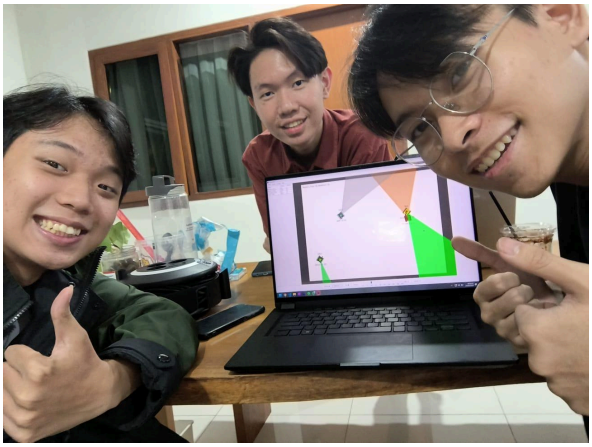
LAPORAN TUGAS BESAR 1

PEMANFAATAN ALGORITMA GREEDY DALAM PEMBUATAN BOT PERMAINAN ROBOCODE TANK ROYALE

Disusun untuk Memenuhi Tugas Besar 1 Mata Kuliah Strategi Algoritma IF2211

Dosen Pengampu:

Dr. Ir. Rinaldi Munir, M.T.



Ahmad Ibrahim	13523089
Karol Yangqian Poetracahya	13523093
Aria Judhistira	13523112

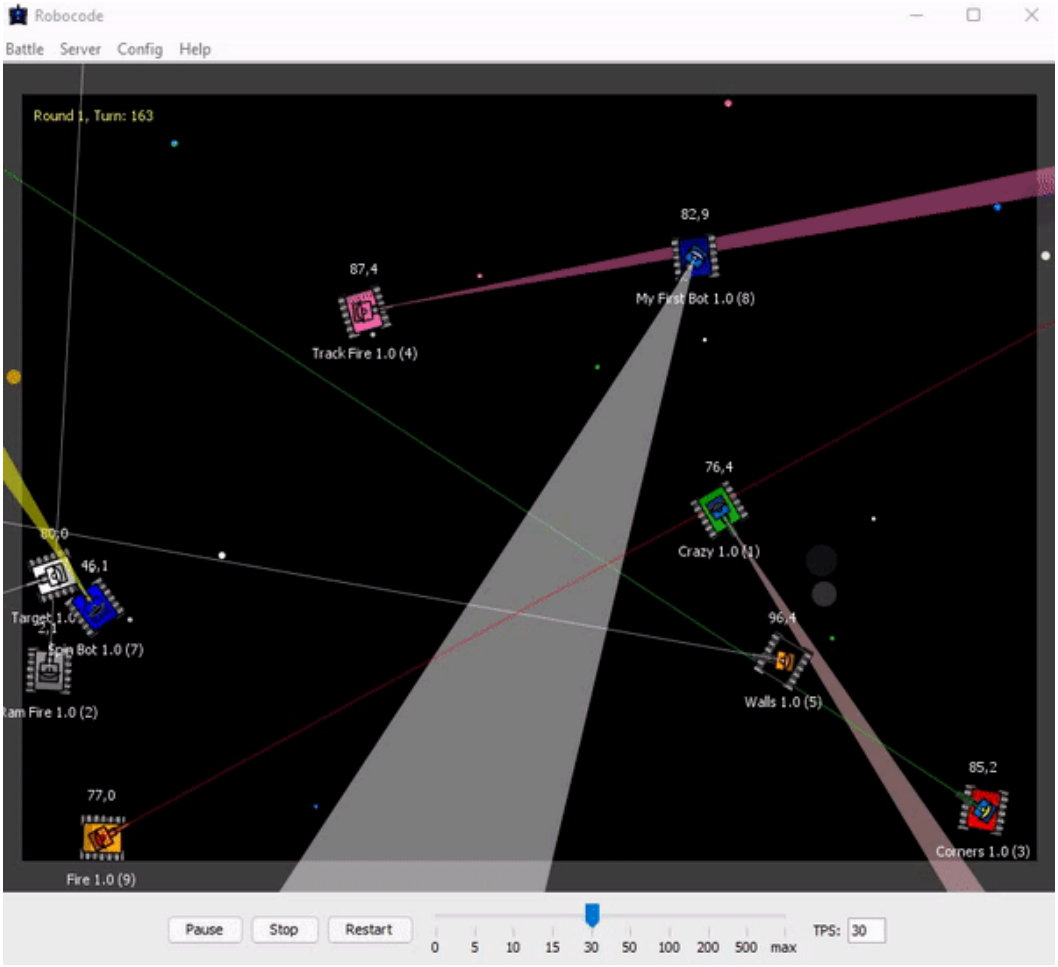
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

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.

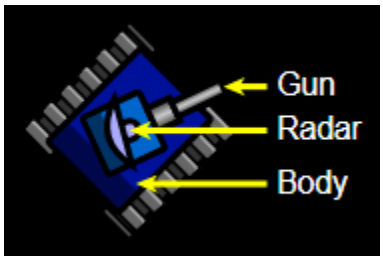
6. Tabrakan

   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.

Starter Pack

Untuk tugas besar ini, game engine yang akan digunakan sudah dimodifikasi oleh asisten. Berikut adalah beberapa perubahan yang dibuat oleh asisten dari game engine Tank Royale:

- **Theme GUI:** diubah menjadi light theme
- **Skor & Energi:** masing-masing bot ditampilkan di samping arena permainan agar lebih mudah diamati saat pertarungan
- **Turn Limit:** Durasi pertarungan tidak akan bergantung pada banyak ronde. Pada game engine ini, pertarungan akan berakhir apabila banyaknya turn sudah mencapai batas tertentu. Apabila batasan ini tercapai, ronde otomatis langsung berakhir dan pemenang pertarungan akan ditampilkan. Turn Limit dapat diatur pada menu “setup rules”.

Source Code untuk game engine dan template bot telah disediakan pada tautan berikut.

tubes1-if2211-starter-pack

Adapun panduan mengenai cara menjalankan game engine, membuat bot, dan melihat referensi API dapat dilihat melalui tautan berikut.

II. LANDASAN TEORI

2.1 Algoritma Greedy

Algoritma *greedy* merupakan salah satu algoritma yang paling populer dalam dunia pemrograman untuk memecahkan suatu permasalahan dan menghasilkan solusi yang optimal. Algoritma *greedy* diterapkan pada berbagai algoritma pemecahan masalah lainnya, seperti Algoritma Djikstra, Algoritma Prim dan Kruskal dalam mencari *Minimum Spanning Tree*, dan Pemampatan Kode Huffman. Sesuai dengan namanya, secara langkah per langkah, algoritma ini mengambil pilihan terbaik pada saat/tahap itu tanpa memerhatikan konsekuensi atau dampaknya untuk kedepannya. Algoritma *greedy* tidak merevisi kembali langkah yang sudah diambil dan bekerja secara *top-down*. Langkah-langkah optimum secara lokal ini diharapkan dapat mencapai solusi global yang optimal.

Algoritma Greedy memiliki beberapa elemen sebagai berikut.

- Himpunan Kandidat (C)

  Himpunan Kandidat berisi kandidat atau kemungkinan yang dapat dipilih pada suatu tahap persoalan.

- Himpunan Solusi (S)

  Himpunan Solusi berisi kandidat-kandidat dari Himpunan Kandidat yang dipilih dan dianggap merupakan pilihan terbaik "pada saat itu."

