LAPORAN TUGAS BESAR 1

IF2211 STRATEGI ALGORITMA

Pemanfaatan Algoritma Greedy dalam pembuatan bot permainan Robocode Tank Royale

Disusun oleh:

“Satu”

Nicholas Andhika Lucas	13523014
Bertha Soliany Frandi	13523026
Michael Alexander Angkawijaya	13523102

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

BAB I

DESKRIPSI TUGAS

1.1 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.

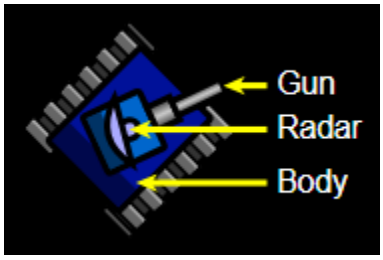
6. Tabrakan

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.

Untuk informasi lebih lengkap, silahkan buka dokumentasi Tank Royale pada link berikut.

1.2 Spesifikasi

1. Buatlah 4 bot (1 utama dan 3 alternatif) dalam bahasa C# (.net) yang mengimplementasikan algoritma *Greedy* pada bot permainan Robocode Tank Royale dengan tujuan memenangkan permainan.
2. Tugas dikerjakan berkelompok dengan anggota minimal 2 orang dan maksimal 3 orang, boleh lintas kelas dan lintas kampus.
3. Strategi *greedy* yang diimplementasikan setiap kelompok harus dikaitkan dengan fungsi objektif dari permainan ini, yaitu memperoleh skor setinggi mungkin pada akhir pertempuran. Hal ini dapat dilakukan dengan mengoptimalkan komponen skor yang telah dijelaskan diatas.

4. Strategi *greedy* yang diimplementasikan harus berbeda untuk setiap bot yang diimplementasikan dan setiap strategi *greedy* harus menggunakan heuristic yang berbeda.
5. Bot yang dibuat TIDAK BOLEH sama dengan SAMPEL yang diberikan sebagai CONTOH. Baik dari starter pack maupun dari repository engine asli.
6. Buatlah strategi *greedy* terbaik, karena setiap bot utama dari masing-masing kelompok akan diadu dalam kompetisi Tubes 1.
7. Strategi *greedy* yang kelompok anda buat harus dijelaskan dan ditulis secara eksplisit pada laporan, karena akan diperiksa saat demo apakah strategi yang dituliskan sesuai dengan yang diimplementasikan.
8. Setiap kelompok dapat menggunakan kreativitas yang bermacam macam dalam menyusun strategi *greedy* untuk memenangkan permainan. Implementasi pemain harus dapat dijalankan pada game engine yang telah disebutkan diatas serta dapat dikompetisikan dengan bot dari kelompok lain.
9. Program harus mengandung komentar yang jelas, dan untuk setiap strategi *greedy* yang disebutkan, harus dilengkapi dengan kode sumber yang dibuat. Artinya semua strategi harus diimplementasikan.
10. Mahasiswa disarankan membaca dokumentasi dari game engine. Perlu diperhatikan bahwa game engine yang digunakan untuk tubes ini SUDAH DIMODIFIKASI. Untuk Perubahan dapat dilihat pada bagian starter pack.

BAB II

LANDASAN TEORI

2.1 Dasar Teori Algoritma *Greedy* secara Umum

Algoritma *greedy* adalah metode yang paling populer dan sederhana untuk memecahkan persoalan optimasi. Persoalan optimasi yang dimaksud adalah persoalan mencari solusi optimal yang mempunyai dua macam, yaitu maksimasi dan minimasi. Prinsip dari algoritma *greedy* adalah mengambil semua yang bisa diambil dengan membentuk solusi langkah per langkah. Setiap langkah yang diambil merupakan keputusan terbaik pada saat itu dan berharap bahwa dengan memilih optimum lokal akan berakhir dengan optimum global. Hal ini karena pada algoritma *greedy* tidak dapat kembali ke langkah sebelumnya.

Algoritma *greedy* mempunyai enam elemen. Elemen tersebut adalah himpunan kandidat, himpunan solusi, fungsi solusi, fungsi seleksi, fungsi kelayakan, dan fungsi obyektif. Himpunan kandidat berisi kandidat yang akan dipilih pada setiap langkah. Himpunan solusi merupakan kandidat yang sudah dipilih. Fungsi solusi merupakan yang menentukan apakah himpunan kandidat yang dipilih memberikan solusi. Fungsi seleksi digunakan untuk memilih kandidat berdasarkan strategi *greedy* tertentu yang dimana bersifat heuristik. Fungsi kelayakan akan memeriksa kelayakan kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi. Sedangkan untuk fungsi obyektif digunakan untuk memaksimumkan atau meminimumkan solusi. Dengan elemen-elemen tersebut, dapat dikatakan bahwa algoritma *greedy* melibatkan pencarian sebuah himpunan bagian dari himpunan kandidat yang dalam hal ini himpunan bagian harus memenuhi beberapa kriteria yang ditentukan, yaitu himpunan bagian menyatakan suatu solusi dan dioptimasi oleh fungsi obyektif.

Perlu dijadikan catatan bahwa optimum global belum tentu merupakan solusi optimum. Optimum global dapat merupakan solusi *sub-optimum* atau *pseudo-optimum*. Alasan dibalik hal ini adalah algoritma *greedy* yang tidak beroperasi secara menyeluruh terhadap semua kemungkinan solusi yang ada seperti metode *exhaustive search*. Algoritma *greedy* juga mempunya fungsi seleksi yang berbeda sehingga kita harus memilih fungsi yang tepat jika menginginkan *greedy* untuk menghasilkan solusi optimal.

2.2 Cara Kerja Program secara Umum

Pada tugas besar ini, digunakan Robocode Tank Royale sebagai sarana penguji pemahaman mengenai algoritma *greedy*. Terdapat *starter pack* yang digunakan pada tugas besar ini. *Starter pack* ini berisi *sample bot* yang disediakan oleh Robocode dan *source code game engine* yang dapat digunakan untuk menjalankan bot. *Starter pack* tersedia pada tautan berikut https://github.com/Ariel-HS/tubes1-if2211-starter-pack.

Dengan menggunakan jar atau *source code*, *game engine* dapat dijalankan dengan menjalankan perintah berikut pada terminal `` `java -jar robocode-tankroyale-gui-0.30.0.jar` ``. Setelah GUI Robocode terlihat pada layar, konfigurasi booter dapat dilakukan dengan memilih *directory* yang berisi folder-folder bot. Konfigurasi ini dilakukan dengan memilih tombol "config" pada kiri atas GUI. Permainan dapat dimulai dengan tombol "Battle". Lakukan pemilihan kepada bot yang ingin dipertarungkan dengan melakukan boot pada bot tersebut. Selanjutnya adalah informasi mengenai bot dan cara mengimplementasikkan algoritma *greedy* ke dalam bot.

2.2.1 Cara Bot Melakukan Aksi

Suatu bot dapat dijalankan pada Robocode dengan beberapa syarat. Syarat paling pertama adalah mempunyai .NET dan C# sebagaimana implementasi algoritma nantinya menggunakan bahasa C#. Pengguna atau orang yang ingin menjalankan bot harus memastikan bahwa .NET yang dimiliki sesuai dengan versi .NET yang ada pada file .csproj bot tersebut. Folder bin dan obj mungkin perlu dihapus jika bot tidak bisa dijalankan. Penghapusan kedua folder ini tidak terbatas pada kondisi bot tidak bisa di boot melainkan berlaku pula pada kondisi ketika telah dilakukan pembaharuan kode bot dan ingin menjalankan bot versi terbaru tersebut. Setelah bot berhasil di boot dan ditambahkan pada permainan, ketika permainan dimulai, bot akan menjalankan kode pada file .cs dan berjalan sesuai kode tersebut.

