Tugas Besar 1 IF2211 Strategi Algoritma
Semester II tahun 2022/2023

Pemanfaatan Algoritma Greedy
dalam pembuatan bot permainan Robocode Tank Royale

Disusun oleh:
Kelompok Septic tank 1000 liter biomif Rp.1.500.000

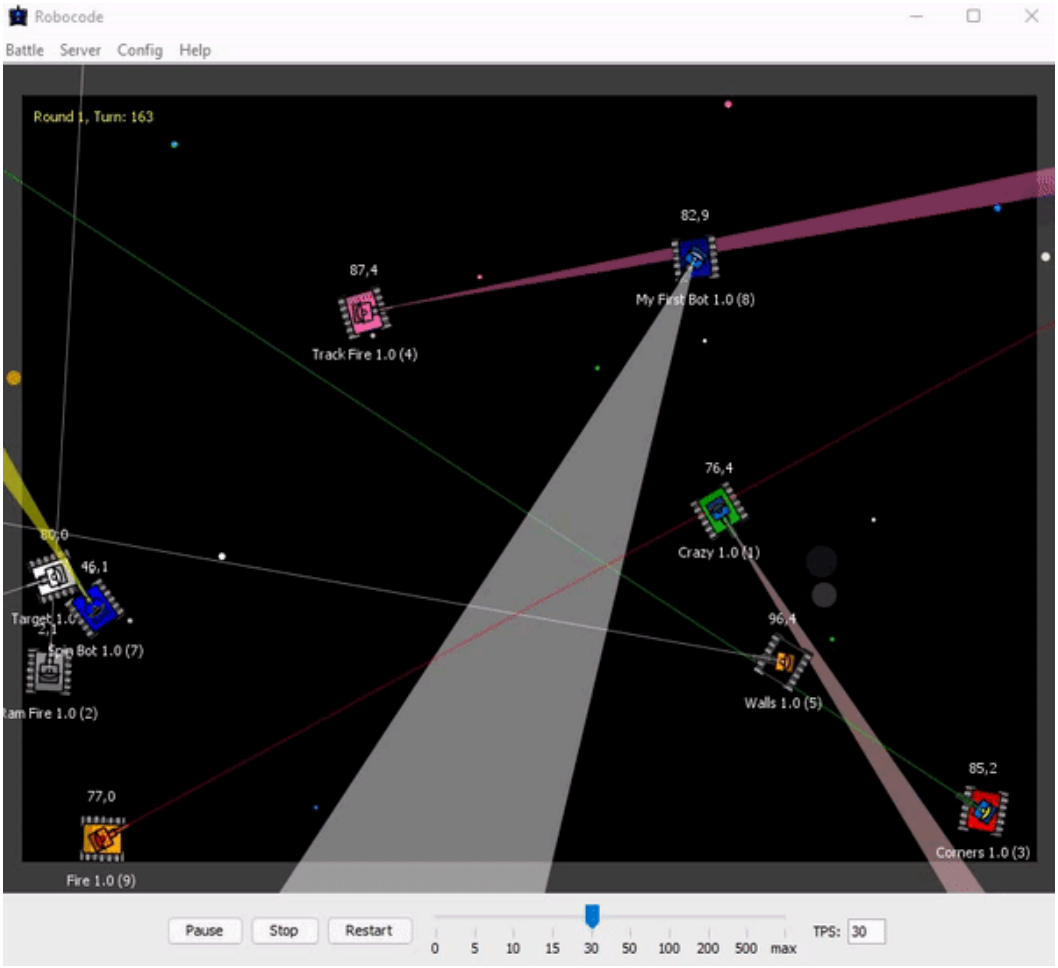
Muhammad Aufa Farabi (13523023)
Ferdinand Gabe Tua Sinaga (13523051)
Muhammad Iqbal Haidar (13523111)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

BAB I
DESKRIPSI MASALAH



Gambar 1 Robocode Tank Royale

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

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

Komponen-komponen dari permainan ini antara lain:

1. Rounds dan Turns

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

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

   Pada setiap turn, sebuah bot dapat:

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

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

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

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

2. Batas Waktu Giliran

   Penting untuk dicatat bahwa setiap bot memiliki batas waktu untuk setiap turn yang disebut turn timeout, biasanya antara 30-50 ms (dapat diatur sebagai aturan

pertempuran). Ini berarti bahwa bot tidak bisa mengambil waktu sebanyak yang mereka inginkan untuk bergerak dan menyelesaikan turn saat ini.

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

3. Energi

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

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

4. Peluru

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

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

5. Panas Meriam (Gun Heat)

   Saat menembakkan peluru, meriam akan menjadi panas. Peluru yang lebih berat menghasilkan lebih banyak panas dibandingkan peluru yang lebih ringan. Ketika meriam terlalu panas, bot tidak dapat menembak hingga suhu meriam turun ke nol. Selain itu,

meriam juga sudah dalam keadaan panas di awal round dan perlu waktu untuk mendingin sebelum bisa digunakan untuk pertama kalinya.

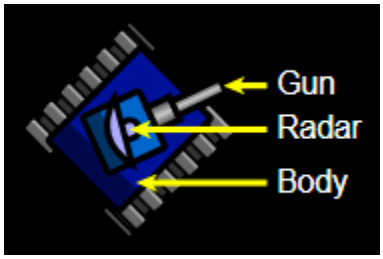
6. Tabrakan

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

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

7. Bagian Tubuh Tank

   Tubuh tank terdiri dari 3 bagian:

   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.