- Fungsi Solusi

  Fungsi Solusi menentukan apakah Himpunan Kandidat yang didapatkan mengandung sebuah solusi yang mengikuti prinsip *greedy* atau tidak.

- Fungsi Seleksi

  Fungsi Seleksi memilih kandidat pada Himpunan Kandidat berdasarkan strategi *greedy* tertentu yang bersifat heuristik.

- Fungsi Kelayakan

  Fungsi Kelayakan menentukan apakah kandidat yang dipilih layak atau tidak dimasukkan ke dalam Himpunan Solusi.

- Fungsi Obyektif

Fungsi Obyektif menentukan sifat memaksimumkan atau meminimumkan pada strategi *greedy* yang digunakan.

Meskipun strategi *greedy* memiliki prinsip yang mudah dipahami, strategi ini tidak selalu menghasilkan solusi yang terbaik atau paling optimal terhadap suatu persoalan. Hal ini disebabkan oleh sifatnya yang tidak bekerja secara menyeluruh terhadap semua kemungkinan seperti algoritma *brute force* dan hanya mengandalkan pilihannya "saat itu." Selain itu, suatu strategi *greedy* dapat memiliki beberapa Fungsi Seleksi berbeda yang sama-sama mengikuti prinsip *greedy* sehingga perlu jeli untuk memilih Fungsi Seleksi yang tepat.

2.2 Cara Kerja Program

Game engine Robocode digunakan untuk menjalankan bot yang sudah dibuat. Konfigurasi dilakukan pada *game engine* agar dapat melakukan *boot* pada bot yang sudah dibuat. Saat melakukan *boot*, *game engine* menjalankan file .cmd pada folder bot yang dipilih yang akan mengompilasi dan membangun bot. Setelah sukses melakukan *boot* robot yang diinginkan, robot dapat dimasukkan ke dalam permainan bersama dengan robot-robot lainnya. Selain itu, peraturan permainan dapat diatur juga oleh pemain sesuai dengan kebutuhan.

Dalam permainan, robot dapat melakukan tiga aksi utama, yakni bergerak, menggerakkan radar dan memindai lawan, serta menggerakkan *gun* dan menembak bot musuh. Aksi robot dapat dilakukan secara runtut dan mengulang (dengan memanfaatkan perulangan *IsRunning*) atau dilakukan secara berurutan pada setiap *tick* atau *turn* dengan fungsi *OnTickEvent*. Selain itu, ketiga aksi tersebut dapat diatur untuk bergerak secara independen dan dilakukan secara bersamaan antara satu komponen robot dengan lainnya menggunakan perintah *Set-*.

Pada proses perancangan robot, terdapat dua aspek yang menjadi fokus utama pengembangan untuk mendapatkan keunggulan dari robot lawan, yakni pada aspek pergerakan dan pada aspek penargetan. Acuan utama untuk aspek pergerakan adalah untuk mencari cara agar robot dapat menghindari *damage* sebisa mungkin untuk mempertahankan *energy* robot. Hal ini dapat dicapai melalui berbagai cara, seperti mengembangkan algoritma untuk menghindari peluru musuh, menjauhi robot musuh, dll. Di sisi lainnya, aspek penargetan fokus pada cara untuk mendapatkan skor terbanyak dari menembak dan mengeliminasikan lawan. Laporan ini akan mengeksplorasi beberapa pengembangan kedua aspek melalui paradigma berbeda.

Proses pengembangan dan implementasi kedua aspek robot tersebut tentu tidak melupakan prinsip-prinsip algoritma *greedy*. Dalam kasus permainan Robocode ini, fokus utama pada implementasi strategi *greedy* adalah pada cara untuk memaksimumkan skor yang diraih pada setiap ronde. Berdasarkan itu, terdapat tiga cara utama untuk mendapatkan skor, yakni dengan menembak dan mengenai musuh, bertahan hidup, dan melakukan *ramming* atau menabrak musuh dengan sengaja. Ketiga cara ini juga memiliki keadaan yang dapat menciptakan skor bonus dengan syarat yang berbeda. Laporan ini akan berisi eksplorasi terhadap beberapa cara dan paradigma berbeda untuk memaksimumkan skor yang dapat diraih oleh robot dengan tetap mengimplementasi dan memperhatikan strategi *greedy* pada prosesnya.

III. APLIKASI STRATEGI GREEDY

3.1. Mapping Persoalan Robocode Menjadi Elemen-Elemen Greedy

Aplikasi strategi *greedy* yang digunakan berupa pemilihan aksi berdasarkan nilai keuntungan (sebagai *heuristic value*) yang dihasilkan. Dengan metode yang didasari nilai *heuristic value*, implementasi *greedy* menjadi lebih fleksibel dan memudahkan dalam memodifikasi prioritas pemilihan aksi sebagai alternatif lain. Permainan Robocode dapat dianalisis sebagai sebuah permasalahan *greedy* dengan elemen-elemen seperti berikut.

a. Himpunan kandidat

   Himpunan kandidat dalam permainan Robocode adalah aksi-aksi yang dapat diambil oleh robot pemain berdasarkan API yang disediakan. Aksi sendiri dapat didefinisikan sebagai sebuah atau serangkaian perilaku robot selama permainan. Karena permainan Robocode menerapkan simulasi fisik yang cukup dinamis, tingkat abstraksi dari sebuah aksi menjadi sangat beragam, mulai dari yang sederhana seperti bergerak maju 10 unit atau menembak musuh pada lokasinya saat ini, hingga yang lebih kompleks seperti pergi ke sebuah sudut dan berosilasi di sana. Aksi dapat diimplementasikan menggunakan fungsi-fungsi yang disediakan seperti SetFire(), SetForward(), SetTurnGunLeft(), dan masih banyak lagi. Aksi juga dapat dilakukan dengan mengatur *member attribute* robot seperti TargetSpeed dan TurnRate yang mengubah perilaku robot tanpa pemanggilan fungsi secara eksplisit.

b. Himpunan solusi

   Himpunan solusi adalah aksi-aksi yang telah dilakukan robot sejauh ini. Karena waktu dalam permainan Robocode bergerak ke satu arah, robot tidak bisa kembali ke masa lalu untuk memeriksa kemungkinan solusi lain dari kandidat yang mungkin dapat memberikan konsekuensi yang lebih menguntungkan.

c. Fungsi solusi

   Solusi terakhir dari permainan Robocode adalah kemenangan yang diperoleh robot dengan poin sebesar mungkin. Pada dasarnya, fungsi solusi sudah diimplementasikan oleh *game engine* dengan sistem perolehan poin dan mekanisme *rounds*-nya untuk menentukan pemenang sehingga pemain cukup merancang fungsi seleksi dengan strategi *greedy* yang sebaik mungkin dengan harapan akan semakin dekat menuju solusi yakni kemenangan.

d. Fungsi seleksi

Fungsi seleksi adalah strategi pemilihan aksi yang akan dilakukan robot apabila diberikan informasi mengenai keadaan permainan saat ini. Fungsi seleksi yang baik dapat memberikan keuntungan maksimum pada saat ini. Keuntungan dikategorikan sebagai menambah atau mempertahankan energi robot, mengurangi energi robot lain, dan menghindari hal-hal yang mengakibatkan energi robot berkurang. Fungsi nilai keuntungan bergantung pada masing-masing aksi. Setiap kandidat aksi yang dapat diambil robot melalui fungsi seleksi memiliki konsekuensi di masa depan yang sangat acak dan bergantung pada *positioning*, strategi setiap pemain, jumlah robot, dan faktor acak lainnya. Konsekuensi ini pada hakikatnya tidak bisa ditelaah karena keacakannya dan linearitas waktu sehingga robot hanya dapat menggunakan fungsi seleksi untuk melakukan aksi yang sekiranya terbaik berdasarkan informasi yang dimiliki saat ini.