2.2.2 Cara Implementasi Algoritma *Greedy* ke dalam Bot

Algoritma *greedy* dapat diimplementasikan pada file .cs yang dimiliki oleh sebuah bot. Satu bot akan mempunyai lima file yang memungkinkan bot untuk berjalan pada sistem Robocode, yaitu file .cmd, file .csproj, file .cs, file .json,

dan file .sh. Untuk membuat bot, harus terdapat folder yang berisi kelima file tersebut dan harus memiliki nama sesuai nama bot yang diinginkan. File .json dibuat terlebih dahulu yang bertujuan untuk memberikan informasi mengenai bot yang akan dibuat. Informasi yang wajib ada pada file .json berupa nama bot, versi bot, dan nama pembuat bot. File .csproj, .cmd, dan .sh adalah file pendukung dimana file .csproj berisi konfigurasi bot untuk mengelola *build* dan *dependency*, file .cmd untuk mengkompilasi dan/atau menjalankan bot tanpa perlu adanya perintah, dan file.sh digunakan untuk kompilasi Unix-based OS.

Pada file .cs, sebelum implementasi algoritma *greedy* terdapat kerangka minimal sebagai syarat suatu bot dapat dijalankan pada Robocode. Kerangka minimal ini berisi *main method* untuk menjalankan bot, konstruktor untuk *load* file konfigurasi (file .json), dan prosedur Run() yang berisi pengaturan warna bot dan kondisi pengulangan yang akan dijalankan ketika bot masih hidup di dalam permainan. Setelah kerangka minimal ini, algoritma *greedy* dapat diimplementasikan dengan menggunakan API Robocode serta menambahkan fungsi dan/atau prosedur sesuai kepentingan algoritma *greedy* yang digunakan.

2.2.3 Cara Menjalankan Bot

Bot dapat dijalankan melalui GUI dan terminal. Cara menjalankan bot melalui GUI Robocode telah dijelaskan sebelumnya. Untuk melalui terminal, bot dapat dijalankan setelah menjalankan *game engine* dan *local server*. Setelah melakukan kedua hal tersebut, bots_secret yang berada pada file server.properties disalin untuk digunakan pada perintah yang akan dijalankan di terminal. Pada terminal masukan `export` diikuti `SERVER_SECRET=yourBotSecret`. Jika menggunakan windows cmd jalankan perintah berikut `set SERVER_SECRET=yourBotSecret`. Setelah itu, bot dapat dijalankan dengan `dotnet run`. Permainan dapat dimulai dengan melakukan *start battle* pada *game engine*. Jika bot sudah muncul pada kotak “Joined Bots”, maka bot sudah berhasil di boot dan dapat ditambahkan pada permainan.

BAB III

APLIKASI STRATEGI *GREEDY*

3.1 Mapping

Proses mapping dalam pembuatan bot menjadi elemen-elemen algoritma *greedy* meliputi himpunan kandidat, himpunan solusi, fungsi solusi, fungsi seleksi, fungsi kelayakan, dan fungsi objektif. Keenam elemen ini mempunyai isi yang berbeda tergantung dengan solusi *greedy* yang dipilih. Pada subbab selanjutnya akan dijelaskan mengenai elemen ini.

3.2 Alternatif Solusi *Greedy*

Dalam proses mengerjakan tugas besar ini, kelompok mencoba beberapa jenis algoritma *greedy*. Algoritma-algoritma tersebut berdasarkan pada serangan, posisi, deteksi, dan pergerakan acak.

3.2.1 *Greedy by Attack*

Greedy by attack adalah pendekatan algoritma *greedy* dengan memprioritaskan menyerang bot lain. Bot dengan algoritma ini berfokus pada melakukan serangan berupa *ramming* yang mengharuskan bot untuk berdekatan dengan bot musuh. Prioritas bot musuh yang diserang adalah dengan jarak tertentu. Jarak ini didapatkan ketika bot melakukan *scanning* terhadap sekitarnya. Bot akan selalu melakukan deteksi pada sekitarnya sehingga dapat dengan mudah berganti target. Bot memiliki gerakan dasar yang memungkinkan untuk melakukan *ramming* tanpa disengaja. Gerakan dasar ini berupa berputar searah jarum jam ketika bot belum mempunyai target serang. Pada percobaannya, gerakan ini menjadi efektif pula untuk menghindari serangan peluru dari bot musuh.

1. Mapping Elemen *Greedy*
    a. Himpunan kandidat: semua bot yang terdeteksi di arena.
    b. Himpunan solusi: bot yang ada di jarak tertentu.
    c. Fungsi solusi: memeriksa apakah bot yang di *scan* berada di jarak sesuai yang ditetapkan.

d. Fungsi seleksi: memilih bot yang akan diserang dan memastikan tetap *ramming*.
e. Fungsi kelayakan: bot memenuhi kriteria jarak dan terus *ramming* selama bot musuh masih hidup serta belum terdeteksi bot lain.
f. Fungsi objektif: menyesuaikan tenaga peluru dengan jarak bot dan hanya *ramming* ketika jarak sudah sangat dekat.

2. Analisis Efisiensi Solusi

Dalam pengaplikasian *greedy by attack*, kompleksitas keseluruhan adalah O(n). Hal ini berlaku untuk skenario terburuk dimana terdapat banyak bot musuh dan melakukan *scanning* terus menerus. Untuk kondisi lain kompleksitas terdapat pada O(1) dan hal ini sudah berlaku untuk semua prosedur yang ada pada implementasi strategi.

3. Analisis Efektivitas Solusi

Greedy by attack berasal dari dasar berfikir mendapatkan probabilitas tinggi untuk menyerang bot lain sebanyak mungkin. Penyerangan ini dilakukan dengan melakukan penembakan peluru dengan kekuatan penuh ketika jaraknya sangat dekat dengan bot musuh sembari melakukan *ramming* yang dapat menambahkan poin serta mengurangi energi bot musuh dengan cepat. Pendekatan ini akan cocok apabila bot berada dekat dengan bot lainnya dikarenakan jika bot musuh jauh, probabilitas peluru terkena musuh sangat kecil dan membutuhkan waktu untuk mendekati bot tersebut. Selain itu, terdapat permasalahan dimana bot akan berpindah dari satu sisi arena ke sisi lainnya untuk menyerang bot terdekat. Hal ini meningkatkan probabilitas terkena peluru "nyasar." Berikut adalah kondisi strategi ini akan efektif dan tidak efektif.

Strategi efektif jika:

- Terdapat banyak bot musuh yang dekat
- Tidak ada bot lain dengan konsep *ramming*
- Bot musuh letaknya berdekatan satu sama lain

Strategi tidak efektif jika:

- Semua bot musuh terletak jauh

- Terdapat bot musuh dengan strategi *ramming* dan *target lock*

3.2.2 *Greedy by Position*

Greedy by position adalah pendekatan algoritma *greedy* dengan memprioritaskan posisi pada arena, memanfaatkan dinding sebagai posisi yang strategis untuk meningkatkan *survivability*. Bot akan bergerak di sepanjang dinding, dengan gerakan bolak-balik atau osilasi, sambil melakukan pemindaian 180 derajat yang efisien dan memindai seluruh arena dengan baik. Strategi ini didasarkan pada pemikiran bahwa dengan berada di pinggir arena, akan terkena lebih sedikit peluru yang "nyasar" dan bot juga hanya perlu untuk memindai area di depannya saja. Implementasi bot ini juga dilengkapi dengan *escape mechanism,* untuk menyelamatkan dirinya ketika ditabrak oleh bot lain, memilih arah terbaik untuk kabur.

