Bot.sh**: Adalah script shell untuk menjalankan bot di Linux/macOS. Fungsinya mirip dengan file Bot.cmd namun untuk sistem berbasis UNIX.\
- **Bot.json**: File berisikan metadata bot tersebut seperti nama, deskripsi, nama pembuat, dan metadata lainnya.

Strategi algoritma *greedy* yang telah dirancang dapat diimplementasikan menjadi serangkaian gerakan yang dapat diakses langsung melalui API Robocode. Rangkaian gerakan dan fitur yang dapat dilakukan oleh suatu bot telah dijelaskan secara cukup rinci pada bagian deskripsi tugas. Ditinjau dari penggeraknya, mekanisme/fungsi yang dapat dilakukan suatu bot dapat terbagi menjadi dua tipe, yaitu *event-driven* dan *direct-execution*.

- **Fungsi *event-driven***: Adalah fungsi yang secara otomatis berjalan ketika suatu kejadian (*event*) terjadi. Contoh *event* dalam konteks Robocode adalah seperti *event* ketika bot terkena tembakan, *event* ketika bot menabrak dinding, dan berbagai *event* lainnya. Namun tidak semua kejadian dapat dijadikan sebagai pemicu dari suatu fungsi, hanya beberapa fungsi dan kejadian tertentu yang telah ditetapkan oleh API dari Robocode.
- **Fungsi *direct-execution***: Adalah fungsi yang secara eksplisit dan terurut (prosedural) dijalankan oleh program selama tidak ada fungsi *event-driven* yang berjalan. Fungsi ini biasanya digunakan dalam lingkup fungsi Run(), yaitu fungsi dari API Robocode yang dijalankan ketika bot mulai dijalankan. Selain itu, ketika ada fungsi yang ingin secara repetitif terus dijalankan selama bot masih hidup, sebuah fungsi dapat dijalankan dalam loop while(IsRunning) yang juga dalam fungsi Run().

Untuk menjalankan bot, pengguna perlu menjalankan GUI Robocode terlebih dahulu dengan menekan ikon GUI.jar atau menjalankannya melalui terminal dengan *command*:

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

Lalu pemain dapat menjalankan satu-satu Bot.cmd dalam folder di mana semua file yang membentuk bot tersebut berada (file cmd, cs, csproj, sh, dan json) jika belum pernah dijalankan sama sekali sebelumnya atau jika ada perubahan implementasi algoritma.

```
./Bot.cmd
```

Lalu, kembali ke GUI, pemain dapat memuat satu persatu bot yang akan diadu kemudian memulai permainannya.

Bab 3
Aplikasi Strategi *Greedy*

3.1 *Mapping* Persoalan Robocode dalam Algoritma *Greedy*

Dalam permainan Robocode, setiap bot melakukan strategi tertentu untuk bertahan dan meraih skor tertinggi. Strategi ini melibatkan cara menembak bot musuh, bergerak, dan lain-lain. Salah satu pendekatan yang dapat digunakan untuk merancang strategi bot adalah algoritma *greedy* yang bertujuan untuk membuat pilihan terbaik pada setiap langkahnya. Elemen-elemen dari algoritma *greedy* dapat dipetakan ke dalam konteks permainan Robocode sebagai berikut.

- Himpunan kandidat (C) dalam konteks robocode adalah kumpulan aksi yang dapat dilakukan oleh bot Robocode dalam satu turn. Aksi ini dapat berarti mengubah posisi dengan bergerak, memindai (scan) bot lain, menghindar bot lain, atau menembak bot lain.
- Himpunan solusi (S) dalam konteks robocode adalah kumpulan aksi dari himpunan kandidat yang telah diseleksi oleh fungsi seleksi berdasarkan heuristik tertentu.
- Fungsi solusi dalam konteks robocode adalah apakah himpunan solusi saat ini memenuhi kriteria sebagai solusi yang valid.
- Fungsi seleksi dalam konteks robocode adalah fungsi yang menyeleksi kumpulan aksi dari himpunan kandidat menjadi himpunan solusi berdasarkan heuristik dari algoritma *greedy* tertentu. Contohnya, heuristik yang memilih untuk terus menerus menembak bot lain atau menghindar dari tembok.
- Fungsi kelayakan dalam konteks robocode adalah fungsi yang mengecek apakah sebuah aksi dapat dilakukan. Misalnya, apakah energi bot cukup untuk menembak atau apakah bot dapat bergerak sejauh yang diinginkan.
- Fungsi objektif dalam konteks robocode adalah fungsi yang memaksimalkan kelangsungan hidup bot (survival score) dengan memanfaatkan survival bonus.

3.2 Eksplorasi 4 Alternatif Solusi *Greedy*

3.2.1 Bot Adudu Sebagai Alternatif Solusi *Greedy*

Bot Adudu adalah bot yang mementingkan tembakan ke bot musuh dan bergerak di jalur yang berbentuk kotak di bagian tengah arena apabila tidak ada gangguan eksternal. Bot ini terus menerus memindai bot musuh dan akan mundur jika terkena peluru atau bot lain.

Penerapan algoritma *greedy* dalam bot Adudu:

- Himpunan kandidat (C) : Semua gerakan/mekanisme bot yang disediakan oleh API Robocode.
- Himpunan solusi (S) :
    - Jika tidak ada gangguan, himpunan solusinya adalah bergerak maju dalam jalur berbentuk kotak.
    - Jika terkena peluru dari musuh, himpunan solusinya adalah bergerak mundur untuk mengecoh sebelum maju kembali.
    - Jika menabrak tembok, himpunan solusinya adalah bergerak menuju tengah arena
- Fungsi solusi : Tidak ada.
- Fungsi seleksi : Selalu memilih untuk bergerak dalam jalur yang berbentuk kotak. Di awal setiap ronde mencari target posisi di tengah untuk dituju.
    - Jika terkena peluru dari musuh, memilih untuk bergerak mundur untuk mengecoh sebelum maju kembali.

    - Jika menabrak tembok, mencari target posisi di tengah arena dan bergerak ke arah itu.
- Fungsi kelayakan : Mengecek sisa energi yang dimiliki oleh bot. Jika energi bot kurang dari 10, bot tidak akan menembak bot musuh.
- Fungsi Objektif : Tidak ada.

3.2.2 Bot Straight Sebagai Alternatif Solusi *Greedy*

Bot Straight adalah bot yang bergerak secara horizontal lurus di tengah arena secara bolak-balik sambil memutarkan meriam dan pemindai secara 360 derajat terus menerus dan langsung menembak ketika menemukan bot lain dalam jangkauan pemindai.