e. Fungsi kelayakan

Fungsi kelayakan dalam Robocode berperan untuk memastikan aksi yang dilakukan setiap robot dalam permainan sesuai dengan aturan dan hukum fisik yang berlaku. Beberapa contoh aturan tersebut adalah kecepatan sudut maksimum yang terbatas pada kelajuan robot melalui persamaan $10 - \frac{3}{4}|v|$, robot yang tidak boleh keluar arena, besar putaran radar yang terbatas pada setiap *tick*, dan lain-lain. Aturan-aturan ini telah diimplementasikan pada setiap fungsi API yang dapat dipanggil.

f. Fungsi objektif

Fungsi objektif digunakan saat melakukan seleksi kandidat aksi yang akan dipilih. Contoh penggunaannya adalah mencari robot musuh dengan jarak atau energi minimum/maksimum atau mencari sebuah titik dari sekumpulan pilihan titik di arena dengan jarak maksimum dari robot musuh untuk mencari lokasi paling aman saat ini.

3.2. Eksplorasi Alternatif Solusi Greedy yang Mungkin

Alternatif solusi *greedy* yang telah dieksplorasi adalah dengan menerapkan variasi fungsi *heuristic value* masing-masing aksi, sehingga mengubah prioritas-prioritas masing-masing aksi. Strategi *greedy* dapat diaplikasikan pada aspek pergerakan (*movement*) dan penargetan (*targeting*).

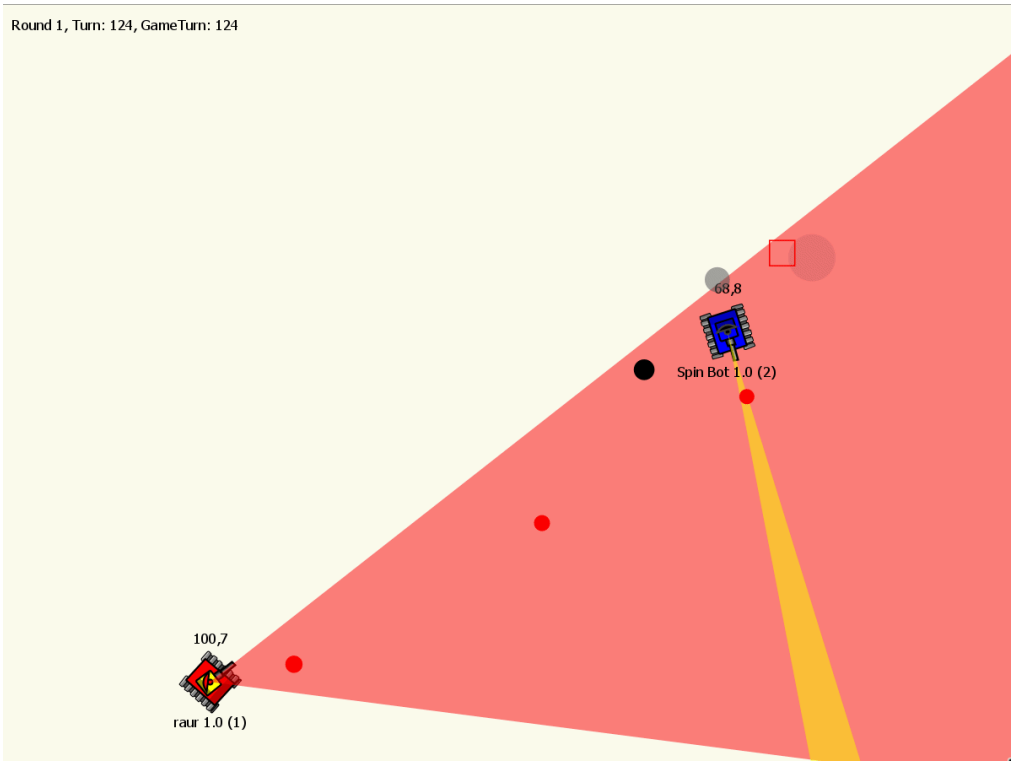
a. *Movement*

i. Ramming

Strategi ini melibatkan robot yang bergerak langsung menuju musuh dengan tujuan menabraknya, mengincar bonus ram dan akurasi yang meningkat. Namun, pendekatan ini berisiko tinggi karena robot juga dapat menerima kerusakan akibat tabrakan.

ii. Circle

Pendekatan ini menggunakan heuristik untuk menghindari peluru yang bersifat head-on dan linear dengan berputar secara konstan.



iii. Random

Robot bergerak dengan pola acak untuk menghindari tembakan musuh. Dengan tidak memiliki pola pergerakan yang jelas, musuh akan kesulitan memprediksi posisi robot Anda selanjutnya.

iv. Osilasi

Pendekatan ini menggunakan heuristik untuk menghindari peluru yang bersifat linear dengan bergerak maju dan mundur secara bergantian, menciptakan pola osilasi yang dapat membingungkan musuh dalam menargetkan.

v. Corner

Robot bergerak ke sudut arena dan bertahan di sana. Strategi ini dapat mengurangi kemungkinan diserang dari berbagai arah.

vi. Anti-Gravity

Robot bergerak menjauh dari titik-titik berbahaya, seperti posisi musuh atau peluru yang mendekat, seolah-olah ada gaya tolak-menolak. Pendekatan ini membantu robot menghindari area berbahaya di medan perang.

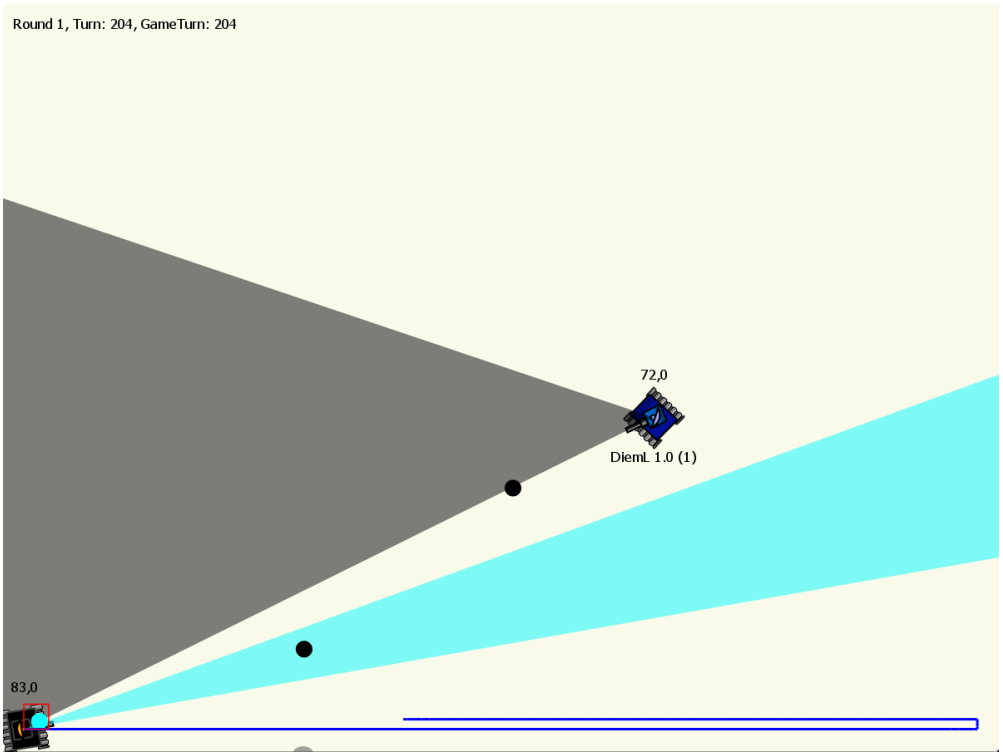
vii. Stop and Go

Robot bergerak setiap kali musuh menembak, hal itu dilakukan untuk meng-*counter* targeting Head-On, Linear, dan Circular. Namun, movement ini tidak cocok jika melawan musuh yang berjumlah lebih dari 1.