1. Mapping Elemen *Greedy*
    a. Himpunan kandidat: Semua kemungkinan posisi pada arena, semua kemungkinan arah gerakan untuk melarikan diri.
    b. Himpunan solusi: posisi dinding terdekat untuk diikuti, arah gerakan untuk melarikan diri yang berlawanan dengan arah serangan.
    c. Fungsi solusi: GoToNearestWall(), EmergencyEscape()
    d. Fungsi seleksi: memilih dinding terdekat dengan menghitung jarak minimum ke semua dinding, memilih sudut kabur optimal dengan mempertimbangkan arah badan bot, arah serangan, dan posisi bot.
    e. Fungsi kelayakan: jarak antara bot dengan wall, jarak antara bot dengan corner, arah body terhadap wall, arah gun terhadap body.
    f. Fungsi objektif: Memaksimalkan keamanan dengan bergerak disepanjang dinding, menjauh dari corner, menjaga jarak aman, memaksimalkan efisiensi gerakan dengan pemilihan arah untuk kabur dari situasi berbahaya, memaksimalkan area pemindaian.
2. Analisis Efisiensi Solusi

    Dalam pengaplikasian *greedy by position,* kompleksitas waktu yang dibutuhkan adalah O(1). Dalam setiap kasus, terdapat sekitar 9 kemungkinan

yang harus diperhatikan, dilihat dari posisi bot, arah badan bot, dan juga arah serangan.

3. Analisis Efektivitas Solusi

   Pendekatan *greedy by position* cukup efektif. Dalam penerapannya, masih terdapat beberapa kasus dimana bot tidak dapat kabur dari serangan ram bot lain sehingga sangat menurunkan *survivability* dari bot ini. Selain itu, karena jarak yang jauh dari lawan, lebih kecil kemungkinan bagi peluru yang ditembakkan oleh bot ini untuk mengenai lawan, sehingga poin yang didapatkan dari tembakan juga berkurang.

   Strategi efektif jika:

   - Escape mechanism bekerja untuk semua kasus, atau tidak ada bot lawan yang menggunakan pendekatan *greedy by attack,* dengan strategi *ramming.*

   Strategi tidak efektif jika:

   - Terdapat bot yang menggunakan pendekatan *greedy by attack,* dengan strategi *ramming.*

3.2.3 *Greedy by First Detected*

Greedy by first detected adalah pendekatan algoritma *greedy* dengan memprioritaskan bot yang pertama terdeteksi. Untuk mendukung algoritma *greedy* ini, bot akan melakukan scan berulang-ulang secara *sweeping* agar dapat menemukan bot di arena. Setelah musuh terdeteksi, bot berkonsep *greedy* ini akan terus menerus menyerang bot tersebut hingga musuh hancur atau keluar dari radar deteksi bot.

Percobaan pertama penyerangan adalah dengan penembakan peluru ke bot terdeteksi, sedangkan percobaan kedua adalah melakukan *ramming*. Kelompok memutuskan untuk menggunakan algoritma ini pada bot yang menembakkan peluru karena lebih aman dibandingkan bot *ramming* yang pada kasus terburuk harus menelusuri jarak yang jauh untuk melakukan *ramming*.

Lalu, seperti algoritma sebelumnya, bot berkonsep *greedy by first detected* mempunyai gerakan melingkar untuk mengurangi probabilitas terkena peluru dari bot lain.

1. Mapping Elemen *Greedy*
    a. Himpunan kandidat: semua bot yang ada di arena.
    b. Himpunan solusi: bot terdekat.
    c. Fungsi solusi: memeriksa apakah bot terdeteksi pada radar.
    d. Fungsi seleksi: memilih bot pertama yang terdeteksi pada radar.
    e. Fungsi kelayakan: bot merupakan yang pertama kali terdeteksi pada radar.
    f. Fungsi objektif: menyesuaikan tenaga peluru dengan jarak bot.
2. Analisis Efisiensi Solusi

    Dalam pengaplikasian *greedy by first detected*, untuk mencari bot yang pertama kali terdeteksi diperlukan kompleksitas O(1) yang berlaku untuk metode Run(), OnScannedBot(), dll.
3. Analisis Efektivitas Solusi

    Greedy by first detected berasal dari dasar berfikir menyerang bot lain secepat mungkin untuk mendapatkan poin, yaitu memilih bot yang pertama kali terdeteksi. Pendekatan ini akan cocok apabila bot berada di sekitar bot lain dalam jarak dekat hingga menengah karena meningkatkan probabilitas serangan bot berhasil. Sedangkan, kelemahannya adalah apabila bot lain yang pertama kali terdeteksi berjarak jauh dan bergerak-gerak karena mengurangi probabilitas serangan bot berhasil. Bot akan terus menerus menyerang bot tersebut secara *tracking,* walaupun ada bot lain yang berjarak lebih dekat ke bot ini. Berikut adalah kondisi strategi ini akan efektif dan tidak efektif.

    Strategi efektif jika:
    - Terdapat banyak bot musuh yang yang berjarak dekat ke menengah.
    - Bot yang terdeteksi relatif tidak banyak bergerak

    Strategi tidak efektif jika:
    - Banyak bot musuh yang berjarak jauh dan bergerak-gerak

3.2.4 *Greedy by Random Movement*

Greedy by random movement adalah pendekatan algoritma *greedy* yang berfokus pada pergerakan bot yang acak. Setiap turn nya, bot akan secara acak,

dengan kemungkinan 50 50 bergerak maju sambil berputar 30 derajat ke arah kiri atau 30 derajat ke arah kanan. Untuk meningkatkan keacakan, ditambahkan 20 persen kemungkinan bot untuk diam dan tidak bergerak sama sekali. Pendekatan ini terpikirkan dengan asumsi bahwa gerakan yang sulit diprediksi akan menyulitkan lawan dalam membidik secara akurat. Pada percobaan, dengan pergerakan acak, bot dapat menghindari tembakan dari musuh dengan baik.

1. Mapping Elemen *Greedy*
    a. Himpunan kandidat: semua kemungkinan gerakan (maju ke kiri, maju ke kanan, stop).
    b. Himpunan solusi: gerakan yang terpilih secara acak. 20 persen kemungkinan untuk diam, 40 persen kemungkinan untuk maju ke kiri, dan 40 persen kemungkinan untuk maju ke kanan.
    c. Fungsi solusi: menghitung angka acak.
    d. Fungsi seleksi: memilih gerakan yang sesuai dengan angka acak yang terpilih.
    e. Fungsi kelayakan: bot memiliki energi yang cukup untuk menembak dan gunHeat = 0.
    f. Fungsi objektif: Memaksimalkan tenaga peluru.
2. Analisis Efisiensi Solusi

    Dengan algoritma yang hanya menggunakan perhitungan yang sederhana, kompleksitas waktu untuk setiap pergerakan bot ini adalah O(1) untuk setiap turn nya. Ketika menembak bot lain, kompleksitas nya juga O(1) karena bot akan langsung menembak bot yang terpindai.
3. Analisis Efektivitas Solusi

    Gerakan *strafe* acak sangat efektif, membuat bot sulit ditarget oleh bot lain. Selain itu, dengan menambahkan variasi stop, bot menjadi lebih susah untuk diprediksi musuh. Namun, terkadang pergerakan acak ini membawa bot menabrak wall dan mengurangi energi dari bot.