Penerapan algoritma *greedy* dalam bot Straight:

- Himpunan kandidat (C) : Semua gerakan/mekanisme bot yang disediakan oleh API Robocode.
- Himpunan solusi (S) :
    - Bergerak menuju titik X=25 dan Y=ArenaHeight/2.
    - Bergerak menuju titik X=ArenaWidth-25 dan Y=ArenaHeight/2.
    - Bergerak maju sambil sejauh ArenaWidth-50 memutarkan meriam dan pemindai secara 360 derajat sambil menembak jika menemukan bot lain dalam jangkauan pemindai.
    - Bergerak mundur sejauh ArenaWidth-50 sambil memutarkan meriam dan pemindai secara 360 derajat sambil menembak jika menemukan bot lain dalam jangkauan pemindai.
    - Bergerak mundur sejauh 60 unit, menunggu 30 unit waktu, maju sejauh 60 unit.
    - Meriam dan pemindai berhenti berputar dan berhenti menembak.
- Fungsi solusi: Tidak ada
- Fungsi seleksi :
    - Jika *round* baru saja dimulai dan *spawn point* bot berada di sisi kiri arena, bot akan menjalankan solusi pertama pada himpunan solusi.
    - Jika *round* baru saja dimulai dan *spawn point* bot berada di sisi kanan arena, bot akan menjalankan solusi kedua pada himpunan solusi.
    - Jika bot sampai pada titik X = 25, maka bot akan menjalankan solusi ketiga pada himpunan solusi.
    - Jika bot sampai pada titik X = ArenaWidth-25, maka bot akan menjalankan solusi keempat pada himpunan solusi.
    - Jika bot menabrak bot lain, maka bot akan menjalankan solusi kelima pada himpunan solusi.
    - Jika energi bot <= 5,maka bot akan menjalankan solusi keenam pada himpunan solusi.
- Fungsi kelayakan : Tidak ada.
- Fungsi objektif : Memaksimalkan lingkup daerah yang berpotensi terdapat bot di saat itu dengan secara terus-menerus memutarkan pemindai dan meriam secara 360 derajat dan selalu menembak jika terdapat bot terpindai. Atau secara umum, memaksimalkan skor dengan selalu mencari bot di sekitar dan selalu menembak setiap ada bot yang terpindai. Sekaligus memaksimalkan usaha untuk tidak terkena tembakan dengan terus bergerak. Lalu ketika energi sudah sisa sedikit, bot akan meminimalkan energi yang dikeluarkan.

3.2.3 Bot Smart_Calendar1874 Sebagai Alternatif Solusi *Greedy*

Bot Smart_Calendar1874 adalah bot yang diam mendekati dinding namun tetap terpisah sejauh suatu margin lalu terus memutarkan meriam dan pemindai 360 derajat dan menembak jika terdapat bot yang terpindai. Namun ketika tertembak, bot akan bergerak menyusuri dinding arena sejauh 100 unit.

Penerapan algoritma *greedy* dalam bot Smart_Calendar1874:

- Himpunan kandidat (C) : Semua gerakan/mekanisme bot yang disediakan oleh API Robocode.
- Himpunan solusi (S) :
  - Bergerak menuju titik dinding terdekat lalu berbelok ke kiri.
  - Memutarkan meriam dan pemindai secara 360 derajat sambil menembak jika menemukan bot lain dalam jangkauan pemindai.
  - Bergerak maju menyusuri dinding arena sejauh 100 unit.
- Fungsi seleksi :
  - Jika *round* baru saja dimulai, bot akan menjalankan solusi pertama pada himpunan solusi.
  - Jika tidak ada *event* lain, bot akan terus menjalankan solusi kedua pada himpunan solusi.
  - Jika bot tertembak, maka bot akan menjalankan solusi ketiga pada himpunan solusi.
- Fungsi kelayakan : Tidak ada.
- Fungsi objektif : Atau secara umum, memaksimalkan skor dengan selalu mencari bot di sekitar dan selalu menembak setiap ada bot yang terpindai dan diam ketika tidak tertembak agar memaksimalkan kestabilan tembakan bot.

3.2.4 Bot CircularMG Sebagai Alternatif Solusi *Greedy*

Bot CircularMG adalah bot yang bergerak dalam pola lingkaran untuk menghindari serangan peluru dari musuh dan mendeteksi musuh dengan lebih efektif serta menyesuaikan kekuatan tembakan berdasarkan energi musuh.

Penerapan algoritma *greedy* dalam bot CircularMG:

- Himpunan kandidat (C) : Semua gerakan/mekanisme bot yang disediakan oleh API Robocode.
- Himpunan solusi (S) :
  - Jika tidak ada gangguan, himpunan solusinya adalah bergerak dalam pola melingkar secara terus-menerus agar dapat mendeteksi musuh dan menghindari tembakan lawan lebih efektif.
  - Jika bot bertemu lawan, himpunan solusinya adalah menyesuaikan kekuatan tembakan berdasarkan energi musuh di mana bot akan menembak dengan kekuatan maksimum apabila musuh memiliki energi rendah.
  - Jika terkena tembok, himpunan solusinya adalah bergerak mundur dan berbelok untuk melanjutkan pola lingkaran.
  - Jika bot terkena peluru, himpunan solusinya adalah menghindari peluru dengan bergerak ke kiri atau kanan tergantung dari arah peluru musuh.
- Fungsi seleksi :
  - Selalu memilih untuk bergerak dalam pola lingkaran untuk menghindari musuh.
  - Saat menemukan musuh di sekitar bot, memilih untuk menembak dengan kekuatan tembakan yang disesuaikan berdasarkan energi musuh.
  - Jika menabrak tembok, melakukan serangkaian gerakan untuk tetap dalam pola melingkar.
  - Jika terkena peluru, menghindar dengan bergerak ke kanan atau kiri tergantung dari arah peluru.

- Fungsi kelayakan : Tidak ada.
- Fungsi objektif : Memaksimalkan skor dengan selalu deteksi musuh dan terus bergerak melingkar untuk menghindari peluru musuh serta memanfaatkan kelemahan musuh dengan meningkatkan kekuatan tembakan saat musuh memiliki energi rendah.

3.3 Dasar Pemilihan Solusi *Greedy* Utama

Solusi *greedy* utama yang pada akhirnya kami pilih adalah solusi *greedy* pada bot CircularMG. Terdapat dua poin yang menjadi dasar alasan pemilihan bot ini sebagai bot utama:

- Seperti yang dilihat pada hasil uji, ketika diadu dengan banyak bot (13 bot), dibandingkan dengan ketiga bot lainnya, total *ranking* bot CircularMG lebih sedikit dibandingkan dengan lainnya (CircularMG: 20, Adudu: 21, Straight: 25). Walaupun memang ketika diadu berempat saja bot Straight selalu menang seperti yang dapat dilihat pada hasil uji, kami memilih untuk melihat hasil ketika diuji dengan banyak bot karena tidak diberitahukan bot ini akan diadu dengan berapa banyak bot lainnya dan bahkan ketika diadu berempat saja, CircularMG juga masih menempati pada posisi kedua pada sebagian besar pengujian. Singkatnya, kami memilih CircularMG karena hasilnya lebih stabil pada berbagai kondisi permainan dibandingkan kandidat bot lainnya.
- Menurut hasil pengamatan kami, bot yang selalu bergerak dan selalu menembak lebih optimal dibandingkan dengan yang tidak. Hal ini dapat dilihat pada implementasi bot Adudu, Straight, dan CircularMG. Namun, yang membedakan CircularMG dengan kedua bot tersebut adalah bot CircularMG tidak terpaku dengan suatu pola gerak/jalur, tidak seperti Adudu dan Straight. Ketika bot yang memaksakan diri untuk mengikuti suatu jalur terjebak dengan bot yang diam atau bot lain yang juga terpaku dengan suatu jalur, kedua bot akan terus saling menabrakan hingga salah satunya mati. Hal ini cukup dapat menjadi masalah ketika terdapat banyak sekali bot dalam satu arena sekaligus. CircularMG menangani masalah itu dengan mencari jalur melingkar baru yang tidak menabrak dengan bot lain. Hal ini mengurangi kemungkinan masa hidup bot yang singkat dan menambah kemungkinan bot untuk mencetak poin.