BAB II
LANDASAN TEORI

1. Konsep Algoritma Greedy

Algoritma Greedy adalah suatu algoritma dalam pemrograman yang memecahkan suatu persoalan optimasi dengan cara yang “rakus”, artinya memilih keputusan yang terbaik (optimum lokal) pada saat itu tanpa memperhatikan konsekuensi yang akan terjadi pada langkah selanjutnya. Pemilihan yang dilakukan pada suatu langkah sebagai optimum lokal diharapkan mampu mencapai optimum global pada langkah-langkah selanjutnya. Terdapat banyak contoh persoalan yang umumnya dapat diselesaikan dengan algoritma greedy, seperti pada persoalan knapsack dan meminimasi waktu dalam sistem. Pada persoalan meminimasi waktu dalam sistem, algoritma greedy akan mencoba menyusun waktu pelayanan pada sistem secara terurut dari yang terkecil.

Kelebihan dari algoritma greedy adalah eksekusi yang cepat dan efisien karena algoritma ini hanya meninjau pilihan-pilihan secara lokal tanpa perlu memikirkan langkah-langkah selanjutnya. Algoritma greedy akan sangat efisien apabila setiap langkah yang merupakan optimum lokal dapat mencapai optimum global. Namun, kelemahan dari algoritma greedy adalah solusi yang tidak selalu optimal karena pemilihan suatu keputusan yang didasarkan pada pencarian optimum lokal tanpa mempertimbangkan konsekuensi selanjutnya tidak selalu mencapai pada optimum lokal.

2. Elemen-Elemen Algoritma Greedy

Pada algoritma greedy, terdapat elemen-elemen yang membentuk komponen permasalahan yang ingin diselesaikan oleh algoritma greedy. Elemen-elemen tersebut antara lain

1. Himpunan Kandidat C, berisi kandidat yang akan dipilih pada setiap langkah, contohnya adalah job, koin, atau task.
2. Himpunan Solusi (S), berisi daftar kandidat yang sudah dipilih
3. Fungsi Solusi, fungsi yang menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi
4. Fungsi Seleksi, fungsi yang memilih kandidat berdasarkan strategy *greedy* tertentu
5. Fungsi Kelayakan, fungsi yang memeriksa apakah kandidat yang dipilih layak dimasukkan ke dalam himpunan solusi
6. Fungsi Obyektif (opsional), fungsi yang meminimumkan atau memaksimumkan

3. Cara Kerja Program

Robocode adalah sebuah game engine yang digunakan untuk mensimulasikan pertempuran antar robot virtual yang diprogram oleh pengguna. Dalam permainan ini, setiap bot dikendalikan oleh kode program yang menentukan bagaimana bot tersebut bergerak, menembak, serta mengambil keputusan dalam pertempuran. Saat game engine dijalankan, pengguna dapat booting kode sumber bot mereka agar dapat bergabung ke dalam permainan. Game engine bertanggung jawab atas segala aspek permainan, termasuk aturan, deteksi tabrakan, pengelolaan waktu eksekusi, serta tampilan visual pergerakan dan aksi bot.

Setiap bot dalam Robocode tidak dikendalikan secara langsung oleh pemain, melainkan melalui serangkaian instruksi pada kode sumbernya. Bot ini beroperasi dengan menggunakan API yang disediakan oleh Robocode untuk bergerak, menembak dan lain sebagainya. Bot dalam Robocode menjalankan aksinya dalam sistem berbasis "turn" atau giliran. Setiap bot diberikan jatah waktu eksekusi pada setiap turn untuk menjalankan kode programnya. Jika kode bot terlalu kompleks atau membutuhkan terlalu banyak perhitungan, bot bisa mengalami penalti waktu atau kehilangan kesempatan untuk bergerak dalam satu turn.

Untuk mengimplementasikan strategi greedy dalam bot, pendekatan yang umum digunakan adalah dengan selalu mengambil tindakan yang dianggap paling menguntungkan pada setiap langkah permainan. Misalnya, bot dapat diprogram untuk selalu menargetkan musuh terdekat, menyerang lawan dengan energi terendah terlebih dahulu, atau bergerak ke posisi yang dianggap paling aman berdasarkan informasi yang tersedia. Strategi greedy ini memungkinkan bot untuk membuat keputusan cepat tanpa perlu melakukan perhitungan kompleks jangka panjang.

BAB III
APLIKASI STRATEGI GREEDY

1. Mapping Persoalan

- **Himpunan Kandidat** dari persoalan ini adalah **semua kemungkinan urutan perilaku** robot yang dapat digunakan selama permainan.
- **Himpunan Solusi** dari persoalan ini adalah **urutan perilaku** dari robot selama pertempuran dari awal sampai akhir sedemikian sehingga skor akhir robot menjadi maksimum.
- **Fungsi Solusi** dari persoalan ini adalah **memeriksa apakah himpunan solusi memberikan skor akhir paling tinggi** diantara robot yang lain.
- **Fungsi Seleksi** dari persoalan ini adalah **algoritma pengambilan keputusan** yang menentukan perilaku robot selama permainan.
- **Fungsi Kelayakan** dari persoalan ini adalah **memeriksa apakah** perilaku robot yang digunakan merupakan **perilaku valid** yang diperbolehkan dalam permainan.
- **Fungsi Objektif** dari persoalan ini adalah untuk **memaksimumkan total score** akhir dari pertempuran.

2. Eksplorasi Alternatif Solusi

1. Coward

Pendekatan heuristik dari algoritma ini adalah dengan menghindari lokasi dimana musuh berada selama pertempuran dengan begitu bot akan memiliki kesempatan survival yang besar tiap rondenya. Harapannya skor survival akhir dari bot tersebut akan menjadi maksimum dan mendapatkan last survival bonus.