b. *Targeting*

i. Head-on

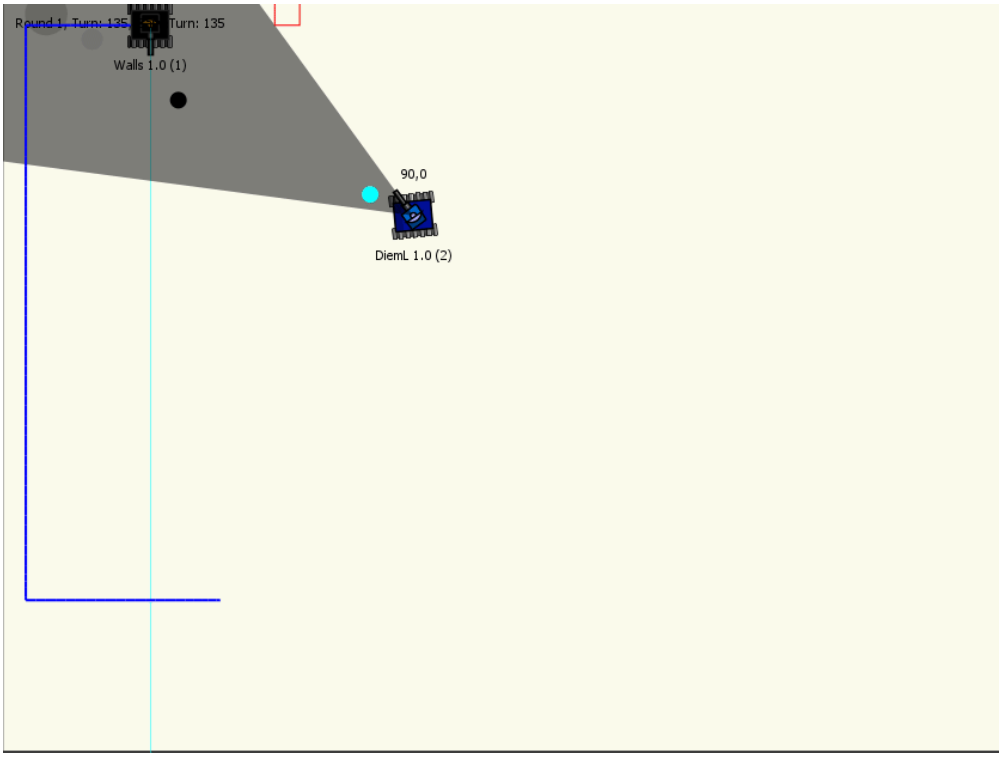
Robot menembak langsung ke arah posisi musuh saat ini tanpa mempertimbangkan pergerakan musuh. Strategi ini efektif untuk melawan robot yang bergerak secara osilasi atau diam namun kurang efektif jika musuh bergerak ke arah yang konstan.



ii. Linear

Robot memprediksi posisi masa depan musuh dengan mengasumsikan musuh akan terus bergerak dalam garis lurus dengan kecepatan konstan. Metode ini menggunakan solusi dari titik potong persamaan parametrik dua garis lurus untuk menentukan arah tembakan. Pendekatan ini akurat jika musuh bergerak

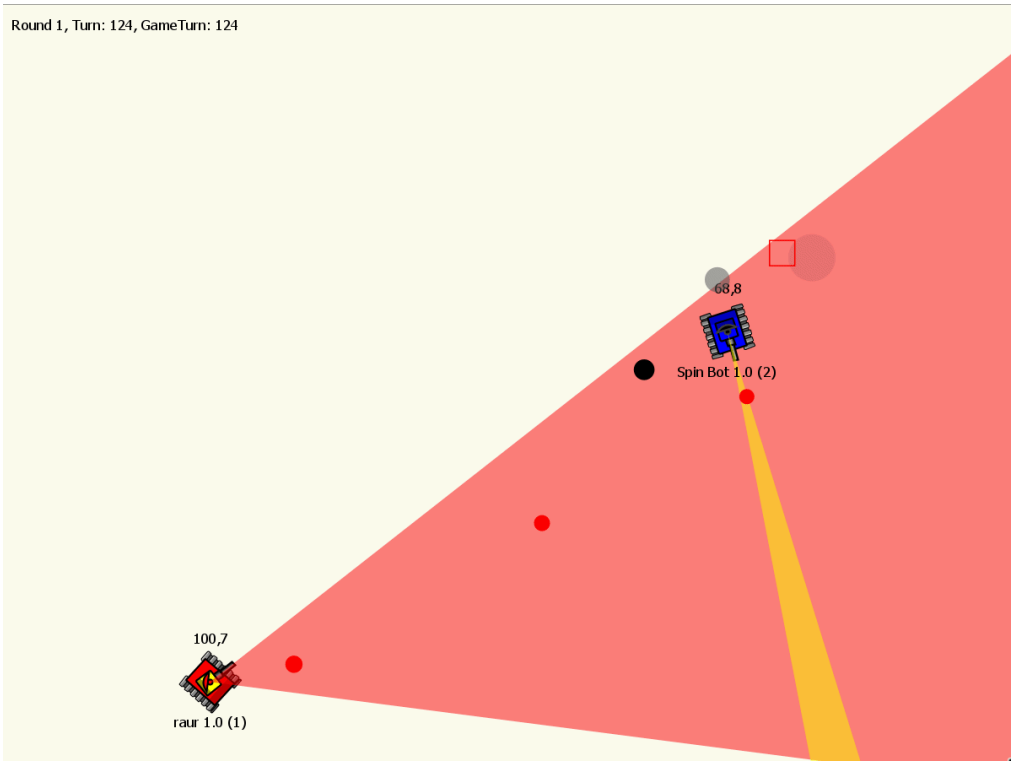
dengan pola linear.



iii. Circular

Robot memprediksi posisi masa depan musuh dengan mengasumsikan musuh bergerak dalam pola melingkar. Metode ini diimplementasikan dengan memperoleh informasi kecepatan sudut dan kecepatan tangensial musuh lalu mengaplikasikannya ke model persamaan gerak melingkar. Strategi ini efektif

jika musuh menggunakan pergerakan melingkar.



iv. Play It Forward

Robot memprediksi pergerakan musuh dengan mempelajari pola pergerakan sebelumnya dan berharap musuh mempertahankan pola pergerakan tersebut. Salah satu implementasi strategi ini adalah N-Gram yang melihat serangkaian pergerakan musuh sebelumnya lalu menentukan arah tembakan yang sesuai dengan pola pergerakan tersebut. Model ini berperan sebagai heuristik yang