Bab 4
Implementasi dan Pengujian

4.1 Implementasi Masing-Masing Bot dengan Strategi *Greedy*

4.1.1 Bot Adudu

```
Function Start Main:
    Buat instance of Kotak
    Start the bot

// Konstruktor untuk menginisialisasi bot
Constructor Kotak:
    Load configuration file "Kotak.json"

// Fungsi yang terus menerus dipanggil
Function Run:
    // Set warna bot
    Set body color to Soft Light Green
    Set turret color to Light Olive Green
    Set radar color to Pastel Green
    Set bullet color to Moss Green
    Set scan color to Pale Green
    Set tracks color to Mint Green
    Set gun color to Fern Green

    // Bot mencari posisi di tengah arena
    FindTargetPosition()

    // Set kecepatan bot menjadi 6
    TargetSpeed <- 6

    While (the bot is running)do
        // Jika bot berada di pinggir arena, berusaha kembali ke tengah
        If the bot is near the arena boundary Then
            FindTargetPosition()

        // Set gerakan pistol dan badan bot agar bergerak di jalur kotak
        GunTurnRate <- 15
        Forward(200)
        TurnLeft(90)

// Prosedur untuk bergerak ke tengah arena
Procedure FindTargetPosition:
    // Target posisi adalah tengah area
    targetX <- ArenaWidth/2
    targetY <- ArenaHeight/2

    // Mengarahkan badan bot ke target
    angleToTarget <- Math.Atan2(targetY - currentY, targetX - currentX) * 180 /
Math.PI;
    turnAngle <- findAngle(angleToTarget)
    TurnLeft(turnAngle)

    // Bergerak ke target dan reset arah badan bot
    distanceToTarget <- Math.Sqrt((targetX - currentX) * (targetX - currentX) +
(targetY - currentY) * (targetY - currentY));
    Forward(distanceToTarget)
    TurnLeft(360-Direction)
```

```
// Fungsi untuk mencari arah target posisi
Function FindAngle(targetAngle):
    turnAngle <- targetAngle - currentDirection
    While turnAngle > 180 do
        turnAngle -= 360
    While turnAngle < -180 do
        turnAngle += 360

    Return turnAngle

// Prosedur dijalankan jika bot memindai bot lain
Procedure OnScannedBot:
    If (checkEnergy) Then
        distance <- distance to scanned bot
        If the bot is far away (distance > 200 or energy < 30) Then
            Fire(1) // Tembak dengan kekuatan 1
        Else if the bot is at medium range (distance between 100 and 200) Then
            Fire(2)
        Else if the bot is very close (distance < 50) Then
            Fire(3)

// Prosedur dijalankan jika bot menabrak tembok
Procedure OnHitWall:
    Back(100)
    FindTargetPosition()

// Prosedur dijalankan jika bot menabrak bot lain
Proced
ure OnHitBot:
    Back(30)

// Prosedur dijalankan jika bot terkena peluru
Procedure OnHitByBullet:
    Back(30)

// Fungsi mengecek apakah bot punya energi > 10
Function checkEnergy:
    Return true if energy > 10
```

Dalam implementasinya, pergerakan bot Adudu cukup efektif karena selalu memilih untuk bergerak dan dapat beradaptasi seperti bergerak ke tengah arena saat menabrak tembok dan mundur saat terkena tembakan musuh. Namun, kelemahan dari bot ini adalah pola gerakan yang mudah diprediksi dan tidak adanya algoritma penargetan tembakan sehingga tembakannya terkadang kurang akurat.

Karena bot Adudu terus menerus menembak (menembak setiap bot yang terpindai), bot ini efisien dalam menghasilkan poin dari *bullet damage*. Bot ini juga mengimplementasikan strategi bahwa semakin dekat bot musuh, semakin besar juga kekuatan pelurunya sehingga lebih efektif dalam menyerang musuh.

Dari segi manajemen sumber daya, bot Adudu memiliki mekanisme konservasi energi sederhana dengan tidak menembak saat energi di bawah 10, tetapi tidak memiliki sistem pengelolaan energi yang lebih kompleks. Secara keseluruhan, sebagai implementasi algoritma *greedy*, bot ini hanya membuat keputusan terbaik di waktu tertentu tanpa strategi jangka panjang yang adaptif sehingga dia akan efektif dalam melawan musuh dengan gerakan sederhana tetapi kemungkinan akan kesulitan menghadapi bot yang memiliki kemampuan prediksi dan penghindaran yang lebih baik.

4.1.2 Bot Straight

```
Program Straight:

Class Straight Extends Bot

    Constant:
        Integer SafeDistance <- 25 {Margin dari tembok agar tidak nabrak}

    Variabel:
        Double TargetX, TargetY {Titik awal tujuan bot (X=25, Y=Tengah)}
        Double minDistance

    Function Main():
        Create new instance of Straight
        Start bot

    Function Constructor():
        Load bot settings from "Straight.json"

    Function Run():
        Set GunTurnRate <- 20
        Set TargetSpeed <- 5

        {menentukan mau ke kiri tengah atau kanan tengah}
        DecideFirstPos()

        {Pastikan bot berada di posisi awal (tengah kiri)}
        While Not IsOnTargetCoord() do
            MoveToMiddle()

        While IsRunning do

            If MovingForward Then
                If X <= ArenaWidth - SafeDistance Then
                    {Maju hingga X = ArenaWidth - SAFE_DISTANCE}
                    Forward (ArenaWidth - (SafeDistance * 2))
                Else
                    MovingForward <- False  {Ubah ke mode mundur}
            Else
                If X > SafeDistance Then
                    {Mundur hingga X = SAFE_DISTANCE}
                    Move Back (ArenaWidth - (SafeDistance * 2))
                Else
                    {Ubah kembali jadi maju}
                    Set MovingForward <- True  {Ubah ke mode maju}

    {Fungsi untuk menentukan koordinat awal mau ke tengah kiri apa tengah kanan}
    Function DecideFirstPos():
        targetY <- ArenaHeight / 2

        If X <= (ArenaWidth / 2) Then
            targetX <- SAFE_DISTANCE
            movingForward <- True  {Menandakan bot bergerak maju dulu}
        Else
            targetX <- ArenaWidth - SAFE_DISTANCE
            movingForward <- False  {Menandakan bot bergerak mundur dulu}


    {Fungsi untuk gerak ke koordinat awal saat mulai awal round}
    Function MoveToMiddle():
```

```
        AngleToTarget <- Atan2(TargetY - Y, TargetX - X) * 180 / Pi
        TurnTo(AngleToTarget)

        DistanceToTarget <- Sqrt((TargetX - X)^2 + (TargetY - Y)^2)
        Forward(DistanceToTarget)

        TurnTo(0)  {Hadap kanan setelah sampai}

    {Fungsi untuk belok ke arah tertentu}
    Function TurnTo(Angle):
        TurnAngle <- Angle - Direction
        If TurnAngle > 180 Then
            Set TurnAngle <- TurnAngle - 360
        Else If TurnAngle < -180 Then
            Set TurnAngle <- TurnAngle + 360

        TurnLeft(TurnAngle)

    {Fungsi untuk cek apakah sudah di koordinat awal yang diinginkan}
    Function IsOnTargetCoord(:
        -> Abs(X - TargetX) < SafeDistance And Abs(Y - TargetY) < SafeDistance

    {Fungsi event-based yang akan menjalankan DorDor() ketika memindai bot}
    Function OnScannedBot(Event):
        Distance <- DistanceTo(Event.X, Event.Y)
        DorDor(Distance)

    {Fungsi untuk menembak sekaligus  mengatur kekuatan tembakan sesuai jarak bot
     target}
    Function DorDor(Distance)
        If Distance > 200 Or Energy < 15 Then
            Fire(1)
        Else If Distance > 50 Then
            Fire(2)
        Else
            Fire(3)

    {Fungsi untuk melakukan gerakan ketika menabrak bot lain}
    Function OnHitBot(Event botHitBotEvent):
        Move Back(60)
        waitTicks <- 30


    {Fungsi untuk melakukan gerakan yang perlu dianalisis per tick}
    Function OnTick(Event tickEvent):
    If waitTicks > 0 Then
        waitTicks <- waitTicks - 1
        Print(waitTicks)
        Return  {Jangan lakukan apa-apa selama masa tunggu}

    If waitTicks = 0 Then
        Forward(60)  {Kembali ke rute awal setelah menunggu}
        waitTicks <- -1

    If Energy <= 5 Then
        Set GunTurnRate <- 0  {Hentikan tembakan jika energi terlalu rendah}
```

Bila ditinjau dari segi menembak, bot ini efektif dan efisien karena meriam dan pemindai terus berputar sejauh 360 derajat, menyapu semua daerah di arena. Bot juga langsung menembak ketika menemukan bot yang terpindai. Hal ini menghasilkan bot yang cenderung memiliki *damage point* yang

mengisi persentase terbesar dibandingkan poin lainnya (menjadi penyumbang poin terbesar). Selain itu, bot ini juga memiliki mekanisme untuk mengatur besar tembakan sesuai jarak. Hal ini meningkatkan tingkat efisien mekanisme menembak bot ini.