2. Kamikaze

Algoritma ini dinamai Kamikaze karena algoritma ini berusaha untuk terus menabrak sembari menembak bot yang ditargetkannya tanpa mempedulikan *energy* dari bot sendiri. Hal yang dilakukan pada algoritma Kamikaze adalah mengidentifikasi bot musuh dengan *energy* paling besar (*greedy by highest energy)* kemudian bot akan mengejar atau meng-*lock-in* (mengunci target) bot musuh yang terdeteksi dan mencoba menabraknya hingga berhasil dihancurkan. Strategi ini difokuskan untuk mendapatkan *ram damage* dan *ram damage bonus* sebanyak-banyaknya dari hasil tabrakan dengan musuh yang memiliki *energy* terbesar secara *lock-in* tanpa memperdulikan kondisi *energy* bot itu sendiri.

3. ThirdParty

Seperti namanya algoritma ini memanfaatkan pertempuran bot lain sebagai keuntungan baginya. Bot akan terlebih dahulu memindai area pertempuran dan mengidentifikasi petak dengan jumlah bot terbanyak. Setelah

menemukan petak dengan jumlah bot terbanyak, bot akan mencari target yang paling dekat dengan posisinya dalam petak tersebut dan mengarahkan tembakannya ke target tersebut. Dengan strategi ini, bot diharapkan dapat menembak musuh dengan akurasi yang lebih tinggi dan mendapatkan bonus point yang besar. Selain itu, dengan memanfaatkan pertempuran yang terjadi dalam satu petak hal ini dapat mempercepat kematian salah satu bot di petak tersebut dan meningkatkan peluang hidup dirinya sendiri.

4. Nearest

Algoritma dari bot ini cukup sederhana dengan pendekatan heuristik bahwa peluru memiliki peluang lebih besar untuk mengenai target dengan jarak musuh yang tidak terlalu jauh. Berdasarkan itu maka bot dirancang untuk selalu menembak bot musuh dengan jarak yang paling dekat terhadapnya. Harapannya bot akan mendapatkan banyak bullet score dan bullet score bonus.

3. Analisis Efisiensi dan Efektivitas

1. Coward

Algoritma Bot Coward

1. Lakukan scan terhadap setiap bot musuh
2. Untuk setiap bot musuh, cari panjang dan arah vektor relatif terhadap posisi bot sedemikian sehingga panjang vektor berbanding terbalik dengan jarak bot musuh
3. Cari resultan dari seluruh vektor tersebut lalu cari vektor berlawanannya
4. Cari koordinat tujuan sedemikian sehingga koordinat tersebut dilintasi oleh vektor berlawanan dan masih berada dalam area permainan
5. Arahkan bot untuk bergerak menuju koordinat tujuan tersebut
6. Selama melakukan pergerakan, bot terus melakukan scan dan dimungkinkan untuk mengubah koordinat tujuan guna menyesuaikan dengan posisi bot musuh
7. Bot tidak akan melakukan tembakan kepada bot musuh kecuali ketika ditabrak musuh atau musuh tersisa satu

Berdasarkan uraian langkah-langkah tersebut, dapat dianalisis efisiensi dari algoritma yakni kompleksitas waktunya yaitu O(n) sebab pada algoritma tersebut dilakukan iterasi satu persatu terhadap seluruh bot musuh yang memakan waktu linear berbanding lurus dengan jumlah musuh.

Apabila dianalisis efektivitasnya, algoritma ini belum dapat memastikan solusi yang diberikan selalu optimal. Salah satu contoh kasus counterexample nya adalah ketika musuh tinggal satu bot, maka permainan akan menjadi 1v1 dimana bot dengan algoritma duel yang lebih canggih tentu memiliki peluang yang lebih

besar untuk menang ketimbang bot coward yang dioptimalisasi untuk pertarungan lebih dari 2 pemain. Akibatnya bot tersebut akan mendapatkan skor last survival bonus, bullet damage, dan bullet damage bonus sehingga skor akhir akan lebih tinggi ketimbang bot coward.

2. Kamikaze

Algoritma Bot Kamikaze

1. Bot hanya bergerak ke tengah pada 60 turn pertama untuk mencari posisi yang strategis. Setelah 60 turn, Bot akan mulai melakukan scanning dengan radar.
2. Dilakukan pencarian bot dengan *energy* terbesar dengan syarat tidak melebihi 90 *energy* memakai scanning dan juga menginisialisasi *maximum energy* awal.
3. Ketika bot terdeteksi pada scan, bot tersebut diperiksa apakah memiliki *energy* yang lebih besar daripada bot sebelumnya. Bot yang memiliki *energy* yang lebih besar disimpan ID beserta *energy* nya untuk pengecekan berikutnya hingga semua bot telah diperiksa.
4. Apabila semua bot musuh telah diperiksa jumlah *energy*-nya, dilakukan scanning untuk mencari Bot musuh dengan ID yang sesuai dengan ID bot yang memiliki *energy* maksimum. Bot musuh yang diidentifikasi sama dengan bot tersebut akan di-*lock-in* sebagai target tabrakan kemudian bot mulai diarahkan menuju bot musuh yang telah di-*lock-in*.
5. Bot digerakkan menuju bot musuh hingga menabraknya. Apabila pada scan selanjutnya ID dari bot musuh yang terdeteksi tidak sama dengan ID bot yang di-*lock-in* maka bot musuh tersebut dihiraukan.
6. Pada proses menuju tabrakan, bot akan melakukan tembakan kepada bot musuh yang di-*lock-in* dengan damage yang bervariasi tergantung pada jaraknya. Jika selama proses tersebut ada bot lain yang menabrak bot maka bot akan mundur dan mereset target bot musuh.
7. Bot menabrak bot musuh dan akan terus menempel pada bot musuh sembari menembaknya hingga bot musuh dikalahkan.
8. Setelah bot musuh dihancurkan, bot akan mundur dan melakukan scan ulang untuk menargetkan bot musuh selanjutnya seperti pada langkah 1 - 5.