Berdasarkan hasil eksplorasi dari berbagai strategi *greedy* yang dapat dipakai, 4 kombinasi strategi *movement* dan *targeting* diujikan dan diimplementasikan pada 4 robot yang berbeda, yakni Woff, Qwuck, Schmelly, dan Pffrrrhh.

a. Woff
   i. Movement

   Woff memiliki 2 mode *movement*, yakni ketika jumlah musuh 1 atau lebih dari 1. Ketika Woff sedang di keadaan 1vs1, Woff akan menggunakan *movement* Stop and Go untuk menghindari musuh yang memiliki jenis targeting Head-on, Linear, atau Circular. Ketika musuh lebih dari 1, Woff memiliki *movement* Anti-Gravity, di mana gravitasi berasal dari semua musuh dan juga *bullet* yang disimulasikan berdasarkan penurunan energi

musuh. Terdapat 2 bullet yang disimulasikan, yaitu dengan asumsi musuh menggunakan targeting Head-on dan asumsi musuh menggunakan targeting Linear. Kedua *bullet* tersebut direpresentasikan sebagai sebuah garis lurus sebagai lintasan. Perhitungan ini membuat Woff pergi dari wilayah yang menjadi lintasan *bullet* linear dan head-on.

ii. Targeting

Woff memiliki targeting berjenis Play It Forward, yaitu mencatat log state dari setiap musuh untuk menentukan kemana musuh akan bergerak pada tick selanjutnya. State musuh yang disimpan berupa *angular velocity*, *speed,* dan *acceleration*. State disimpan secara sekuensial tiap 4 sekuens state. State tersebut berguna untuk menentukan kemana arah bot musuh bergerak di tiap ticknya. Penentuan state mana ditentukan dengan cara mencari state yang paling banyak terjadi setelah 3 tick state terakhir. Data state disimpan dalam struktur data segment tree yang berguna untuk mempercepat query. Kompleksitas query dari segment tree adalah O(logN) dengan N (state) berjumlah 1024 (angularvelocity) * 17 (speed) * 3 (acceleration).

b. Qwuck

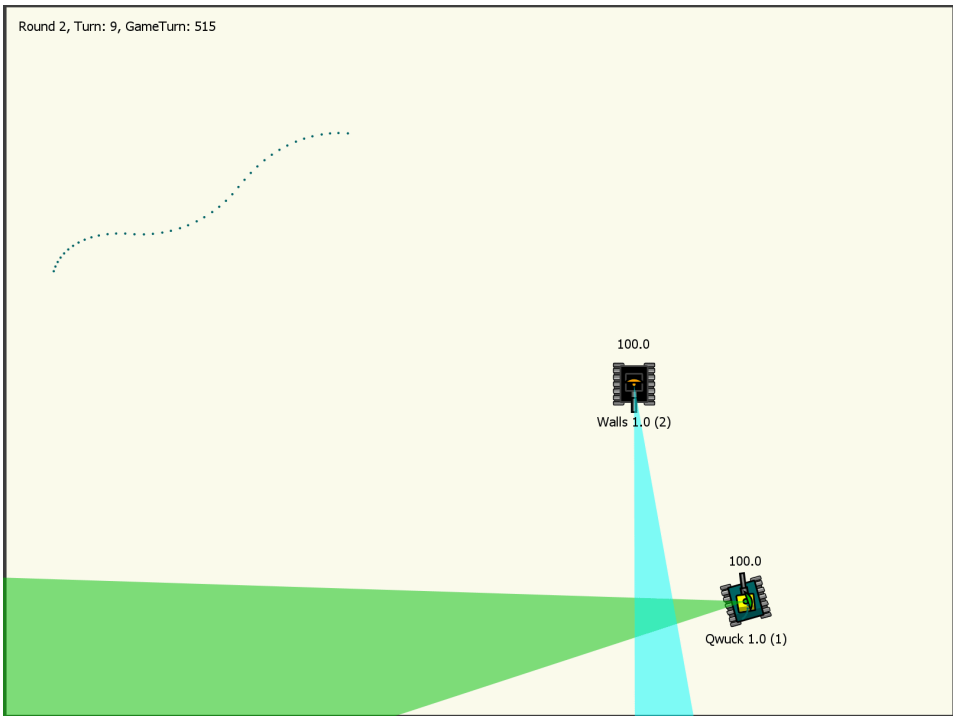
i. *Movement*

Strategi *greedy* yang digunakan pada pergerakan Qwuck adalah bergerak menuju sudut arena yang memiliki rata-rata jarak paling jauh ke semua robot musuh. Di sudut yang telat dipilih, Qwuck bergerak secara melingkar dan akan kembali ke sudut tersebut apabila posisinya sudah menyimpang melebihi suatu tetapan jarak. Fitur ini harus didukung dengan radar yang melakukan pemindaian 360 derajat setiap saat untuk mendeteksi lebih dari satu musuh.

Qwuck bergerak dengan mekanisme *blind man's walking stick* yang menciptakan sebuah titik maya pada radius tetap dan sudut tertentu terhadap arah geraknya di depannya lalu bergerak ke arah titik tersebut. Sudut *walking stick* dimanipulasi dengan fungsi sinus terhadap TurnNumber sehingga menciptakan efek lintasan yang meliuk pada setiap gerakannya. Strategi ini bertujuan untuk memaksimumkan frekuensi perpindahan arah untuk menghindari peluru musuh.

Robot ini juga memiliki fitur *wall avoidance* untuk bergerak menjauhi tembok ketika posisinya terlalu dekat. Untuk mengetahui bahwa robot akan menabrak tembok, digunakan 3 titik maya pada radius tertentu, yakni

di depan, kiri, dan kanan. Ketika titik maya depan berada di luar margin tembok, komponen titik tersebut diperbarui menjadi margin tembok. Kemudian, sudut yang dibentuk dari titik depan-kiri dan depan-kanan dihitung lalu robot digerakkan menuju arah dengan sudut yang lebih kecil.



ii. *Targeting*

Qwuck menggunakan 3 pilihan metode *targeting*, yakni *head-on*, *linear*, dan *circular* targeting. *Circular targeting* digunakan ketika musuh yang terdeteksi memiliki kecepatan sudut yang besar yakni lebih dari 1 atau kurang dari -1. Jika musuh memiliki mutlak dari kecepatan sudut kurang dari 1, *linear targeting* diaktifkan. *Head-on targeting* digunakan jika musuh tidak bergerak. Untuk memperoleh data dari semua musuh, dilakukan *scan* 360 derajat di awal permainan. Apabila terdapat lebih dari satu musuh, Qwuck melakukan *scan* ke segala arah terus menerus dan menembak ke musuh yang paling dekat sebagai strategi *greedy*-nya. Apabila tersisa satu musuh atau terdeteksi musuh yang terlalu dekat (kurang dari 200 satuan), Qwuck mengaktifkan mode *locking* untuk mempercepat pembaharuan data musuh agar informasi yang digunakan tidak terlalu lampau. *Locking* adalah pemindaian radar yang dilakukan hanya di sekitar musuh yg terdeteksi, seakan-akan radar robot mengikuti musuh tersebut.