Selanjutnya, bila ditinjau dari segi mengelak tembakan, bot ini juga cukup efektif dan efisien karena bot ini akan selalu bergerak secara horizontal di tengah arena. Karena selalu bergerak, sulit untuk bot lain yang memiliki strategi menembak dengan spontan untuk menembakkan bot ini secara akurat.

Selanjutnya, bila ditinjau dari segi mekanisme *ramming*, karena bot ini akan memaksakan diri untuk bergerak dalam suatu *path* tertentu, ketika ada bot lain berada di jalur tersebut maka bot ini akan terus menerus menabrak bot tersebut. Jadi saya rasa cukup efektif dalam menabrak namun tidak efisien dalam mengatur apakah harus terus menabrak atau tidak. Karena ketika energi bot lebih sedikit dibandingkan bot lawan, maka bot sudah dipastikan akan mati jika terus menabrak bot tersebut. Namun memang cukup efektif, tidak jarang mekanisme *ramming* ini menyumbang poin.

Selanjutnya, dari segi manajemen energi, bot ini tidak terlalu efisien dan efektif. Bot ini akan selalu memprioritaskan untuk menembak setiap bot yang terpindai. Hanya saja, ketika sisa energi kurang atau sama dengan lima, bot akan berhenti memindai dan menembak dan hanya fokus bergerak saja dengan harapan dapat menghindar dengan terus bergerak.

4.1.3 Bot Smart_Calender1874

```
Program Straight:

Class Smart_Calendar Extends Bot

    Constant:
        Integer SafeDistance <- 25  {Jarak aman dari tembok}

    Variabel:
        Double TargetX, TargetY  {Titik tujuan bot}
        Double minDistance  {Jarak minimum ke dinding terdekat}

    Function Main():
        Create new instance of Smart_Calendar1874
        Start bot

    Function Constructor():
        Load bot settings from "Smart_Calendar1874.json"

    Function Run():

        {Menentukan dinding terdekat untuk posisi awal}
        FindNearestWall()

        {Bergerak menuju dinding terdekat}
        While Not OnTargetWallCoord() do
            MoveToNearestWall()
        {Hadap ke arah tengah arena setelah sampai di dinding}
        TurnLeft(90)

        While IsRunning do
            TurnGunRight(360)  {Putar turret untuk scan musuh}

    {Fungsi untuk menentukan koordinat awal}
    Function FindNearestWall():
        distanceToNorth <- ArenaHeight - Y
```

```
        distanceToSouth <- Y
        distanceToEast <- ArenaWidth - X
        distanceToWest <- X

        minDistance <- Minimum(distanceToNorth, distanceToSouth, distanceToEast,
        distanceToWest)

        If minDistance = distanceToNorth Then
            TargetX <- X
            TargetY <- ArenaHeight - SafeDistance
        Else If minDistance = distanceToSouth Then
            TargetX <- X
            TargetY <- SafeDistance
        Else If minDistance = distanceToEast Then
            TargetX <- ArenaWidth - SafeDistance
            TargetY <- Y
        Else
            TargetX <- SafeDistance
            TargetY <- Y

    {Fungsi untuk gerak ke koordinat awal saat mulai awal round}
    Function MoveToNearestWall():
        AngleToWall <- Atan2(TargetY - Y, TargetX - X) * 180 / Pi
        OppositeAngle <- (AngleToWall + 180) % 360  {Arah sebaliknya}

        TurnTo(OppositeAngle)
        Back(DistanceTo(TargetX, TargetY))

    {Fungsi untuk belok ke arah tertentu}
    Function TurnTo(Angle):
        TurnAngle <- Angle - Direction
        If TurnAngle > 180 Then
            TurnAngle <- TurnAngle - 360
        Else If TurnAngle < -180 Then
            TurnAngle <- TurnAngle + 360

        TurnLeft(TurnAngle)

    {Fungsi untuk cek apakah sudah di koordinat awal yang diinginkan}
    Function OnTargetWallCoord():
        -> Abs(X - TargetX) < SafeDistance / 2 And Abs(Y - TargetY) < SafeDistance /
        2

    {Fungsi event-based yang akan menjalankan DorDor() ketika memindai bot}
    Function OnScannedBot(Event e):
        Distance <- DistanceTo(e.X, e.Y)
        DorDor(Distance)

    {Fungsi untuk menembak sekaligus  mengatur kekuatan tembakan sesuai jarak bot
     target}
    Function DorDor(Distance):
        If Distance > 200 Or Energy < 15 Then
            Fire(1)
        Else If Distance > 50 Then
            Fire(2)
        Else
            Fire(3)

    {Fungsi untuk bergerak ketika bot terkena peluru}
    Function MoveWhenHit():
        If Direction = 180 Then
            minDistance <- X - SafeDistance
```

```
        Else If Direction = 270 Then
            minDistance <- Y - SafeDistance
        Else If Direction = 0 Then
            minDistance <- ArenaWidth - SafeDistance - X
        Else If Direction = 90 Then
            minDistance <- ArenaHeight - SafeDistance - Y

        If minDistance < 100 + SafeDistance Then
            moveDistance <- minDistance - SafeDistance
        Else
            moveDistance <- 100

        If minDistance < SafeDistance + 5 Then
            TurnRight(90)
            Forward(110)

        Forward(Max(moveDistance, 0))

    {Fungsi event-based yang akan menjalankan movewhenhit() ketika tertembak peluru
    bot lain}
    Function OnHitByBullet(Event e):
        MoveWhenHit()
```

Bila ditinjau dari segi menembak, bot ini tidak terlalu efektif dan efisien karena meriam dan pemindai terus berputar sejauh 360 derajat walaupun sebenarnya cukup 180 derajat saja.Selain itu, ketika bot tertembak dan berpindah tempat, bot akan berhenti memindai dan menembak sejenak dan baru melanjutkan kembali setelah berhenti bergerak. Oleh karena itu, bot ini kurang efisien dan efektif dalam konteks menembak.

Selanjutnya, bila ditinjau dari segi mengelak tembakan, bot ini juga kurang efektif dan efisien karena bot ini pada dasarnya tidak bergerak, diam saja, kecuali setelah tertembak bot lain. Hal ini menyebabkan bot ini menjadi target sasaran empuk bot lawan dan mudah ditembak pada sebagian besar waktu.