Pada Algoritma Bot Kamikaze, kompleksitas waktu Big-O yang telah dianalisis adalah O(n). Hal ini dapat dijelaskan dengan langkah-langkah yang dilakukan algoritma pada setiap turn pada *loop* permainan, seperti pengecekan bot dengan *energy* terbesar dan mekanisme *lock-in* pada bot musuh, dilakukan secara berurutan dan tanpa memiliki *loop* di dalamnya.

Apabila ditinjau dari tingkat efektifitasnya, algoritma Bot Kamikaze dapat memberikan solusi yang cukup optimal berdasarkan pengujian yang dilakukan

karena *ram damage* yang dihasilkan mampu menopang skor akhir yang tinggi. Bot dengan algoritma Kamikaze akan sangat efektif melawan bot yang tidak mempunyai pertahanan ketika ditabrak atau dalam area permainan yang tidak terlalu aktif pertempurannya. Namun untuk kasus yang lebih kompleks, algoritma ini akan kurang optimal karena bot yang menggunakan algoritma ini memiliki kesempatan *survival* yang rendah jika berada di area dengan pertempuran yang tinggi dan sulit melawan bot yang bertipe serupa (*ramming bot)*. Hal ini akan jauh merugikan bot dengan strategi kamikaze apabila bot yang dimainkan pada permainan berjumlah banyak. Selain itu, bot musuh yang ditarget dengan jarak yang jauh akan berisiko pada terjadinya tembakan terhadap bot dalam proses menuju tabrakannya.

3. ThirdParty

Algoritma Bot ThirdParty

1. Lakukan scan terhadap bot musuh dan petakan posisi mereka ke sebuah zona
2. Berdasarkan hasil scanning tersebut, cari zona dengan jumlah bot terbanyak
3. Jika ada maka cari bot yang posisinya terdekat dengan kita tapi masih dalam lingkup zona bot terbanyak.
4. Jika ternyata seluruh bot tersebar merata, ia cukup cari bot dengan zona terdekat dengan dirinya
5. Setelah mengetahui ingin menembak bot dari zona mana sekarang bot harus mencari berapa sudut yang dibentuk antara dirinya dengan bot target.
6. Selanjutnya, bot akan mengarahkan meriamnya ke bot tersebut dan menembakkan pelurunya.
9. Setelah proses 1 hingga 6 dipenuhi bot akan kembali meninjau situasi terkini dari bot-bot yang ada disekitarnya dan mengulangi langkah 1 - 6 sampai dirinya menang atau dieliminasi bot lain.

Dari algoritma yang telah diuraikan, dapat disimpulkan bahwa kompleksitas waktu algoritma berjalan secara linear (O(n)). Hal ini terlihat dari fakta bahwa untuk setiap n iterasi, algoritma menjalankan loop sebanyak 2*9, di mana 2*9 ini berasal dari dua langkah utama:

1. Pengecekan zona dengan konsentrasi bot tertinggi
2. Pengecekan bot mana yang paling dekat dengan dirinya dalam zona tersebut.

Kedua langkah ini dilakukan secara berurutan dan tidak bersarang, sehingga kompleksitasnya tetap linear terhadap jumlah bot (n). Dengan demikian, notasi Big-O dari algoritma ini adalah O(n).

Namun, jika ditinjau dari efektivitasnya, algoritma ini memiliki kelemahan dalam situasi di mana bot dikejar oleh bot lain. Dalam kondisi seperti ini, algoritma mungkin kurang optimal karena bot memerlukan waktu untuk mengatur meriamnya, sehingga memberikan kesempatan bagi bot lain untuk menabrak atau menembaknya. Tabrakan atau tembakan dari bot lain dapat memberikan bonus damage yang lebih besar daripada sekadar menembak balik bot yang mengejar.

4. Nearest

Algoritma Bot Nearest

1. Scan seluruh posisi bot musuh
2. Cari posisi musuh terdekat dan tembak musuh tersebut
3. Setelah melepaskan tembakan, bot bergerak ke arah random
4. Ulangi langkah 1-3 sampai permainan selesai/bot mati

Berdasarkan uraian langkah-langkah tersebut, dapat dianalisis efisiensi dari algoritma yakni kompleksitas waktunya yaitu O(n) sebab pada algoritma tersebut dilakukan iterasi satu persatu terhadap seluruh bot musuh untuk mencari posisi musuh terdekat yang memakan waktu linear berbanding lurus dengan jumlah musuh.

Apabila dianalisis efektivitasnya, algoritma ini belum dapat memberikan solusi optimal dari permainan. Salah satu contoh skenarionya adalah ketika bot dikepung oleh lebih dari dua bot musuh dengan jarak yang sama maka bot terlihat "bingung" menentukan targetnya. Padahal seharusnya apabila posisi berada dalam kepungan maka prioritas utamanya adalah menyelamatkan diri. Akibatnya bot terbunuh oleh musuh lain dan tidak mendapatkan skor akhir tertinggi.