c. Schmelly

i. Movement

Prinsip utama strategi *greedy* pada pergerakan Schmelly adalah meminimalkan risiko atau bahaya berupa musuh dan dinding. Dengan begitu, Schmelly akan menjauhi bahaya menggunakan pergerakan Anti-Gravity. Sederhananya, Schmelly akan melakukan perhitungan vektor "gaya" pada musuh yang ia pindai, kemudian bergerak menuju arah vektor gaya tersebut. Gaya dihitung berdasarkan sumbu-X dan sumbu-Y menggunakan rumus $F = \frac{1}{r^2}$, dengan r merupakan jarak Schmelly terhadap musuh. Karena arah gaya merupakan kebalikan arah musuh, Schmelly akan secara otomatis menjauhi musuh, dengan harapan bahwa arah geraknya yang menjauhi musuh mengurangi risiko/bahaya. Jika posisi Schmelly berada di dekat tembok, Schmelly akan bergerak sejauh beberapa piksel menuju tengah arena untuk menghindari tabrakan dengan dinding. Ketika berada di luar jangkauan yang dianggap dekat dengan dinding, Schmelly akan kembali melakukan pergerakan Anti-Gravity sehingga pergerakannya dinamis dan selalu menjauhi bahaya.

ii. Targeting

Mode *targeting* Schmelly adalah *linear targeting* atau penargetan secara linear. Berdasarkan data posisi, arah gerak, dan kecepatan musuh, Schmelly melakukan perhitungan terhadap posisi musuh yang diprediksi jika musuh mempertahankan arah dan kecepatannya. Schmelly kemudian mengarahkan dan menembakkan senjatanya pada posisi tersebut. Dalam proses pemilihan targetnya, Schmelly mula-mula akan memutarkan radarnya terus-menerus hingga menemukan musuh. Jika ditemukan musuh, Schmelly akan memindai datanya dan langsung melakukan perhitungan berdasarkan *linear targeting*, kemudian mengarahkan dan menembakkan senjata pada arah yang diprediksi. Setelah menembak, Schmelly akan kembali memutar radar dan melakukan pemindaian hingga menemukan musuh berikutnya. Jika tersisa satu musuh, Schmelly akan *lock* atau mengunci radar yang mengarah kepada musuh tersisa supaya posisi musuh dapat selalu diketahui dan datanya dapat selalu diperbarui.

d. Pffrrrhh

i. Movement

Pffrrrhh menggunakan pergerakan High Risk, yaitu kebalikan dari Anti-Gravity yang menjauhi robot, dia melakukan *ramming* dan

menembak ke musuh terdekat yang dia lihat. Hal ini bertujuan untuk mencari skor bonus dari peluru di atas 1, mendapatkan skor dari melakukan *ramming* beserta meningkatkan akurasi atau *hit rate*.

ii. Targeting

Pffrrrhh menggunakan sistem penargetan secara linear. Hal ini berkesinambungan dengan movement yang digunakan, karena jika musuh berjarak dekat, penggunaan targeting linear sudah memiliki akurasi yang sangat tinggi dan memiliki kemungkinan besar untuk mengenai musuhnya.

3.3. Analisis Efisiensi dan Efektivitas dari Alternatif Solusi Greedy

a. Analisis Woff

Strategi Woff dapat dikatakan menarik karena menyesuaikan jumlah musuh yang ada. Dalam kondisi 1 vs 1, Woff akan menggunakan pergerakan secara Stop and Go, sedangkan saat kondisi melawan banyak robot, Woff akan menggunakan Anti-Gravity. Kelebihan dari Stop and Go adalah kemampuannya untuk mengurangi risiko peluru musuh yang menggunakan metode penargetan *head-on*, *linear*, dan *circular*. Akan tetapi, pergerakan ini belum tentu dapat menghindari peluru dan justru dapat mengalami kendala dengan sistem penargetan acak. Di sisi lainnya, pergerakan Anti-Gravity yang ditambah dengan simulasi peluru musuh memungkinkannya untuk mencari posisi dengan risiko terkecil.

Strategi ofensif Woff menggunakan penargetan dengan sistem *Play It Forward*. Strategi ini sangat adaptif terhadap pergerakan musuh sehingga robot dapat memprediksi pergerakan musuh pada *turn* berikutnya dengan baik. Namun, sistem ini dapat memiliki kelemahan ketika menghadapi robot dengan pergerakan acak atau *random* akibat prediksi yang salah.

Secara keseluruhan, kekuatan utama Woff berada pada pergerakannya yang kompleks dan mampu menyesuaikan keadaan musuh sehingga dapat memilih posisi yang terbaik. Selain itu, metode penargetan Woff yang mampu memprediksi pergerakan musuh memungkinkan pencetakan skor dengan mengenai peluru kepada musuh yang tinggi juga.

b. Analisis Qwuck

Strategi utama pergerakan Qwuck adalah mencari sudut arena dengan risiko minimum kemudian melakukan gerakan osilasi pada sudut tersebut. Kelebihan strategi pergerakannya adalah kemampuan untuk menjauhi robot dari kerumunan dapat mengurangi risiko dengan drastis. Akan tetapi, di sisi lainnya,

karena berada pada sudut, Qwuck dapat diapit dan ditarget oleh beberapa robot lainnya secara bersamaan sehingga justru dapat membahayakannya.

Strategi ofensif Qwuck melibatkan penargetan dinamis berdasarkan perhitungan dan posisi musuh. Qwuck dapat memilih penargetan berbasis *head-on*, *linear*, atau *circular* berdasarkan data musuh. Hal ini memungkinkan prediksi posisi musuh dengan lebih baik dan dapat mencakup kasus pergerakan musuh yang lebih luas. Selain itu, Qwuck akan menembak pada musuh terdekat untuk memaksimalkan peluang terkenanya peluru pada musuh.

Qwuck memiliki strategi pergerakan yang menarik dan penargetan musuh yang adaptif. Pemilihan musuh terdekat juga membuat peluang terkenanya peluru kepada musuh menjadi lebih besar sehingga mengoptimalkan pendapatan skor dari penembakan musuh. Meskipun begitu, pergerakan di sekitar sudut justru dapat berakibat buruk bagi Qwuck jika diapit oleh banyak robot dan menjadi sasaran bersama.

c. Analisis Schmelly

Strategi pergerakan Schmelly berbasis pada pendekatan mencari risiko minimum, hanya secara tidak langsung. Schmelly berusaha untuk menjauhi bahaya, seperti ada gaya yang mendorong Schmelly hingga menjauhi bahaya. Hal ini berlaku baik untuk robot musuh maupun untuk dinding. Hasil pergerakan Schmelly dapat berupa pergerakan yang menyerupai pergerakan osilasi sehingga meningkatkan kemampuan bertahan hidupnya. Akan tetapi, pergerakan yang relatif sederhana ini dapat rentan terhadap sistem penargetan yang maju.

Di sisi lainnya, sistem penargetan Schmelly menggunakan penargetan linear. Hal ini memungkinkannya untuk memprediksi posisi musuh kedepannya dan meningkatkan peluang mengenainya. Namun, sistem penargetan ini dapat dihindari dengan pergerakan osilasi yang arah geraknya berubah dengan konstan.

d. Analisis Pffrrrh

Strategi Pffrrrh yang berbasis High Risk High Reward menyebabkan Pffrrrh menjadi sangat agresif. Pffrrrh berusaha untuk mendapatkan skor dari ram bonus dan damage bonus. Pergerakan Pffrrrh yang mendekati musuh sangat selaras dengan targetingnya yang Linear, karena ketika jarak musuh cukup dekat, kombinasi dari Linear targeting dengan peluru berkaliber besar memberikan keuntungan yang sangat besar bagi Pffrrrh, di mana perhitungan bonus dari damage peluru kalilber besar yang sangat tinggi. Selain itu peluru yang berkaliber besar juga memberikan regenerasi energi yang cukup kencang, sehingga dimungkinkan Pffrrrh mematikan lawan dengan energi yang masih banyak. Jika

Pffrrrh mendapatkan posisi yang baik ketika spawn, Pffrrrh dapat dengan cepat menyelesaikan suatu ronde dengan gampang.