Selanjutnya, bila ditinjau dari segi mekanisme *ramming*, karena bot ini akan memaksakan diri untuk terus diam di tempatnya walaupun sedang tertabrak dengan bot lain, bot ini cukup efektif namun kurang efisien. Bot ini tidak dapat mempertimbangkan apakah energi yang dimiliki lebih besar dari energi lawan, sehingga ketika bot memaksakan diri untuk terus menabrak bot lain, ada kemungkinan untuk mati lebih awal dibandingkan bot lawan.

Selanjutnya, dari segi manajemen energi, bot ini tidak terlalu efisien dan efektif. Bot ini tidak memiliki mekanisme yang mengatur pergerakannya berdasarkan energi yang tersisa. Oleh karena itu, bot ini kurang efektif dan kurang efisien dalam manajemen energi.

4.1.4 Bot CircularMG

```
Program CircularMG:

Class CircularMG Extends Bot

    {Prosedur main untuk menjalankan bot}
    Procedure Start Main:
        Create an instance of CircularMG
        Start the bot
```

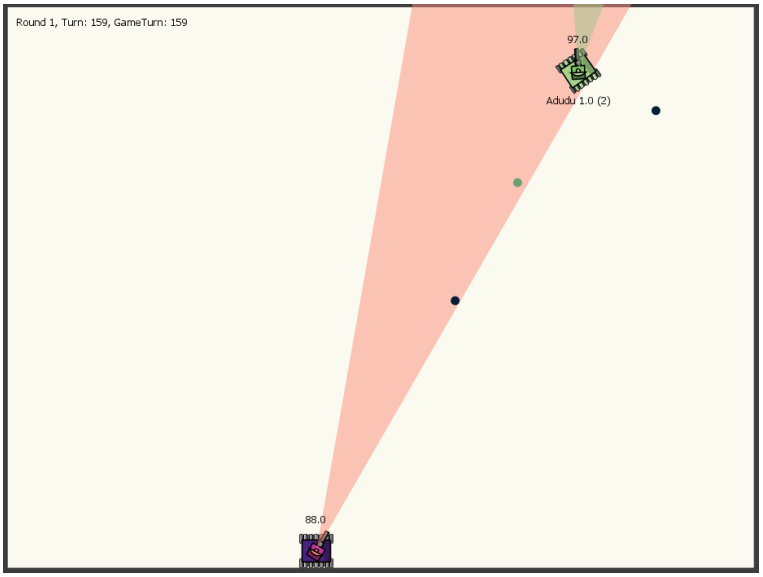
```
{Konstruktor yang memuat file konfigurasi bot}
Constructor CircularMG:
    Load configuration file "CircularMG.json"

{Melakukan inisialisasi pergerakan bot ketika permainan dimulai}
Procedure Run:
    {Set warna bot}
    Set body color to Light Blue
    Set turret color to Black
    Set radar color to Light Blue
    Set bullet color to Blue
    Set scan color to Light Purple
    Set tracks color to Purple
    Set gun color to Dark Purple

    While IsRunning do
        {Bot akan terus bergerak dalam bentuk lingkaran untuk mendeteksi musuh
        dan agar susah untuk ditarget oleh musuh}
        TurnGunLeft(360)
        TargetSpeed <- 4
        TurnRate <- 5

{Prosedur dijalankan jika bot memindai bot lain}
Procedure OnScannedBot:
    Double firePower
    {Menentukan kekuatan tembakan sesuai energi dari musuh/bot lain}
    If bot lain memiliki lebih dari 50 energi (e.Energy > 50) Then
        firepower <- 1
    Else {Jika bot lain memiliki energi yang lemah yaitu <= 50 energy}
        Firepower <- 3 {tingkat tembakan maksimum}
    Fire(firepower) {Bot melakukan tembakan}

{Prosedur dijalankan jika bot terkena bot lain}
Procedure OnHitBot:
    TargetSpeed <- 4
    Back(100)
    TurnRight(45)
    Forward(100)

{Prosedur dijalankan jika bot menabrak tembok}
Procedure OnHitWall:
    Back(50)
    TurnRight(90)
    i traversal [0..4] {Gerak melingkar untuk membingungkan musuh}
        TargetSpeed <- 5
        TurnRate <- 20

{Prosedur dijalankan jika bot terkena peluru}
Procedure OnHitByBullet:
    {Menyimpan informasi arah peluru untuk menentukan langkah selanjutnya untuk
    menghindar dari peluru musuh}
    bulletDirection <- e.Bullet.Direction
    {Jika arah peluru datang dari depan bot, maka bergerak ke kanan}
    If (bulletDirection > -90 && bulletDirection < 90) Then
        TurnRight(90)
        Forward(150)
    Else {Jika peluru datang dari belakang bot, maka bergerak ke kiri}
        TurnLeft(90)
        Forward(150)
```

Berdasarkan implementasi bot CircularMG di atas, pergerakan bot tersebut cukup efektif karena selalu bergerak dalam pola lingkaran saat bot masih aktif atau masih memiliki energi agar sulit ditembak oleh musuh dan juga memberikan keuntungan dalam menghindari peluru. Lalu, bot selama melakukan pergerakan ini akan terus mendeteksi musuh di sekitarnya dan jika musuh terdeteksi akan melakukan penembakan dengan kekuatan terendah tetapi juga memanfaatkan kondisi musuh ketika memiliki energi yang lemah maka kekuatan tembakan menjadi maksimum.

Kemudian, bot ini akan menghindari arah datangnya peluru dengan bergerak ke kanan atau ke kiri, tetapi pergerakan ini hanya aktif ketika bot sudah terkena tembakan dari musuh. Lalu, bot ini segera mundur dan berbelok jika terkena musuh serta mundur dan kembali ke pola lingkaran apabila terkena tembok. Oleh karena itu, dengan melihat strategi *greedy* bot CircularMG ini cukup efektif dalam bertahan hidup dan mendapatkan skor di arena, tetapi bot ini akan kesulitan jika terdapat musuh yang memiliki strategi yang dapat memprediksi pergerakan bot, bot ini juga tidak memiliki manajemen energi sendiri sehingga jika tembakannya selalu meleset atau tidak sering terkena musuh maka energi bot ini akan cepat berkurang dan tidak mendapatkan skor yang baik dalam arena.

4.2 Pengujian Semua Bot

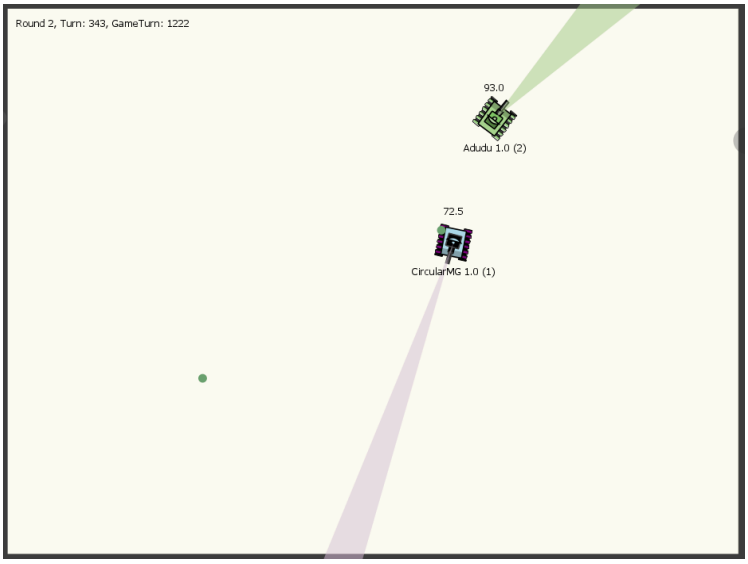
4.2.1 Adudu vs Smart_Calendar1874



Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Adudu 1.0	811	200	40	494	73	4	0	5	1	0
2	Smart_Calendar1874 ...	348	50	10	286	0	2	0	2	4	0