    Strategi efektif jika:

    - Terdapat banyak bot musuh yang menggunakan prediksi gerakan linear.

    Strategi tidak efektif jika:

- Pergerakan random mengarahkan bot ke arah dinding dan menabrak dinding.

3.3 Strategi *Greedy* yang dipilih

Seluruh strategi *greedy* yang disebutkan pada subbab sebelumnya diimplementasikan pada bot utama dan bot alternatif. Perbedaannya, bot utama adalah bot yang akan digunakan dalam pertandingan dengan kelompok lain karena penting untuk memilih bot utama yang memiliki probabilitas kemenangan yang tinggi. Sedangkan, bot dengan strategi *greedy* lainnya dapat diimplementasikan sebagai alternatif dari bot utama.

Berdasarkan hasil diskusi dan pengujian yang dilakukan, kelompok memutuskan untuk menggunakan **strategi *greedy by attack*** dan bot **NearbyRam**. Alasan dibalik pemilihan strategi agresif adalah berdasarkan hasil pengujian yang menunjukkan perbedaan skor yang signifikan. Secara teoritis, merupakan hal yang logis untuk mengutamakan penyerangan karena akan menghasilkan skor tertinggi yang menjadi tujuan utama permainan. Selain itu, kelompok memprediksi bahwa bot kelompok lain tidak mempunyai mekanisme lari (*escape mechanism*) dari *ramming.* Pertimbangan lain dari kelompok adalah bot **RandomStrafe** yang mempunyai keunggulan dengan pergerakannya yang tidak dapat diprediksi dan bot **Johnny** yang mempunyai keberlangsungan hidup yang tinggi.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Alternatif Solusi

Berikut adalah penjelasan prosedur/fungsi untuk setiap bot yang diimplementasi oleh kelompok ditampilkan dengan pseudocode. Prosedur main tidak disertakan karena untuk semua bot sama dimana prosedur main digunakan untuk membaca konfigurasi dari file .json, membuat objek bot, dan memulai bot.

4.1.1 NearbyRam Bot

Pada kelas NearbyRam

1. Prosedur Run()

   Prosedur untuk mengatur warna bot dan perilaku gerakan bot. Dalam bot NearbyRam, gerakan dasar yang akan di-*loop* selama bot berjalan adalah berputar searah jarum jam.

```
procedure Run()
    { Set the colors of the bot }
    BodyColor ← Color.Cyan;
    TurretColor ← Color.Pink;
    RadarColor ← Color.White;
    ScanColor ← Color.Black;
    BulletColor ← Color.Red;

    while (IsRunning) do
        if (DistanceRemaining=0) then
            SetTurnLeft(10_000)
            MaxSpeed ← 8
            Forward(10_000)
```

2. Prosedur OnScannedBot(ScannedBotEvent evt)

   Prosedur untuk menentukan target *ramming* dan menembak semua bot yang terdeteksi.

```
procedure OnScannedBot(input ScannedBotEvent evt)
```

```
        // shot all the bot that is scanned
        if (DistanceTo(evt.X, evt.Y) < 100) then
                 Fire(3)
        else if (DistanceTo(evt.X, evt.Y) < 300) then
                Fire(2)
        else
                Fire(1)

        // ram the nearest bot
        var bearing ← BearingTo(evt.X, evt.Y)
        SetTurnLeft(bearing)
        if (DistanceTo(evt.X, evt.Y) < 50) then
            SetForward(DistanceTo(evt.X, evt.Y));
        else
            SetForward(DistanceTo(evt.X, evt.Y)/2);
```

3. Prosedur OnHitWall(HitWallEvent e)

   Prosedur untuk mengatur bot berjalan mundur ketika menabrak dinding untuk memastikan bahwa bot tidak terkena dinding lagi saat kembali berjalan.

```
procedure OnHitWall(input HitWallEvent e)
    Back(100)
```

4. Prosedur OnHitBot(HitBotEvent e)

   Prosedur untuk mengatur variabel mundur. Jika bot adalah yang melakukan *ramming*, mengurangi variabel mundur dan jika sebaliknya, variabel mundur ditambahkan.

```
procedure OnHitBot(input HitBotEvent e)
        if (e.IsRammed) then
            SetForward(DistanceTo(evt.X, evt.Y));
```

4.1.2 Johnny Bot

1. Prosedur Run()

   Prosedur untuk mengatur warna bot dan perilaku gerakan bot. Dalam bot Johnny, gerakan dasar yang akan di-*loop* selama bot berjalan adalah

berputar searah jarum jam dengan kondisi awal *gun* mengikuti pergerakan *body*.

```
procedure Run()
    BodyColor ← Color.Cyan
    TurretColor ← Color.Yellow
    RadarColor ← Color.Cyan
    ScanColor ← Color.DeepPink
    BulletColor ← Color.Yellow

    while (IsRunning) do
        AdjustGunForBodyTurn ← false;
        SetTurnLeft(10000)
        MaxSpeed ← 6
        Forward(10000)
```

2. Prosedur OnScannedBot(ScannedBotEvent evt)

   Prosedur untuk mengukur jarak dan arah bot yang terdeteksi, mengatur arah gun ke arah bot tersebut, dan melakukan Smart Fire – penembakan yang kondisional berdasarkan jarak musuh dan energi diri –. Prosedur disertai juga dengan pengaturan kecepatan maksimal bot yang dikembalikan ke default state setelah kecepatan diubah ke maks saat collision dengan bot lain. Kemudian, bot akan melakukan scan musuh secara *sweeping* sebanyak 450 derajat ke kanan, kemudian 450 derajat kembali ke kiri tiap pengulangan prosedur.

```
procedure OnScannedBot(input ScannedBotEvent e)
    { Enable gun tracking enemy on scan, independent of body
movement }
    AdjustGunForBodyTurn ← true

    { Get bearing and distance to enemy }
    bearing ← BearingTo(e.X, e.Y)
    distance ← DistanceTo(e.X, e.Y)

    { Calculate enemy angle }
```

```
    absoluteBearing ← Direction + bearing

    { Calculate angle to turn gun }
    gunTurn ← absoluteBearing - GunDirection

    { Turn gun to face enemy }
    SetTurnGunRight(gunTurn)

    { SmartFire, fire based on enemy's distance and current
energy }

    if (distance > 500) then
        SetFire(1.5)
    else if (distance > 200 and distance <= 500 and Energy >
10) then
        SetFire(2.3)
    else if (distance <= 200 and Energy > 10) then
        SetFire(3)
    else if (distance < 50 and Energy < 10) then
        SetFire(1.5)

    SetForward(10000)

    { Return speed back to original velocity some time after
OnHitBot }
    if (velocityCounter = 3) then
        MaxSpeed ← 6
        velocityCounter ← 0

    { Invert gun sweep direction per turn }
    gunDirection ← -gunDirection
    SetTurnGunRight(450 * gunDirection)
    Go()
    velocityCounter ← velocityCounter + 1
```