4. Strategi Greedy Yang Dipilih

Kelompok kami memilih KamikazeBot sebagai bot utama kami sebab berdasarkan analisis teoritis di atas, KamikazeBot yang memiliki peluang paling besar untuk mendapatkan skor akhir tertinggi sebab mampu menghasilkan *ram damage* sekaligus *bullet damage* yang besar. Ram damage dan bullet damage sendiri menurut kelompok kami adalah penentu kemenangan dalam permainan robocode karena komponen skor tersebut memiliki pengaruh yang besar terhadap skor akhir dibandingkan skor survival dan dari semua strategi yang dibuat hanya strategi Kamikaze yang memiliki kemampuan untuk menghasilkan *ram damage* dan *bullet damage* yang besar sekaligus.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

1. Coward

Pseudocode

```
procedure CowardBot()
{ Menjalankan robot dengan menghindari musuh dan menembak jika
memungkinkan }

Deklarasi:
    type koordinat:
      <x : float,
       y : float>
    type vektor:
      <p : float,
       a : float>

    e  : list of ScannedBotEvent { himpunan musuh yang terdeteksi }
    x, m  : koordinat  { posisi kandidat untuk dituju }
    v, t  : vektor
    jumlahMusuh : integer

Algoritma:
    e ← {}  { inisialisasi himpunan musuh kosong }

    while (isRunning) do
      e ← e ∪ SCAN_BOT()
      jumlahMusuh ← length(e)

      for i ← 0 to jumlahMusuh do
          t ← HITUNG_VEKTOR(e[i]) { Membuat vektor dari ScannedBotEvent }
          v ← v + t { Menghitung resultan vektor }

      v ← HITUNG_VEKTOR_BERLAWANAN(v)
      x ← CARI_KOORDINAT_DILINTASI_VEKTOR(v)

      GoTo(x) { Memerintahkan bot menuju x }

      if (isRammed) then
          m ← CARI_KOORDINAT_PENABRAK() { Mencari koordinat penabrak }
          Tembak(m) { Menembak bot musuh m }
      endif

      if (jumlahMusuh < 2) then
          m ← CARI_KOORDINAT_MUSUH(e[0]) { Mencari musuh terakhir }
          Tembak(m)
      endif

    endwhile
```

2. Kamikaze

Pseudocode

```
Procedure KamikazeBot()
{ Menjalankan robot dengan menabrak musuh yang memiliki energy
terbesar secara lock-in }

Deklarasi:
    type koordinat:
      <x : float,
       y : float>
    scannedBots : Dictionary of Bot ID and scanned status {Dictionary
    berisi bot id dengan status scan nya)
    currTurn = integer {Jumlah turn selama game berjalan}
    LastScannedTurn : integer {Jumlah turn yang mengalami event
    onscanned)
    TargetID  : integer  { ID musuh yang di lock-in }
    TargetX, TargetY : koordinat{ koordinat target }
    IsTempEscape : boolean  { status robot ketika menghindar}
    Banzai : boolean { status mode lock-in pada bot musuh }
     HitByBot : integer  { jumlah tabrakan dengan musuh dalam satu
     turn }
    HighestEnergy : integer {Nilai energy maksimal saat ini}


Algoritma:
    currTurn ← 0
    LastScannedTurn ← 0
    TargetID ←  -1 { inisialisasi targetID awal }
    TargetX ← 0
    TargetY ← 0
    HitByBot ← 0
    HighestEnergy ← -1
    IsTempEscape ← false
    Banzai ← false

    while (isRunning) do
        currTurn ← currTurn + 1

        {Bergerak ke tengah untuk 60 turn pertama dan selanjutnya
        bergerak melingkar}
        DefaultMove()

        if (TargetID = -1) or not(Banzai) then
            setTurnRadarRight(45)
        else if (Banzai) then
            If ((currTurn - LastScannedTurn) > 5) then
                SetTurnRadarRight(360)
            endif
        endif

        {Pada satu turn OnScanned dapat terjadi lebih dari sekali}
        if (OnScanned) then
```

```
            If not(Banzai) and (currTurn > 60)  then
                If not (scannedBots.ContainsKey(e.scannedBotId) then
                    scannedbots[e.ScannedBotId] ← true
                Endif
                allBotsScanned ← (scannedbOts.Count == EnemyCount)
            If ((TargetID = -1) and e.Energy < 90) or ((e.Energy >
            HighestEnergy) and (e.Energy < 90)) then
                If (e.Energy > HighestEnergy) then
                    HighestEnergy ← e.Energy
                endif
                {Lock pilihan target sementara}
                LockTarget()
            endif
            else if ((targetID = e.ScannedBotId) and allBotscanned)
            and (EnemyCount = 1) then
                If (EnemyCount = 1) then
                    LockTarget(e)
                endif
                Banzai ← true
                SetTurnRadarRight(360)
            endif

            if (e.ScannedBotId = TargetID) and (Banzai = True) then
                TargetX ← e.X {lock-in pada koordinat bot yang
                dipilih}
                TargetY ← e.Y
                distance ← DistanceTo(e.X, e.Y)
                radarTurn ← RadarBearingTo(e.X, e.Y)
                SetTurnRadarLeft(radarTurn * 2)
                TurnToFaceTarget(TargetX, TargetY)
                TurnGunToFaceTarget(TargetX, TargetY)
                SetForward(distance + 5)

                FireByDistance()
                Go()
            endif
        endif

        if (OnHit) then
            HitByBot ← HitByBot + 1
            {Kondisi bot harus menghindar}
            if not (isTempEscape) then
                 if  ((HitByBot  > 1) and (DistanceTo(TargetX,
                 TargetY) > 80)) or (DistanceTo(TargetX, TargetY) then
                      IsTempEscape ← true
                      Back (80)
                      SetTurnLeft(60)
                      SetTurnRight(3600)
            {Kondisi bot menabrak musuh yang ditarget}
                 else
                      TargetX ← e.X
                      TargetY ← e.Y
                      radarTurn ← RadarBearingTo(e.X, e.Y)
```

```
                    SetTurnRadarLeft(radarTurn * 2)
                    TurnToFaceTarget(TargetX, TargetY)
                    TurnGunToFaceTarget(TargetX, TargetY)

                    FireByEnergy(e)
                    SetForward(40)
                    Go()
                Endif
                {Lanjut ke turn selanjutnya}
                ClearEvents()
            endif
        endif
         IsTempEscape ← false
         HitByBot ← 0
    endwhile
```