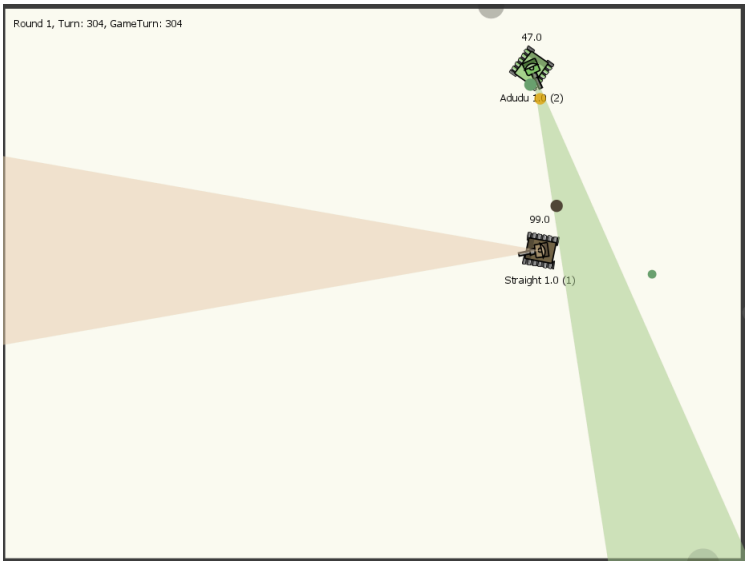
4.2.2 Adudu vs CircularMG



Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	CircularMG 1.0	484	150	30	268	29	6	0	3	2	0
2	Adudu 1.0	343	50	10	250	18	14	0	2	3	0

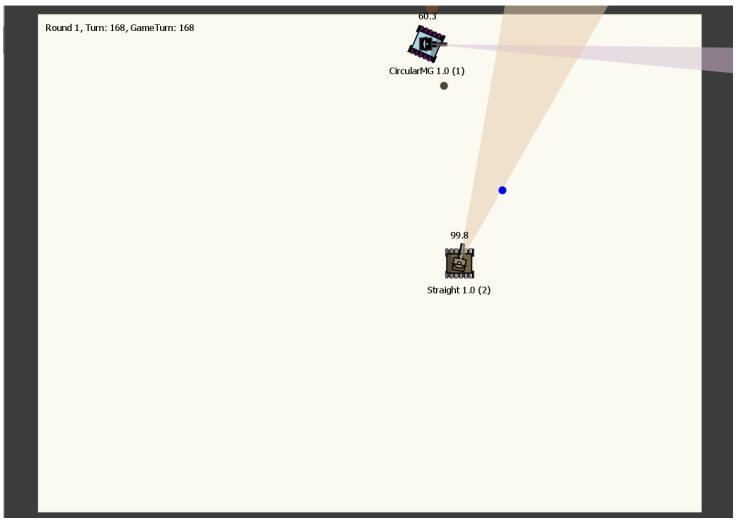
4.2.3 Adudu vs Straight



Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Straight 1.0	1817	500	100	962	164	90	0	10	0	0
2	Adudu 1.0	342	0	0	272	0	70	0	0	10	0

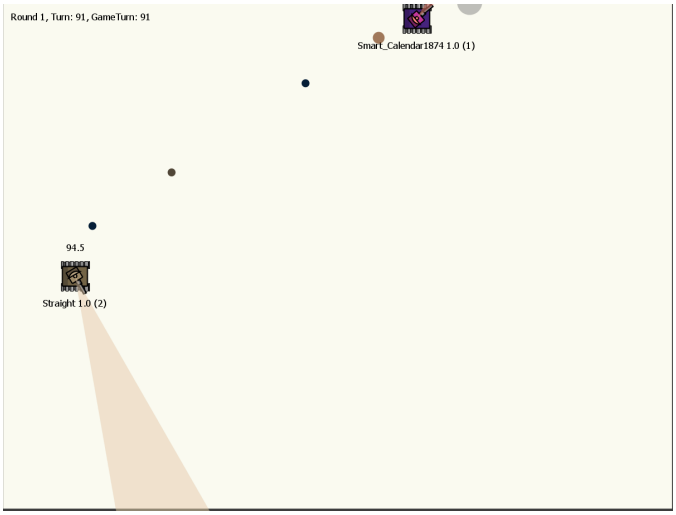
4.2.4 Straight vs CircularMG



Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Straight 1.0	947	300	60	444	93	49	0	6	0	0
2	CircularMG 1.0	100	0	0	88	0	12	0	0	6	0

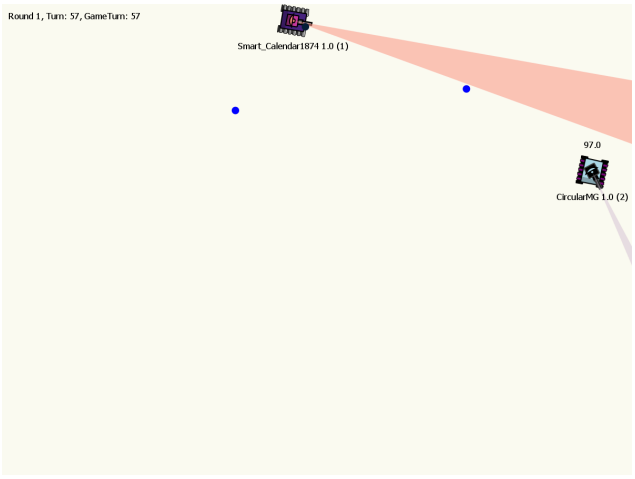
4.2.5 Straight vs Smart_Calendar1874



Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Straight 1.0	1270	350	70	678	114	24	34	8	0	0
2	Smart_Calendar1874 ...	276	0	0	270	0	6	0	0	8	0

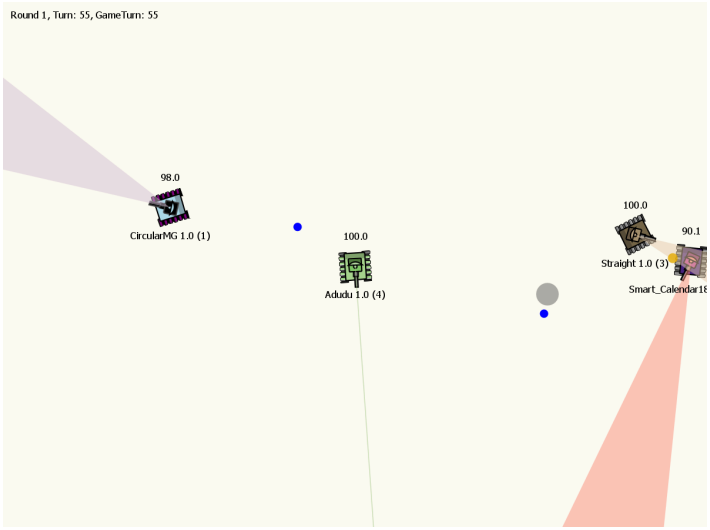
4.2.6 Smart_Calendar1874 vs CircularMG



Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	CircularMG 1.0	1074	300	60	596	112	6	0	7	1	0
2	Smart_Calendar1874 ...	370	50	10	286	20	4	0	1	7	0

4.2.7 Hasil Uji Keempat Bot Secara Bersamaan



Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Straight 1.0	1434	450	90	730	82	82	0	4	0	0
2	CircularMG 1.0	860	400	30	392	30	7	0	1	3	1
3	Smart_Calendar1874 ...	450	250	0	198	1	0	0	0	1	2
4	Adudu 1.0	409	100	0	230	24	55	0	0	1	2