3. Prosedur OnHitBot(HitBotEvent e)

   Prosedur untuk mengatur perlakuan bot ketika terjadi collision dengan bot lain, yaitu dengan mengubah kecepatan maksimal bot ke 8 (tertinggi).

```
procedure OnHitBot(input HitBotEvent e)
    MaxSpeed ← 8
```

4. Prosedur Run()

   Prosedur untuk mengatur perlakuan bot ketika menabrak dinding permainan dengan mundur sebanyak 100 satuan.

```
procedure OnHitWall(input HitWallEvent e)
    SetBack(100)
```

4.1.3 Border Bot

1. Prosedur Run()

   Prosedur untuk mengatur warna bot dan perilaku gerakan bot. Dalam bot Border, gerakan dasar yang akan di-*loop* selama bot berjalan adalah GoToNearestWall(), atau PatrolWall() diikuti ScanSweepRadar().

```
procedure Run()
   { Initialize state }
   ResetState()

   { Set bot colors }
   BodyColor ← Color.Gray
   GunColor ← Color.DarkGray
   RadarColor ← Color.LightGray
   ScanColor ← Color.Yellow
   BulletColor ← Color.Red

   while (IsRunning) do
      if (DistanceToWall() > WALL_THRESHOLD and not escaping
and not awal) then
         ResetState()
```

```
      if (not awal2) then
         ScanSweepRadar()
      if (escaping) then
         if (DistanceRemaining = 0) then
            escaping ← false
            StopReset()
            if (DistanceToWall() > WALL_THRESHOLD) then
               ResetState() { GoToNearestWall() }
            else
               UpdateWallFlags()
      else if (awal) then
         GoToNearestWall()
         awal ← false
      else
         if (IsNearCorner()) then
            AwayFromCorner() { avoid corner }
         else
            PatrolWall() { osilasi di dinding }
      Go()
```

2. Prosedur ScanSweepRadar()

Prosedur untuk mengatur pergerakan pistol dan radar. Dengan SCAN_SWEEP merupakan konstan yang bernilai 180, pistol dan radar akan berotasi ke kanan sejauh 180 derajat, lalu kembali ke kiri sejauh 180 derajat, dan seterusnya.

```
procedure ScanSweepRadar()
   if (kiri) then
      if (GunTurnRemaining = 0) then
         SetTurnGunLeft(SCAN_SWEEP)
         kiri ← false
      else
         if (GunTurnRemaining = 0) then
            SetTurnGunRight(SCAN_SWEEP)
            kiri ← true
```

3. Prosedur ResetState()

Prosedur untuk mengatur ulang semua state yang dimiliki bot kembali menjadi *default*.

```
procedure ResetState()
   kiri ← false
   maju ← true
   awal ← true
   awal2 ← true
   escaping ← false
   OnLeftWall ← false
   OnRightWall ← false
   OnTopWall ← false
   OnBottomWall ← false
   lastEnemyID ← -1
   lastEnemyDirection ← -1
```

4. Prosedur PatrolWall()

Prosedur untuk mengatur pergerakan bot, yaitu pergerakan osilasi maju-mundur, dan dilengkapi dengan penyesuaian arah tembak dengan arah badan bot.

```
procedure PatrolWall()
   { Enable independent gun movement }
   AdjustGunForBodyTurn ← true

   if (awal2) then
      { Align gun with body initially }
      angleToBody ← GunDirection - Direction
      TurnGun(angleToBody)
      awal2 ← false
   else
      if (maju) then
         if (DistanceRemaining = 0) then
            SetForward(150)
            maju ← false
         else
```

```
            if (DistanceRemaining = 0) then
               SetBack(150)
               maju ← true
```

5. Prosedur OnScannedBot(ScannedBotEvent e)

   Prosedur untuk menembak target yang terdeteksi, dan mencari arah dari lawan yang menabrak (ramming) bot ini.

```
procedure OnScannedBot(input ScannedBotEvent e)
   SetFire(3)

   if (e.ScannedBotId = lastEnemyID) then
      lastEnemyDirection ← e.Direction
```

6. Prosedur OnHitBot(HitBotEvent e)

   Prosedur untuk kabur jika ditabrak bot lawan atau collision dengan bot lawan.

```
procedure OnHitBot(input HitBotEvent e)
   Interruptible ← false
   lastEnemyID ← e.VictimId
   if (not escaping) then
      EmergencyEscape()
```

7. Prosedur OnHitWall(HitWallEvent e)

   Prosedur untuk kabur jika menabrak dinding.

```
procedure OnHitWall(input HitWallEvent e)
   Interruptible ← false
   if (not escaping) then
      EmergencyEscape()
```

8. Prosedur EmergencyEscape()

   Prosedur untuk kabur Ketika ditabrak bot lain, akan lari ke arah yang berlawanan.

```
procedure EmergencyEscape()
```

```
if lastEnemyDirection = -1 then
   →
if (not escaping) then
   StopReset()  { Prioritas escape }
escaping ← true

inLeftWallArea ← (X < CORNER_THRESHOLD)
inRightWallArea ← ((ArenaWidth - X) < CORNER_THRESHOLD)
inBottomWallArea ← (Y < CORNER_THRESHOLD)
inTopWallArea ← ((ArenaHeight - Y) < CORNER_THRESHOLD)

cornersNearby ← 0
if inLeftWallArea then
   cornersNearby ← cornersNearby + 1
if inRightWallArea then
   cornersNearby ← cornersNearby + 1
if inBottomWallArea then
   cornersNearby ← cornersNearby + 1
if inTopWallArea then
   cornersNearby ← cornersNearby + 1

{ Hubungkan rammedDirection dengan posisi terakhir musuh}
rammedDirection ← lastEnemyDirection
escapeAngle ← 0

if OnTopWall or OnBottomWall then
   { If rammed from right }
   if rammedDirection > 90 and rammedDirection < 270 then
      escapeAngle ← 180  { move left }
   else { If rammed from left }
      escapeAngle ← 0  { move right }
else if OnLeftWall or OnRightWall then
   { If rammed from below }
   if rammedDirection < 180 then
      escapeAngle ← 90  { move up }
   else { If rammed from right }
      escapeAngle ← 270  { move down }
```

```
   else
      escapeAngle ← rammedDirection + 180

      if escapeAngle > 360 then
         escapeAngle ← escapeAngle - 360

   if escapeAngle < 0 then
      escapeAngle ← escapeAngle + 360

   { corner handling }
   if cornersNearby >= 2 then
      if inLeftWallArea and inBottomWallArea then
         if rammedDirection >= 45 and rammedDirection < 225
then
            escapeAngle ← 90  { move up }
         else
            escapeAngle ← 0  { move right }

      else if inLeftWallArea and inTopWallArea then
         if rammedDirection >= 135 and rammedDirection < 315
then
            escapeAngle ← 270  { move down }
         else
            escapeAngle ← 0  { move right }

      else if inRightWallArea and inBottomWallArea then
         if rammedDirection ≥ 135 and rammedDirection < 315
then
            escapeAngle ← 180  { move left }
         else
            escapeAngle ← 90  { move up }

      else if inRightWallArea and inTopWallArea then
         if rammedDirection >= 45 and rammedDirection < 225
then
            escapeAngle ← 180  { move left }
         else
```

```
            escapeAngle ← 270  { move down }



   turnAngle ← escapeAngle - Direction
   if escapeAngle = Direction then
      SetForward(200)  { searah, lari maju }
   else if escapeAngle = Direction + 180 or escapeAngle =
Direction - 180 then
      SetBack(200)  { berlawanan, lari mundur }
   else { corner atau not on wall, lari belok }
      SetTurnBody(turnAngle)
      SetForward(200)
      awal2 ← true
      { untuk ketika ke wall lagi, adjust gun to body }
      if turnAngle < 0 then
         kiri ← false
      else
         kiri ← true

   MaxSpeed ← 8
   SetTurnGunLeft(10000)
```