3. ThirdParty

Pseudocode

```
procedure ThirdPartyBot()
{ Menjalankan robot dan menembak dengan memilih area dengan konsentrasi
bot tertinggi atau bot pada area yang dekat dengannya}

Deklarasi:
    type Zone:
      <a : int,
       b : int>
    type location:
      <x : double,
       y : double>

    type playerZone:
      <key : Zone,
      <Value : list of location>

    player  : list of playerZone { himpunan musuh yang terdeteksi }
    TurnNumber: integer {Jumlah turn selama game berjalan}
    gridWidthScaling : integer {Skala pembagian lebar arena}
    gridHeightScaling : integer {Skala pembagian tinggi arena}

Algoritma:
    player ← {}  { inisialisasi himpunan musuh kosong }
    player ← player ∪ scan_bot() {Mengisi himpunan}
    gridWidthScaling ← 3
    gridHeightScaling ← 3

    while (isRunning) do
      if(isNearWall()) then
        AvoidWall()
      endif

      if (TurnNumber mod 8 = 0) then
        PlayerInZone ← [] {Inisialisasi List of location}
```

```
        count ← 0  { Inisialisasi jumlah musuh terbanyak }
        nearestZone ← (0,0){ Variabel untuk menyimpan zona terdekat }
        minDistance ← ∞  { Inisialisasi jarak minimum }

        {Mencari zona dengan jumlah musuh terbanyak dan paling dekat}

        for i ← 0 to 9 do
            playerCount ← jumlah(player[i].Value)
            distance ← (player[i].key.a - X/gridWidthScaling )² +
                       (player[i].key.b - Y/gridHeightScaling )²

            if (playerCount > count) OR (playerCount = count AND
                distance < minDistance) then

                count ← playerCount
                minDistance ← distance
                nearestZone ← key
                PlayerInZone ← player[i].Value

            end if
        end for

        if (jumlah(PlayerInZone) > 0) then
            minDistancePlayer ← ∞
            playerX ← 0
            playerY ← 0

            { Menentukan musuh terdekat di dalam zona terpilih }
            for i ← 0 to jumlah(PlayerInZone) do
                distancePlayer ←(player[i].Value.x - X)²+
                                (player[i].Value.y - Y)²

                if (distancePlayer < minDistancePlayer) then
                    minDistancePlayer ← distancePlayer
                    playerX ← player[i].x
                    playerY ← player[i].y
                end if
            end for

            targetDirection ← CalcDegree(X, Y, playerX, playerY)
            deltaDirection ← CalcGunBearing(targetDirection)


            TurnGunLeft(deltaDirection)
        end if

        Fire(2)
        cleanUpData()
        SetTurnRadarRight(360)
      end if


      SetForward(20)
      Go()
    endwhile
End Procedure
```

4. **Nearest**

Pseudocode

```
procedure NearestBot()
{ Menjalankan robot dengan menembak musuh terdekat dan bergerak secara
random }

Deklarasi:
    e  : list of ScannedBotEvent { himpunan musuh yang terdeteksi }
    x  : ScannedBotEvent { Bot musuh dengan jarak terdekat }
    jumlahMusuh : integer

Algoritma:
    e ← {}  { inisialisasi himpunan musuh kosong }

    while (isRunning) do
      e ← e ∪ SCAN_BOT()
      jumlahMusuh ← length(e)

      if (jumlahMusuh > 0) then
          x ← e[0]
      endif

      for i ← 0 to jumlahMusuh do
          if (HITUNG_JARAK(x) > HITUNG_JARAK(e[i])) then
              x ← e[i] { Update bot musuh terdekat }
          endif

      Tembak(x) { Metode menembak kearah bot musuh x }
      MoveRandom(x) { Memerintahkan bot bergerak random }

    endwhile
```

Struktur Data, Fungsi, Prosedur

Berdasarkan hasil pengujian bot yang dipilih sebagai bot utama adalah **KamikazeBot** karena menghasilkan skor akhir kumulatif paling tinggi.

Variabel

Variabel yang digunakan pada algoritma bot Kamikaze adalah

- currTurn (integer), jumlah turn pada game
- LastScannedTurn (integer), jumlah turn sementara yang mengalami event onscanned
- HitByBot (integer), jumlah tabrakan yang terjadi dalam satu turn
- TargetID (integer), ID bot musuh yang menjadi target untuk tabrakan
- TargetX (double), posisi X bot musuh yang ditarget
- TargetY (double), posisi Y bot musuh yang ditarget
- HighestEnergy (double), menampung *energy* dari bot dengan *energy* maksimum

- Banzai (boolean), *state* untuk *lock-in* musuh
- IsTempEscape (boolean), *state* ketika bot sedang kabur atau menghindar
- scannedBots (dictionary of boolean), menyimpan kondisi apakah suatu bot telah di-scan

Prosedur

Prosedur yang dipakai pada algoritma ini di antaranya adalah

- **Public override void Run()**
  Prosedur untuk menjalankan program
- **Public override void OnScanned(ScannedBot e)**
- Prosedur *event handler* yang dilakukan ketika bot berhasil meng-scan musuh
- **Public override void OnHitBot(HitBotEvent e)**
  Prosedur *event handler* yang dilakukan ketika bot menabrak atau ditabrak oleh musuh
- **Public override void OnBotDeath(BotDeathEvent e)**
  Prosedur *event handler*yang dilakukan ketika bot berhasil menghancurkan bot musuh
- **public override void OnHitWall(HitWallEvent e)**
  Prosedur *event handler* yang dilakukan apabila bot mengenai tembok area
- **public override void OnTick(TickEvent tickEvent)**
  Prosedur *event handler* yang terjadi ketika ada tick event
- **private void TurnToFaceTarget(double x, double y)**
  Prosedur untuk mengarahkan body dari bot ke arah musuh yang ditarget
- **private void TurnGunToFaceTarget(double x, double y)**
  Prosedur untuk mengarahkan gun dari bot ke arah musuh yang ditarget
- **private void LockTarget(ScannedBotEvent e)**
  Prosedur yang mengunci suatu bot musuh sebagai musuh yang ingin ditabrak
- **private void ResetTarget()**
  Prosedur yang mereset target bot musuh yang dikunci
- **private void FireByEnergy(HitBotEvent e)**
  Prosedur yang menentukan besar Tembakan berdasarkan kondisi *energy* musuh
- **private void FireByDistance()**
  Prosedur yang menentukan besar Tembakan berdasarkan jarak antara bot dengan musuh
- **private void DefaultMove()**
  Prosedur ini berfungsi untuk menentukan gerak bot pada 60 turn pertama agar ia mencoba bergerak ke tengah.