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Straight 1.0	1611	600	120	756	95	40	0	5	0	0
2	CircularMG 1.0	659	300	0	312	29	18	0	0	4	0
3	Smart_Calendar1874 ...	511	250	0	260	0	1	0	0	1	4
4	Adudu 1.0	289	50	0	226	0	13	0	0	2	0

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Straight 1.0	1552	650	120	662	78	42	0	4	1	0
2	CircularMG 1.0	641	300	0	332	4	5	0	0	1	3
3	Smart_Calendar1874 ...	622	250	0	342	23	7	0	1	2	0
4	Adudu 1.0	602	150	0	336	19	97	0	0	1	2

4.2.8 Hasil Uji Keempat Bot dengan Sembilan Bot Template Lainnya

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Spin Bot 1.0	2244	1250	120	752	69	53	0	2	0	0
2	Track Fire 1.0	1953	1300	0	612	40	0	0	0	1	1
3	Walls 1.0	1801	1250	0	490	34	23	3	0	0	1
4	Velocity Bot 1.0	1663	1150	0	304	25	161	22	0	0	1
5	Crazy 1.0	1583	1200	120	224	3	36	0	1	0	0
6	CircularMG 1.0	1579	1050	0	436	52	41	0	0	2	0
7	Dunggulge Dunggulg...	1076	700	0	346	14	16	0	0	0	0
8	Adudu 1.0	999	500	0	266	0	209	24	0	0	0
9	Straight 1.0	987	500	0	252	0	235	0	0	0	0
10	Corners 1.0	846	600	0	242	0	4	0	0	0	0
11	Target 1.0	704	700	0	0	0	4	0	0	0	0
12	Painting Bot 1.0	550	550	0	0	0	0	0	0	0	0
13	Template Bot 1.0	335	300	0	28	0	7	0	0	0	0

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Walls 1.0	2644	1600	240	730	48	26	0	2	1	0
2	Velocity Bot 1.0	2303	1700	120	424	17	41	0	1	0	1
3	Straight 1.0	2027	1300	0	658	19	50	0	0	1	0
4	Adudu 1.0	1852	1400	0	422	10	19	0	0	0	1
5	CircularMG 1.0	1826	1350	0	432	26	17	0	0	0	1
6	Spin Bot 1.0	1767	1050	0	624	16	76	0	0	1	1
7	Track Fire 1.0	1485	950	0	495	39	0	0	0	1	0
8	Crazy 1.0	1466	1200	0	224	6	36	0	0	0	0
9	Dunggulge Dunggulg...	1113	650	0	434	21	7	0	1	0	0
10	Template Bot 1.0	939	750	0	52	0	112	26	0	0	0
11	Target 1.0	902	900	0	0	0	2	0	0	0	0
12	Painting Bot 1.0	400	400	0	0	0	0	0	0	0	0
13	Corners 1.0	282	100	0	180	0	2	0	0	0	0

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Spin Bot 1.0	2661	1600	240	720	61	40	0	2	0	0
2	Track Fire 1.0	2011	1050	0	886	74	0	0	1	1	0
3	Adudu 1.0	1974	1450	0	476	32	16	0	0	0	2
4	Walls 1.0	1925	1350	0	520	24	31	0	0	1	0
5	Velocity Bot 1.0	1455	900	0	304	14	236	0	0	1	0
6	CircularMG 1.0	1285	950	0	256	30	49	0	0	0	0
7	Template Bot 1.0	1205	1200	0	4	0	1	0	0	0	0
8	Dunggulge Dunggulg...	1132	800	0	324	5	2	0	0	0	1
9	Straight 1.0	1088	650	0	298	0	124	16	0	0	0
10	Crazy 1.0	600	450	0	124	0	26	0	0	0	0
11	Corners 1.0	538	400	0	134	0	4	0	0	0	0
12	Target 1.0	452	450	0	0	0	2	0	0	0	0
13	Painting Bot 1.0	400	400	0	0	0	0	0	0	0	0

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Spin Bot 1.0	2244	1250	120	752	69	53	0	2	0	0
2	Track Fire 1.0	1953	1300	0	612	40	0	0	0	1	1
3	Walls 1.0	1801	1250	0	490	34	23	3	0	0	1
4	Velocity Bot 1.0	1663	1150	0	304	25	161	22	0	0	1
5	Crazy 1.0	1583	1200	120	224	3	36	0	1	0	0
6	CircularMG 1.0	1579	1050	0	436	52	41	0	0	2	0
7	Dunggulge Dunggulg...	1076	700	0	346	14	16	0	0	0	0
8	Adudu 1.0	999	500	0	266	0	209	24	0	0	0
9	Straight 1.0	987	500	0	252	0	235	0	0	0	0
10	Corners 1.0	846	600	0	242	0	4	0	0	0	0
11	Target 1.0	704	700	0	0	0	4	0	0	0	0
12	Painting Bot 1.0	550	550	0	0	0	0	0	0	0	0
13	Template Bot 1.0	335	300	0	28	0	7	0	0	0	0

4.3 Analisis Hasil Uji

4.3.1 Adudu

Strategi *greedy* pada bot seringkali mendapatkan skor yang cukup optimal pada pertandingan dengan bot-bot lain. Bot dapat menembak bot lain dengan cukup akurat dan mendapatkan poin dari tembakan yang berhasil. Di sisi lain, bot kurang optimal dari segi menghindari serangan musuh karena gerakannya yang mudah diprediksi. Selain itu, tabrakan dengan bot lain akan mengganggu rute dan membuat Adudu kehilangan kestabilannya.

4.3.2 Straight

Strategi *greedy* pada bot ini akan hampir selalu mendapatkan nilai optimal dan meraih urutan pertama selama arena tidak terlalu penuh dengan bot lain. Jika dalam satu arena terdapat 2-8 bot, kemungkinan besar bot ini dapat memberikan hasil yang paling optimal. Namun, ketika jumlah bot di dalam satu arena sudah berjumlah 10 atau lebih, kemungkinan bot ini akan meraih nilai optimal akan semakin kecil karena bot akan cukup kesulitan ketika ada banyak bot yang menghalangi jalur geraknya. Hal ini dapat dilihat pada bagian hasil uji dan membandingkan hasil uji 1v1, 1v3, dan 1v13. Pada 1v1 dan 1v3 bot ini kemungkinan besar dapat memberikan hasil yang optimal, namun pada uji 1v13, bot ini gagal memberikan hasil yang optimal.

4.3.3 Smart_Calendar1874

Strategi *greedy* pada bot ini dapat cukup meraih nilai optimal dalam konteks menembak karena seringkali bot ini akan diam sehingga tembakan lebih stabil. Namun hal ini juga tidak dapat tercapai jika bot lawan memiliki gerakan yang cukup ekstrem. Lalu, bot ini sangat tidak optimal dari segi mengelak dari peluru karena bot hanya akan bergerak ketika tertembak oleh bot lain dan akan diam jika tidak, karena bot relatif lebih sering diam, bot ini dapat menjadi sasaran empuk untuk bot lawan.

4.3.4 CircularMG

Strategi *greedy* pada bot ini dapat memaksimalkan nilai secara optimal apabila jumlah bot dalam arena tersebut sedikit tetapi jika jumlah bot dalam suatu arena mencapai 10 atau lebih bot maka berdasarkan hasil uji di atas, bot CircularMG ini mendapatkan peringkat 5-6 sehingga nilai yang didapatkan kurang optimal. Hal ini dapat disebabkan dari beberapa faktor dalam arena seperti kurangnya keakuratan dalam menembak karena selalu bergerak dalam pola lingkaran, jika bot ini terkena bagian salah satu sudut tembok maka terdapat masalah dalam keluar dari situasi ini sehingga musuh dapat memanfaatkan situasi ini dan menembak bot CircularMG, serta karena bot ini dalam pola lingkaran terus menerus dapat menjadi kelemahan ketika bertemu dengan musuh yang dapat memprediksi pergerakan bot ini.