9. Prosedur StopReset()

   Prosedur untuk menghapus semua pergerakan yang ada saat ini.

```
procedure StopReset()
   SetForward(0)
   SetBack(0)
   SetTurnLeft(0)
   SetTurnRight(0)
   SetTurnGunLeft(0)
```

10. Fungsi DistanceToWall() → double

    Fungsi yang menghitung jarak dari bot ke dinding terdekat.

```
private function DistanceToWall() → double
   distanceToLeft ← X
```

```
   distanceToRight ← ArenaWidth - X
   distanceToBottom ← Y
   distanceToTop ← ArenaHeight - Y
   → Math.Min(Math.Min(distanceToLeft, distanceToRight),
               Math.Min(distanceToBottom, distanceToTop))
```

11. Fungsi IsNearCorner() → boolean

    Fungsi yang mengembalikan true jika jarak bot ke salah satu corner kurang dari CORNER_TRESHOLD yaitu 100.

```
function IsNearCorner() → bool
   → (X < CORNER_THRESHOLD or (ArenaWidth - X) <
CORNER_THRESHOLD) and
      (Y < CORNER_THRESHOLD or (ArenaHeight - Y) <
CORNER_THRESHOLD)
```

12. Prosedur AwayFromCorner()

    Prosedur untuk menjauh dari corner.

```
procedure AwayFromCorner()
   if X < CORNER_THRESHOLD then
      if Y < CORNER_THRESHOLD then
         { Bottom left corner }
         if (Direction = 0) or (Direction = 90) then
            SetForward(100)
         else if (Direction = 180) or (Direction = 270) then
            SetBack(100)
      else
         { Top left corner }
         if (Direction = 0) or (Direction = 270) then
            SetForward(100)
         else if (Direction = 90) or (Direction = 180) then
            SetBack(100)
   else
      if Y < CORNER_THRESHOLD then
         { Bottom right corner }
         if (Direction = 90) or (Direction = 180) then
```

```
            SetForward(100)
         else if (Direction = 0) or (Direction = 270) then
            SetBack(100)
      else
         { Top right corner }
         if (Direction = 180) or (Direction = 270) then
            SetForward(100)
         else if (Direction = 0) or (Direction = 90) then
            SetBack(100)
```

13. Prosedur UpdateWallFlags

Prosedur untuk memperbarui state bot berada. Untuk contoh, jika OnTopWall bernilai `true`, maka saat ini bot sedang berada di dinding bagian atas.

```
procedure UpdateWallFlags() {
   { Reset all flags }
   OnTopWall ← false
   OnBottomWall ← false
   OnLeftWall ← false
   OnRightWall ← false

   distToLeft ← X
   distToRight ← ArenaWidth - X
   distToBottom ← Y
   distToTop ← ArenaHeight - Y

   minDist ← Math.Min(Math.Min(distToLeft, distToRight),
                      Math.Min(distToBottom, distToTop));

   if (minDist < WALL_THRESHOLD) then
      if (minDist = distToLeft) then
         OnLeftWall ← true
      else if (minDist = distToRight) then
         OnRightWall ← true
      else if (minDist = distToBottom) then
         OnBottomWall ← true
```

```
      else if (minDist = distToTop) then
         OnTopWall ← true
```

14. Prosedur GoToNearestWall()

Prosedur untuk bergerak ke dinding terdekat.

```
procedure UpdateWallFlags()
   UpdateWallFlags()

   distToLeft ← X
   distToRight ← ArenaWidth - X
   distToBottom ← Y
   distToTop ← ArenaHeight - Y

   minDist ← Math.Min(Math.Min(distToLeft, distToRight),
                      Math.Min(distToBottom, distToTop));

   if (minDist = distToLeft) then
      { Left wall is closest }
      angleToWall ← DirectionTo(0, Y)
      distanceToWall ← DistanceTo(0, Y)
      OnLeftWall ← true
   else if (minDist = distToRight) then
      { Right wall is closest }
      angleToWall ← DirectionTo(ArenaWidth, Y)
      distanceToWall ← DistanceTo(ArenaWidth, Y)
      OnRightWall ← true
   else if (minDist = distToBottom) then
      { Bottom wall is closest }
      angleToWall ← DirectionTo(X, 0)
      distanceToWall ← DistanceTo(X, 0)
      OnBottomWall ← true
   else if (minDist = distToTop) then
      { Top wall is closest }
      angleToWall ← DirectionTo(X, ArenaHeight)
      distanceToWall ← DistanceTo(X, ArenaHeight)
      OnTopWall ← true
```

```
   turnAngle ← angleToWall - Direction
   TurnBody(turnAngle)
   Forward(distanceToWall - WALL_DISTANCE)
   TurnRight(90)
   awal2 ← true
   kiri ← false
```

15. Prosedur TurnBody(double angle)

    Prosedur helper, untuk normalisasi angle, dan memilih arah rotasi yang lebih optimal.

```
procedure TurnBody(input double angle)
   while (angle < 0) do
      angle ← angle + 360

   while (angle >= 360) do
      angle ← angle - 360

   if (angle <= 180) then
      TurnLeft(angle)
   else
      TurnRight(360 - angle)
```

16. Prosedur SetTurnBody(double angle)

    Sama seperti procedure TurnBody, hanya saja bukan *blocking commands.*

```
procedure SetTurnBody(input double angle)
   while (angle < 0) do
      angle ← angle + 360

   while (angle >= 360) do
      angle ← angle - 360

   if (angle <= 180) then
      SetTurnLeft(angle)
   else
```

```
      SetTurnRight(360 - angle)
```

17. Prosedur TurnGun(double angle)

    Prosedur helper, untuk normalisasi angle, dan memilih arah rotasi yang lebih optimal.

```
procedure TurnGun(input double angle)
   while (angle < 0) do
      angle ← angle + 360

   while (angle >= 360) do
      angle ← angle - 360

   if (angle <= 180) then
      TurnGunRight(angle)
   else
      TurnGunLeft(360 - angle)
```

18. Prosedur SetTurnGun(double angle)

    Sama seperti procedure TurnGun, hanya saja bukan *blocking commands.*

```
procedure SetTurnGun(input double angle)
   while (angle < 0) do
      angle ← angle + 360

   while (angle >= 360) do
      angle ← angle - 360

   if (angle <= 180) then
      SetTurnGunRight(angle)
   else
      SetTurnGunLeft(360 - angle)
```