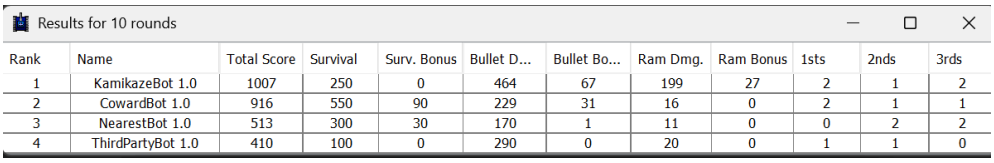
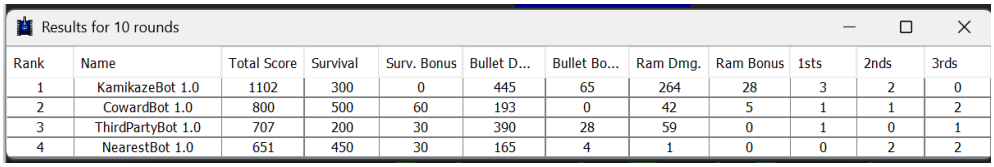
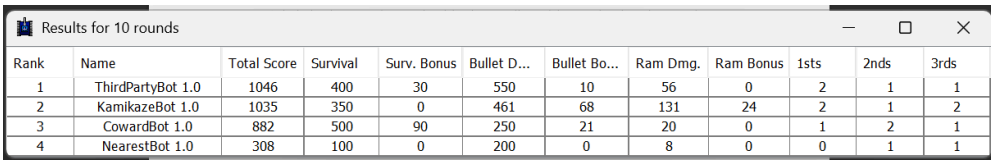
Fungsi

Tidak ada fungsi yang dibuat pada algoritma Kamikaze. Pada algoritma ini, hanya dipakai fungsi bawaan dari API game Robocode Tank Royale.

Struktur Data

Dalam bot ini struktur data utama yang dipakai adalah dictionary. Dictionary ini akan dimanfaatkan oleh bot untuk memastikan bahwa seluruh bot yang aktif dalam *battlefield* telah di scan sebelum scanning bot yang telah teridentifikasi memiliki energy tertinggi (dari ID botnya). Hal tersebut berguna demi memastikan bahwa bot yang di-*lock-in* sebagai target tabrakan lehnya adalah bot dengan energi tertinggi dan energi < 90 diantara semua bot yang masih hidup.

Pengujian

Pengujian	Hasil
1	
2	
3	

Analisis Pengujian

Berdasarkan hasil pengujian terhadap empat bot yang dibuat dalam permainan dengan format 1 vs 1 vs 1, dapat disimpulkan bahwa bot yang memakai strategi Kamikaze (KamikazeBot) memiliki peluang tertinggi untuk mendapatkan skor akhir tertinggi (solusi optimal) walaupun tidak selalu. Bot dengan algoritma Kamikaze memenangkan mayoritas tes pengujian karena bot didesain oleh algoritma untuk menabrak bot dengan energi terbesar (*greedy by highest energy*) yang di-*lock* secara agresif demi meraih skor ram damage beserta ram damage bonus sebanyak-banyaknya. Besarnya r*am damage* yang ditimbulkan menjadi penentu kemenangan bagi bot dengan strategi kamikaze. Algoritma Kamikaze sangat efisien dalam *1 vs 1* melawan bot yang tidak bertipe *ramming* karena algoritma ini juga mengombinasikan tabrakan dengan tembakan yang memiliki damage yang besar untuk membunuh musuh yang ditarget secara cepat.

Di sisi lain, algoritma Kamikaze yang *greedy* dalam mencari bot dengan energi terbesar demi mendapatkan ram damage telah *survivability* yang sebetulnya dapat menjadi nilai tambah bagi skor akhir. Seringkali, bot dengan algoritma Kamikaze mudah tereliminasi pada awal permainan akibat tembakan bot musuh lain ketika sedang menabrak bot musuh yang ditarget. Selain itu, algoritma Kamikaze tidak efisien pada kondisi berikut ketika berada di area dengan tingkat pertempuran yang tinggi atau bot yang ditabrak memiliki mekanisme pertahanan yang

baik. Bot dengan algoritma Kamikaze juga cukup rentan apabila target bot dengan energi terbesar yang di-*lock* berada pada jarak yang sangat jauh sehingga bot sangat berisiko mengalami tabrakan dengan bot musuh lain dan membuang-buang tembakan jarak jauh yang tidak akurat.