3.4. Strategi Greedy yang Dipilih dari Semua Alternatif

Dari keempat strategi *greedy* yang telah dibuat, Woff merupakan robot dengan strategi unggul secara umum berdasarkan berbagai pengujian. Metode *targeting* Woff lebih unggul dibandingkan robot-robot lainnya karena memiliki akurasi dan tingkat keberhasilan mengenai target yang lebih tinggi. Selain itu, pergerakan Woff juga dapat beradaptasi terhadap berbagai skenario dengan efektif, baik saat 1 vs 1 maupun saat terdapat banyak musuh.

IV. IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi

a. Woff

1. Pseudocode

```
procedure Run
    setup colors and continuous radar turn
    initialize targetDistance, enemyDistance, bullets, myBullets,
dontsag, hitsag


procedure OnTick(TickEvent e)
    // Update bullet positions and draw them
    for each bullet in bullets do
        update bullet position using its speed and direction
        if bullet is outside arena bounds then
            remove bullet

    for each myBullet in myBullets do
        update myBullet position using its speed and direction
        if myBullet is near its target enemy then
            adjust enemy score and remove myBullet
        else if myBullet is outside arena bounds then
            adjust enemy score and remove myBullet

    // Check for condition "Stop And Go
    if hitsag > SAG_LIMIT then
        dontsag ← true
    if (not dontsag) and (EnemyCount = 1) and (targetDistance > 250)
then
        return

    // find the best destination point
    bestX, bestY ← current position
    minGrav ← infinity
    for each angle theta in 0 to 2π (divided into POINT_COUNT parts) do
        for two radius values between MIN_RADIUS and MAX_RADIUS do
            candidateX, candidateY ← point at (theta, r) from current
position
            if candidate is near wall then continue
            grav ← CalcGrav(candidateX, candidateY)
            if grav < minGrav then
```

```
                minGrav ← grav
                bestX, bestY ← candidateX, candidateY
    if minGrav < 0.9 * CalcGrav(destX, destY) then
        destX, destY ← bestX, bestY

    // Move towards the destination
    turnAngle ← angle from current position to (destX, destY)
    set turn and forward movement based on turnAngle and distance


procedure OnScannedBot(ScannedBotEvent e)
    // Update enemy information
    if enemyData does not contain enemy e.id then
        enemyData[e.id] ← new EnemyData(e)
    else
        update enemyData[e.id] with position, energy, and alive status

    // Choose target based on closest distance
    scannedDistance ← distance(current position, (e.X, e.Y))
    if scannedDistance < targetDistance then
        targetId ← e.id
    targetDistance ← scannedDistance

    // Adjust radar and gun
    set radar turn toward (e.X, e.Y)
    firePower ← (Energy / scannedDistance) * GUN_FACTOR
    if gun is ready and (Energy > MIN_ENERGY or scannedDistance is very
short) then
        fire with firePower

    // Detect energy drop and add virtual bullets
    energyDrop ← previous enemy energy - e.Energy
    if energyDrop is within valid range then
        AddVirtualBullet(e.X, e.Y, CalcBulletSpeed(energyDrop),
energyDrop, calculatedDirection)
        AddLinearVirtualBullet(e.X, e.Y, CalcBulletSpeed(energyDrop),
energyDrop)
        if in one-enemy scenario and not near wall then
            adjust movement by reversing sag if necessary
            adjust turn and forward movement based on enemy bearing and
sag

    // Update enemy state history
    currentState ← new State(calculated angular velocity, e.Speed,
calculated acceleration)
    add currentState to enemyData[e.id].StateHistory
    update enemyData[e.id]'s NgramTree with currentState
```

```
    // Targeting: if enemy movement is not head-on, turn gun toward
enemy and exit
    if enemy movement is not head-on then
        set gun turn toward enemy position
        return

    // "Play It Forward": predict enemy future position
    predictedX, predictedY ← e.X, e.Y
    predictedDirection ← e.Direction (in radians)
    predictedSpeed ← e.Speed
    time ← 0
    while (time * CalcBulletSpeed(firePower) < distance(current
position, (predictedX, predictedY))) and (time < 100) do
        if prediction context exists in enemy's NgramTree then
            nextState ← most frequent state from NgramTree
            predictedSpeed ← predictedSpeed + nextState.Acceleration
            update prediction context with nextState
        predictedDirection ← predictedDirection +
nextState.AngularVelocity (if available)
        predictedX ← predictedX + predictedSpeed *
cos(predictedDirection)
        predictedY ← predictedY + predictedSpeed *
sin(predictedDirection)
        increment time
    clamp predictedX and predictedY within arena bounds
    draw predicted target marker
    set gun turn toward (predictedX, predictedY)


procedure OnBulletFired(BulletFiredEvent e)
    AddMyVirtualBullet(current position, e.Bullet.Speed, e.Bullet.Power,
GunDirection, targetId, type 0)
    AddMyVirtualBullet(current position, e.Bullet.Speed, e.Bullet.Power,
direction toward enemyData[targetId].LastPosition, targetId, type 1)


procedure OnHitByBullet(HitByBulletEvent e)
    if EnemyCount = 1 then
        increment hitsag


procedure OnBotDeath(BotDeathEvent e)
    mark enemyData[e.VictimId] as dead
    if e.VictimId = targetId then
        targetDistance ← infinity
```

2. Implementasi Struktur Data

Struktur Data	Penjelasan

`public struct State`
`{`
`  public int AngularVelocity;`
`  public int Speed;`
`  public int Acceleration;`
`}`
	State berisi *Angularvelocity, Speed,* dan *Acceleration* yang didapatkan setial kalo bot men-*scan* musuh.

`public class StateSequence`
`{`
`  public List<State> States { get; }`
`}`
	StateSequence berisi sekuens dari state. Kelas ini berfungsi sebagai parameter saat dilakukan query state.

`public class EnemyData`
`{`
`  public List<State> StateHistory { get; } = new List<State>();`
`  public Dictionary<StateSequence, TransitionSegmentTree> NgramTree { get; } = new Dictionary<StateSequence, TransitionSegmentTree>();`
`  public double LastDirection { get; set; }`
`  public bool HasPrevious { get; set; } = false;`
 	
`  public double LastX { get; set; }`
`  public double LastY { get; set; }`
`  public double LastEnergy { get; set; }`
`}`
`  public double LastSpeed { get; set; }`
`}`
`  public bool IsAlive { get; set; } = true;`
`}`
	EnemyData berisi StateHistory yang berupa List of State. StateHistory berfungsi untuk membantu pembentukan NgramTree. NgramTree sendiri berisi Dictionary untuk menyimpan Sequence of State. Atribut lain seperti LastDirection, HasPrevious, LastX, LastY berfungsi untuk membantu pembentukan suatu State. LastEnergy berfungsi untuk penentuan kapan suatu bullet ditembakkan oleh musuh. IsAlive dan atribut lain berfungsi untuk penentuan movement.

`public struct Bullet`
`{`
`  public double X;`
`  public double Y;`
	Bullet berisi atribut dari sebuah peluru yang ditembakkan sebuah bot. Bullet digunakan sebagai virtual bullet.