4.1.4 RandomStrafe Bot

1. Prosedur Run()

Prosedur untuk gerakan dasar bot yang dijalankan selama bot masih hidup. Bot memiliki kemungkinan sebesar 50% untuk bergerak ke kanan atau kiri dan 20% untuk berhenti.

```
procedure Run()
    BodyColor ← Color.Black
    TurretColor ← Color.Red
    RadarColor ← Color.Yellow
    ScanColor ← Color.Orange
    TracksColor ← Color.Gray
    GunColor ← Color.Red

    // Memisahkan gun dan radar dari body
    AdjustGunForBodyTurn ← true;
    GunTurnRate ← MaxGunTurnRate;

    while (IsRunning) do
        SetTurnGunLeft(10000)
        if (isStopped) then
            isStopped ← false
            // 50 50 kemungkinan untuk strafe ke kanan atau
kiri
            if (rand.Next(2) = 0) then
                strafeDirection ← 1
            else strafeDirection ← -1
        else
// jika strafeDirection = 1, akan strafe ke kanan. Sebaliknya
jika strafeDirection = -1, akan strafe ke kiri
            SetTurnRight(30 * strafeDirection)
            SetForward(100)
                // 20 persen kemungkinan untuk Stop
                if (rand.Next(5) == 0) then
                    isStopped ← true
                    SetStop()
        Go()
```

2. Prosedur OnScannedBot(ScannedBotEvent e)

   Prosedur yang memerintahkan bot untuk menembak setiap bot yang terdeteksi oleh radar.

```
procedure OnScannedBot(input e:ScannedBotEvent)
    Fire(3)
```

3. Prosedur OnHitWall(HitWallEvent e)

   Prosedur untuk mengatur bot saat menabrak dinding. Dilakukan untuk menjaga bot tidak menabrak dinding berkali-kali.

```
procedure OnHitWall(input e:HitWallEvent)
    TurnRight(180)
    Forward(150)
```

4. Prosedur OnHitBot(HitBotEvent e)

   Prosedur yang memungkinkan bot untuk menembak bot musuh saat terjadi tabrakan dengan dirinya. Keputusan menembak diambil ketika letak bot musuh berada dalam 10 derejat di depannya dan senjata tidak berada di kondisi menunggu *cooldown*.

```
procedure OnHitBot(input e:HitBotEvent)
    double bearing ← BearingTo(e.X, e.Y)
    if (Math.Abs(bearing) < 10 and GunHeat = 0) then
        Fire(Math.Min(3, Energy - 0,1)
```

4.2 Penjelasan Struktur Data

Bot yang digunakan dalam Robocode Royal Tank dikembangkan dalam bahasa C#. Berikut adalah struktur data yang dimiliki oleh bot utama.

```
├── src/
│   ├── main-bot/
│   │   ├── NearbyRam/
│   │   │   ├── NearbyRam.cmd
│   │   │   ├── NearbyRam.cs
│   │   │   ├── NearbyRam.csproj
│   │   │   ├── NearbyRam.json
│   │   │   ├── NearbyRam.sh
```

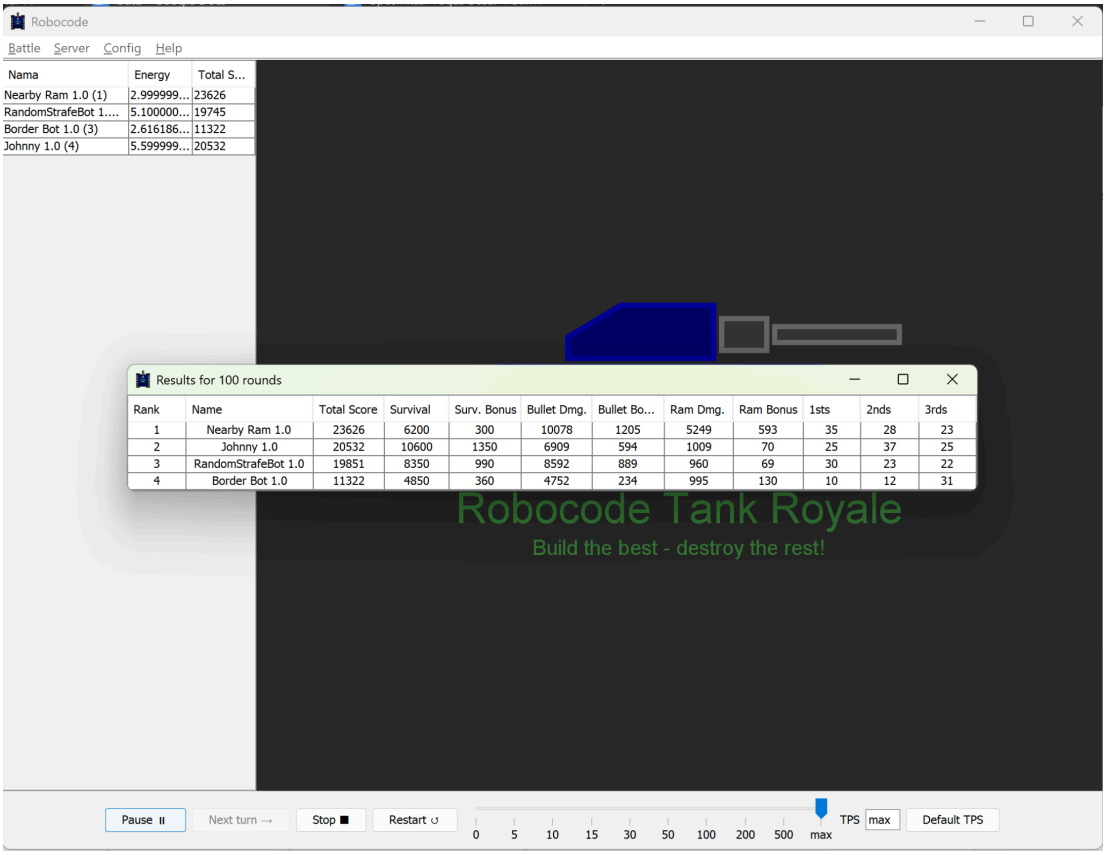
Folder src adalah folder yang berisi kode untuk masing-masing bot, baik bot utama maupun bot alternatif. Folder main-bot dan nama bot utama digunakan untuk

menyimpan keperluan bot utama yang dipilih. File .cmd, .cs, .csproj, .json, dan .sh adalah file yang dibutuhkan oleh satu bot agar bisa berjalan pada Robocode yang dimana kegunaannya sudah dijelaskan pada subbab lain. Penjelasan mengenai fungsi/prosedur pada bot utama sudah dijelaskan pada subbab 3.2.1 dan 4.1.1.

4.3 Analisis dan Pengujian

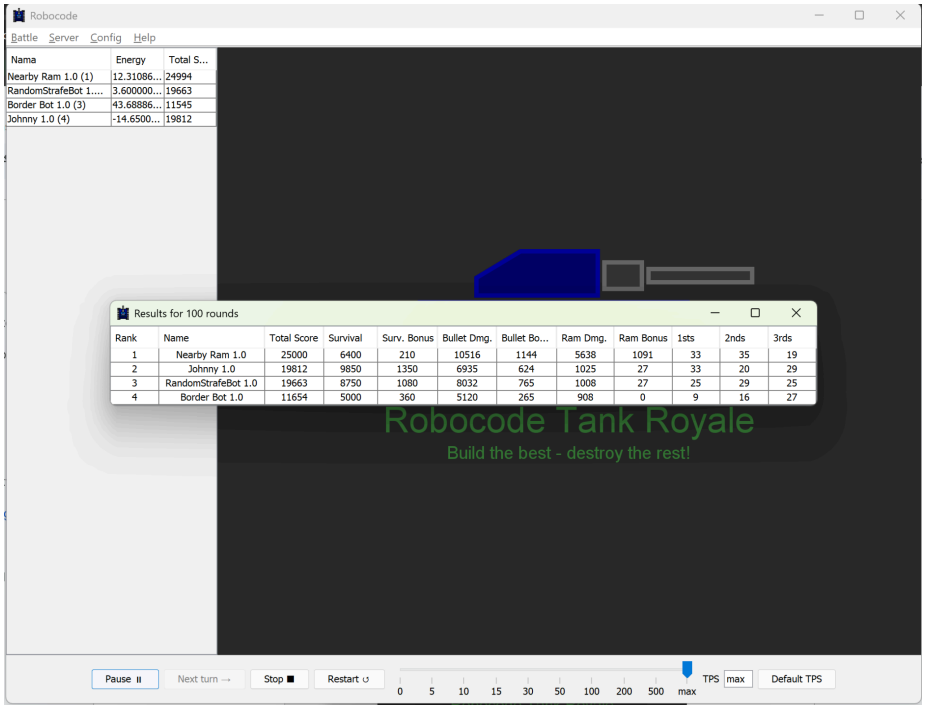
Pengujian dilakukan dengan melakukan 100 *turn* dengan keempat bot – dari strategi algoritma *greedy* yang dijelaskan pada laporan ini – ditandingkan satu sama lain. Berikut adalah hasil dari tiga kali pertandingan yang dilakukan.