CowardBot belum bisa mendapatkan solusi optimal yakni skor akhir tertinggi untuk pertarungan Robocode salah satu penyebab utamanya adalah sifat alami bot yang terbilang pasif sehingga poin yang didapatkan akibat bullet damage dan ram damage sangat kecil ketimbang bot lain yang lebih agresif. Namun bot ini memiliki kelebihan ketika dihadapkan dengan pertempuran yang melibatkan banyak bot musuh sebab strategi *CowardBot* membuat peluang bot untuk selalu survive sampai musuh tinggal satu cukup besar. Salah satu perbaikan yang dapat dilakukan untuk bot ini supaya bisa mendapatkan skor akhir yang lebih baik adalah dengan memberikan prosedur tambahan yang selalu menembak musuh selama permainan dengan catatan tembakan yang dilepaskan harus akurat. Hal tersebut tentu akan menambah poin *bullet damage* kepada *CowardBot*.

Sementara itu, algoritma *Third Party* juga menunjukkan kurangnya efisiensi dalam permainan *Robocode*. Strategi utama bot ini adalah menembak ke area dengan kepadatan musuh tertinggi, namun keterbatasan dalam melakukan *scan* secara *real-time* menjadi kelemahan utama. Karena data yang diperoleh memiliki *delay*, sering kali posisi musuh sudah berubah sebelum peluru mencapai target, mengurangi efektivitas serangan. Kelemahan ini semakin diperparah dengan tidak digunakannya *predictive shooting*, sehingga dalam situasi *1 vs 1*, di mana musuh lebih aktif menghindar, bot ini menjadi kurang akurat dalam menyerang.

Selain masalah akurasi, bot dengan algoritma Third Party juga memiliki kelemahan dalam hal *survivability*. Karena fokusnya hanya pada menembak ke arah kerumunan lawan, bot ini menjadi rentan ketika ada musuh yang secara aktif mengejarnya. Hal ini terlihat dalam pengujian pertama, di mana bot Third Party lebih mudah dieliminasi karena tidak memiliki mekanisme bertahan yang memadai. Namun, meskipun memiliki akurasi yang kurang baik dalam duel *1 vs 1*, performa secara keseluruhan masih cukup baik, terutama dalam skenario di mana banyak musuh berkumpul. Pada pengujian kedua dan ketiga, bot ini terbukti mampu memberikan *bullet damage* yang sebanding dengan KamikazeBot karena kecenderungan strateginya yang menargetkan kelompok musuh, meningkatkan peluang tembakan mengenai target, meskipun bukan target utama yang sebenarnya dituju.

NearestBot belum bisa mendapatkan solusi optimal untuk setiap pertempuran. Beberapa faktor penyebabnya adalah pergerakan bot yang masih sering menabrak dinding serta algoritma penembakan yang masih belum akurat sehingga hal tersebut membuang-buang *energy* dari *NearestBot*. Selain itu, algoritma yang terbilang cukup naif ini tentu saja mudah untuk diatasi oleh bot musuh dengan algoritma yang lebih canggih. Namun, kelemahan tersebut juga dapat menjadi kelebihan dari bot ini, dengan algoritma yang cenderung sederhana maka orang awam pun dapat mengerti maksud dan tujuan bot ini. Secara keseluruhan, bot ini hampir tidak pernah menang melawan bot manapun terkecuali bot template ataupun bot sample.

BAB V
KESIMPULAN DAN SARAN

KESIMPULAN

Pendekatan strategi greedy dalam pembuatan algoritma Robocode tidak selalu menghasilkan solusi yang optimal. Meskipun demikian, algoritma ini memiliki beberapa kelebihan, seperti kemudahan dalam perancangan dan kompleksitas yang relatif rendah, sehingga mudah dipahami dan diimplementasikan.

Dalam praktiknya, pendekatan greedy dapat menghasilkan solusi suboptimal, karena keputusan yang diambil hanya mempertimbangkan keuntungan jangka pendek tanpa memperhatikan dampaknya di masa depan. Berdasarkan hasil pengujian yang dilakukan, tidak ada strategi algoritma *greedy* pada bot yang menghasilkan solusi global optimal karena setiap strategi *greedy* memiliki kelebihan dan kekurangannya masing-masing. Oleh karena itu, dalam penyelesaian kasus persoalan optimasi, pemilihan algoritma *greedy* sebagai algoritma penyelesaian belum tentu dapat menjadi jalan keluar untuk mencapai solusi yang optimal. Namun, untuk beberapa aplikasi di dunia nyata, solusi suboptimal ini sering kali sudah cukup untuk memenuhi kebutuhan, terutama dalam skenario yang menuntut keputusan cepat dan efisien.

SARAN

Untuk kedepannya, strategi bot yang sekarang digunakan dapat ditingkatkan lagi dengan mempertimbangkan pendekatan-pendekatan heuristik lainnya atau bahkan dengan pembelajaran mesin dengan begitu algoritma bot akan memiliki peluang yang lebih besar menghasilkan solusi optimum untuk berbagai skenario pertempuran.

Lalu saran terkhusus untuk kelompok adalah untuk memperbaiki manajemen waktu serta komunikasi antar anggota kelompok. Manajemen waktu yang baik sangat penting agar setiap tugas dapat diselesaikan dengan lebih terstruktur, menghindari penumpukan pekerjaan di akhir tenggat waktu, dan memastikan setiap anggota memiliki waktu yang cukup untuk melakukan revisi jika diperlukan. Hal tersebut penting untuk dilakukan agar pengerjaan tugas-tugas dimasa yang akan datang dapat menghasilkan output yang lebih baik.

LAMPIRAN

Pranala

Github Repository:
https://github.com/AgungLucker/Tubes1_Septic-tank-1000-liter-biomif-Rp.1.500.000

Video Youtube: https://youtu.be/Rt_1_5x2ueE

Checklist

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

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/04-Algoritma-Greedy-(2025)-Bag1.pdf Diakses pada tanggal 23 Maret 2025

https://robocode-dev.github.io/tank-royale/ Diakses pada tanggal 23 Maret 2025

https://robowiki.net/ Diakses pada tanggal 23 Maret 2025