4.4 Rincian Implementasi Bot Utama

Bot utama yang dipilih yaitu bot CircularMG berdasarkan hasil uji dari keempat bot dengan alternatif solusi *greedy* yang berbeda. Berikut adalah analisis berdasarkan struktur data, fungsi, dan prosedur yang digunakan pada bot dengan solusi *greedy* yang dipilih.Bot ini tidak menggunakan struktur data kompleks seperti array atau tree, melainkan hanya menggunakan beberapa variabel untuk menyimpan informasi yang diperlukan bot dalam mengambil keputusan ketika permainan sedang berlangsung di arena seperti,

- TargetSpeed : menyimpan informasi kecepatan bot untuk mengontrol pergerakan bot
- TurnRate : menyimpan sudut pergerakan putaran bot
- firePower : menyimpan kekuatan tembakan bot terhadap musuh
- bulletDirection : menyimpan arah peluru musuh untuk menentukan strategi menghindar dari peluru musuh
- e.Energy : menyimpan energi musuh untuk menentukan strategi tembakan

Bot ini memiliki beberapa prosedur yang digunakan untuk menentukan strategi *greedy* yaitu,

- Prosedur Run() berguna untuk inisialisasi pergerakan pertama bot serta bagaimana bot bergerak terus menerus apabila energi bot masih ada yaitu bergerak dalam pola lingkaran dengan kecepatan tetap (TargetSpeed = 4).
- Prosedur OnScannedBot() untuk menentukan kekuatan tembakan berdasarkan energi musuh. Jika energi musuh di atas 50 maka bot akan menghemat energi dengan menggunakan kekuatan tembakan ringan yaitu 1, sedangkan jika energi musuh di bawah atau sama dengan 50 maka bot segera menembak dengan kekuatan penuh yaitu 3.
- Prosedur OnHitByBullet() yang berfungsi menghindari tembakan musuh jika terkena peluru di mana peluru yang datang dari arah depan akan menyebabkan bot berbelok ke kanan sejauh 90 derajat dan maju, sedangkan peluru yang datang dari arah belakang akan menyebabkan bot berbelok ke kiri sejauh 90 derajat dan maju.

- Prosedur OnHitBot() untuk menghindari tabrakan dari bot lain dengan cara bot langsung mundur, berbelok ke kanan sejauh 45 derajat, dan maju.
- Prosedur OnHItWall() berfungsi untuk menghindari tabrakan dengan tembok dengan cara bot mundur, berbelok ke kanan sejauh 90 derajat, dan bergerak dalam pola lingkaran untuk melanjutkan gerak melingkar sebelumnya serta membingungkan musuh.

Bab 5
Kesimpulan dan Saran

5.1 Kesimpulan

Algoritma *greedy* dapat diterapkan untuk menentukan strategi bot dalam permainan Robocode. Fungsi dan himpunan yang merupakan komponen dari algoritma *greedy* dapat dipetakan dengan baik ke algoritma bot Robocode. Bot dengan algoritma *greedy* dapat dirancang untuk merespons kondisi-kondisi khusus dalam permainan, seperti terkena tembakan, menabrak dinding, atau menemukan musuh dengan energi rendah, dengan pemilihan aksi optimal untuk setiap kondisi tersebut.

Heuristik yang digunakan dalam penentuan algoritma juga dapat diadaptasi sehingga setiap bot dapat memiliki pendekatan yang unik dalam optimasi pergerakan dan penembakan. Setiap heuristik *greedy* memungkinkan bot untuk mengembangkan keunggulan dalam aspek tertentu dari permainan. Misalnya, bot CircularMG menggunakan pola pergerakan melingkar untuk menghindari serangan dan deteksi musuh sementara bot Straight mengoptimalkan efisiensi penembakan dengan memaksimalkan area pemindaian.

Akan tetapi, penggunaan algoritma *greedy* dalam pembuatan bot Robocode juga memiliki kekurangan. Algoritma *greedy* berfokus pada langkah terbaik yang dapat dilakukan pada waktu tertentu sehingga terkadang kurang optimal dalam perencanaan strategis jangka panjang. Contohnya, bot Adudu memiliki pola gerakan yang mungkin mudah diprediksi oleh lawan karena tidak mempertimbangkan konsekuensi jangka panjang dari pola pergerakannya.

5.2 Saran

Dalam pengembangan bot dalam permainan Robocode, terdapat beberapa aspek yang masih dapat dieksplorasi dan dapat menjadi fokus penelitian lebih lanjut. Salah satunya adalah mekanisme prediksi dan penghindaran peluru yang lebih canggih. Sebagian besar bot yang dibangun untuk permainan Robocode saat ini hanya menggunakan pola gerakan dasar seperti zig-zag atau maju-mundur untuk menghindari serangan lawan. Penggunaan algoritma yang lebih maju dan kompleks mungkin dapat membantu bot dalam memprediksi lintasan peluru lawan dan melakukan manuver optimal untuk menghindarinya.

Selain strategi penghindaran, eksplorasi lebih lanjut juga dapat dilakukan pada strategi pergerakan. Alih-alih hanya bergerak dalam pola tetap, bot dapat menerapkan strategi bergerak yang adaptif, mengubah pola gerakan secara dinamis berdasarkan analisis kondisi pertempuran. Selain itu, eksplorasi mekanik *ramming* juga menjadi potensi pengembangan yang menarik. Eksplorasi algoritma bot yang dapat menentukan kapan harus menabrak lawan untuk mendapatkan keuntungan maksimal tanpa mengorbankan terlalu banyak energi masih cukup kurang. Di sisi lain, strategi manajemen energi yang lebih canggih juga dapat dikembangkan agar bot dapat mempertimbangkan kapan harus menyerang, bertahan, atau menghemat energi untuk memperpanjang durasi pertarungan. Dengan mengeksplorasi aspek-aspek tersebut, bot Robocode dapat semakin optimal dalam menghadapi berbagai skenario pertempuran yang kompleks.

Lampiran

Tautan *Repository* GitHub
https://github.com/numshv/Tubes1_Dunggulge-Dunggulge

Tautan Video Youtube
https://youtu.be/TnOxrnCBm8s?si=Zylf2hntdDwcqHiv

Tabel Pengerjaan

No	Poin	Ya	Tidak
1	Bot dapat dijalankan pada Engine yang sudah dimodifikasi asisten.	✔	
2	Membuat 4 solusi *greedy* dengan heuristic yang berbeda.	✔	
3	Membuat laporan sesuai dengan spesifikasi.	✔	
4	Membuat video bonus dan diunggah pada Youtube.	✔	

Daftar Pustaka

Munir, R. (2025). *Algoritma Greedy (Bagian 1)*. Institut Teknologi Bandung. Diakses pada https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/04-Algoritma-Greedy-(2025)-Bag1.pdf

Lab IRK ITB. (2025). *Robocode APIs*. Diakses pada https://drive.google.com/file/d/12Woc3ni-x2GQud24WJRmACm3ZghoZIUP/view?pli=1

Robocode Developers. (2025). *Robocode Tank Royale*. Diakses pada https://robocode-dev.github.io/tank-royale/

Microsoft. (2025). *C# Documentation*. Microsoft Learn. Diakses pada https://learn.microsoft.com/en-us/dotnet/csharp/