1. Pengujian Pertama



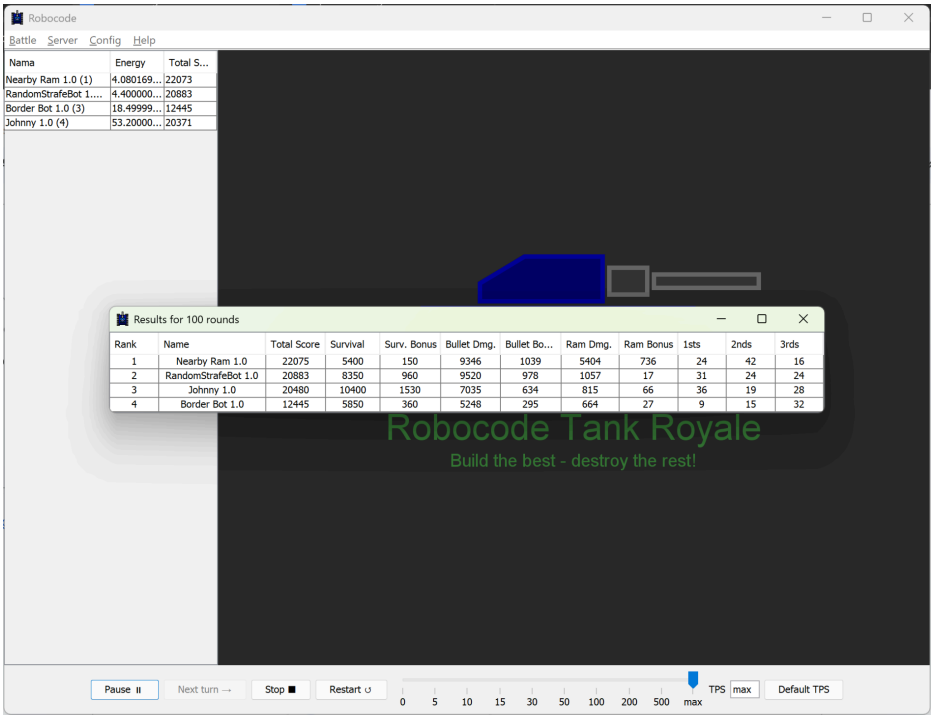
Gambar 1. Pertandingan Pertama. Sumber: Anggota kelompok

2. Pengujian Kedua



Gambar 2. Pertandingan Kedua. Sumber: Anggota kelompok

3. Pengujian Ketiga



Gambar 3. Pertandingan Ketiga. Sumber: Anggota kelompok

Berdasarkan hasil pengujian, dapat dilihat bahwa hasil dari ketiga pengujian tidak menunjukkan perubahan yang signifikan. Nearby Ram secara konsisten menempati peringkat pertama dengan perbedaan skor yang cukup besar dengan peringkat 2. Bot Johnny dan Random Strafe bergantian menempati posisi kedua dan ketiga, sedangkan Border selalu menempati peringkat terakhir dengan perbedaan skor yang sangat jauh. Total skor dari Nearby Ram didominasi oleh *Bullet Damage* dan *Ram Damage* yang sangat besar. Walaupun *survival* s*core*-nya sangat rendah, dengan pendekatan agresif, Nearby Ram dapat menghasilkan total skor yang paling optimal.

Dilihat dari sistem skoring, bot akan mendapatkan poin sebesar damage yang diberikan kepada lawan ketika menggunakan peluru. Damage yang diberikan kepada lawan adalah sebesar *4 * firepower + max (0, 2 * (firepower - 1)*. Tidak hanya itu, jika peluru terkena musuh, bot akan mendapatkan energy recovery sebesar *3 * firepower*. Dengan multiplier-multiplier yang cukup besar ini ditambah dengan 20% poin bonus ketika berhasil membunuh musuh (30% jika dengan ramming), strategi agresif lebih diuntungkan.

Dengan pendekatan *greedy by attack*, bot akan bergerak mendekati bot lain untuk melakukan ramming. Secara tidak langsung, jarak antara kedua bot tersebut menjadi minimal, dan jika peluru ditembakkan, besar kemungkinan peluru akan mengenai musuh. Dengan demikian, poin yang didapatkan dari *Bullet Damage* juga menjadi maksimal, ditambah dengan poin dari *Ram Damage* yang besar meskipun mengorbankan skor *survival*-nya.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dalam pengerjaan tugas besar ini, kelompok mempelajari cara mengimplementasikan algoritma *greedy* melalui pendekatan yang menyenangkan. Dengan menggunakan media berupa Robocode, kelompok dapat melihat hasil dari implementasi bersama dengan kelompok lain yang merupakan pengalaman menyenangkan dan memberikan rasa *refreshing*. Selain itu, kelompok mendapatkan pengetahuan dalam menggunakan bahasa C#. Dari seluruh bot yang diimplementasikan oleh kelompok, ditentukan NearbyRam yang menggunakan algoritma *greedy by attack* sebagai bot utama yang harapannya merupakan bot dengan efektivitas dan peluang menang tertinggi.

5.2 Saran

Saran untuk implementasi strategi algoritma *greedy* pada bot Robocode adalah sebagai berikut.

1. Berkomunikasi mengenai strategi algoritma yang ingin diimplementasi dengan lebih jelas.
2. Mengerjakan implementasi strategi algoritma dengan maksimal dan tidak menunda pekerjaan.
3. Peningkatan penulisan komentar pada kode agar mudah dimengerti oleh anggota kelompok yang lain.
4. Melakukan pengujian pada strategi algoritma *greedy* yang lain dan jika memungkinkan lebih banyak dari yang dilakukan oleh kelompok saat ini untuk melihat strategi *greedy* yang paling optimal di antara semua pilihan yang ada.

LAMPIRAN

1. Repository Github

   https://github.com/BerthaSoliany/Tubes1_satu

2. Video

   https://youtu.be/QiYT7tmqdGU

3. Tabel

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

DAFTAR PUSTAKA

Munir, Rinaldi. 2021. *Algoritma Greedy*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf. Diakses pada 21 Maret 2025.

Robocode. *What is Robocode?*. https://robocode-dev.github.io/tank-royale/articles/intro.html. Diakses pada 21 Maret 2025.