`    public double Speed;`
`    public double Direction;`
`    public double Power;`
`}`	

`public class Line2D`
`{`
`    public double X1 { get; }`
`    public double Y1 { get; }`
`    public double X2 { get; }`
`    public double Y2 { get; }`
`}`
	Line2D berisi atribut sebuah garis dalam 2 dimensi. Kelas ini digunakan untuk perhitungan gravitasi lintasan peluru yang direpresentasikan sebagai sebuah garis.

`public class TransitionSegmentTree`
`{`
`    private List<KeyValuePair<State, int>> data;`
`    private int size;`
`    private (State state, int frequency)[] tree;`
`    private Dictionary<State, int> stateToIndex;`
`}`
	TransitionSegmentTree digunakan untuk menyimpan state. Segment Tree digunakan untuk mempercepat proses query ketika melakukan Play It Forward.

3. Implementasi Fungsi dan Prosedur

Nama Fungsi/Prosedur	Deskripsi
`public override void Run()`	Prosedur utama yang menjalankan robot. Pada prosedur ini juga radar robot diatur agar selalu berputar ke kiri dan setiap komponen diatur agar dapat bergerak secara independen.
`public override void OnTick(TickEvent e)`	Prosedur yang dijalankan pada setiap *tick* atau *turn*. Berisi simulasi pergerakan peluru dan perhitungan pergerakan robot berikutnya dengan strategi risiko minimum.
`public override void OnScannedBot(ScannedBotEvent e)`	Prosedur yang dijalankan jika robot berhasil memindai sebuah tank musuh. Berisi pengisian data musuh, pemindaian dan pengaturan radar, pengaturan untuk menembak musuh, prediksi gerak musuh, dan membuat simulasi peluru musuh.
`public override void`	Prosedur ini dijalankan jika ada robot yang

`OnBotDeath(BotDeathEvent e)`	tereliminasi. Robot akan *reset* atau "melepaskan" variabel *targetDistance* untuk diperbarui.
`private double CalcRisk(double candidateX, double candidateY)`	Fungsi ini melakukan perhitungan risiko berdasarkan pergerakan musuh, posisi peluru, dan target yang diincar robot saat ini sehingga dapat menemukan pilihan pergerakan yang paling aman.
`private void AddVirtualBullet(double x, double y, double speed, double power)`	Prosedur ini mensimulasikan dua peluru musuh, yakni berdasarkan penargetan secara langsung (*head-on targeting*) dan penargetan secara linear (*linear targeting*), terhadap robot ini.
`private double distanceSq(double x1, double y1, double x2, double y2)`	Fungsi pembantu yang mengembalikan jarak antara dua titik dipangkatkan.
`private double distance(double x1, double y1, double x2, double y2)`	Fungsi pembantu yang mengembalikan jarak antara dua titik.

b. Qwuck

1. Pseudocode

```
procedure Run()
    // Atur warna dan rotasi independen
    set bot color
    set bot body, gun, and radar to turn independently

    // Inisialisasi variabel
    enemies <- empty dictionary          // Data musuh
    navigating <- true                   // Status navigasi aktif
    targetId <- -1                       // ID target awal
    targetLocked <- false                // Status kunci target
    oneVsOne <- false                    // Flag one-vs-one
    isScanningAll <- true                // Flag scan semua musuh

    // Lakukan scan radar awal
    for i from 1 to 8 do
        TurnRadarRight(45 degrees)
    isScanningAll <- false

    // Tentukan pojok (corner) teraman
    corner <- SafestCorner()
```

```
    print "Hello! I'm Qwuck!"             // Tampilkan pesan sambutan


procedure OnTick(TickEvent e)
    // Perbarui jejak visual (opsional)
    add current position to trail queue
    if size(trail) > 50 then
        remove oldest point from trail
    for each point in trail do
        draw ellipse at point        // Gambar titik jejak

    // Logika pergerakan
    if navigating then
        if WallAvoidance() returns false then
            turn <- BearingTo(corner.x, corner.y)   // Hitung sudut ke pojok
teraman
            walkingStick <- CalcStickEnd(turn + 40 * sin(TurnNumber * (2π /
40)), current Direction)
            MoveTo(walkingStick.x, walkingStick.y)    // Gerak ke titik
walking stick
            if DistanceTo(corner.x, corner.y) < OSCILLATION_RADIUS then
                TargetSpeed <- 0                      // Hentikan pergerakan
                navigating <- false                  // Nonaktifkan navigasi
        else
            Oscillate()                               // Lakukan gerakan
osilasi

    // Periksa ulang posisi aman jika terlalu jauh atau tidak aman
    if DistanceTo(corner.x, corner.y) > 4 × OSCILLATION_RADIUS then
        navigating <- true
    if SafestCorner() is not equal to corner then
        navigating <- true
        corner <- SafestCorner()

    // Logika targeting
    if oneVsOne then
        if targetLocked then
            TrackScanAt(enemies[targetId].LastX, enemies[targetId].LastY)
            CircularTargetingConditional(targetId,
CalcFirePower(enemies[targetId].LastX, enemies[targetId].LastY))
            targetLocked <- false
        else
            TurnRadarLeft(20 degrees)
    else
        TurnRadarLeft(20 degrees)
        targetId <- SelectTargetEnemy()
        if targetId ≠ -1 then
```

```
            CircularTargetingConditional(targetId,
CalcFirePower(enemies[targetId].LastX, enemies[targetId].LastY))


procedure OnScannedBot(ScannedBotEvent e)
    // Perbarui data musuh
    if enemies does not contain key e.ScannedBotId then
        enemies[e.ScannedBotId] <- new EnemyData(e)    // Simpan data musuh
baru
    else
        update enemies[e.ScannedBotId] with new values from e  // Perbarui
data musuh yang sudah ada

    // Pilih target berdasarkan jarak
    if DistanceTo(e.X, e.Y) < 200 or (number of enemies equals 1) then
        targetId <- e.ScannedBotId                  // Set target ke musuh
terdekat
        oneVsOne <- true
    else
        oneVsOne <- false

    // Kunci target jika musuh sesuai target saat ini
    if e.ScannedBotId equals targetId then
        targetLocked <- true


procedure OnBotDeath(BotDeathEvent e)
    // Hapus data musuh yang telah mati
    if enemies contains key e.VictimId then
        remove enemies[e.VictimId]
        if e.VictimId equals targetId then
            oneVsOne <- false
            targetId <- -1
        if number of enemies equals 1 then
            targetId <- remaining enemy's id     // Pilih musuh yang tersisa
sebagai target
            oneVsOne <- true
```

2. Implementasi Struktur Data

Struktur Data	Penjelasan

`class EnemyData {`
`  public double LastX { get; set; }`
`  public double LastY { get; set; }`
	Struktur data ini digunakan untuk menyimpan informasi musuh pada deteksi terakhir dan

`public double LastDirection { get; set; }`
`public double LastSpeed { get; set;}`
`public double LastEnergy { get; set; }`
`public double LastTurnNumber { get; set; }`
`public double PrevX { get; set; }`
`public double PrevY { get; set; }`
`public double PrevDirection { get; set; }`
`public double PrevSpeed { get; set; }`
`public double PrevTurnNumber { get; set; }`
`}`
	deteksi sebelumnya. Data setiap musuh disimpan pada sebuah Dictionary dengan id robot musuh sebagai *key* dan EnemyData sebagai *value*.

`class Point2D {`
`public double x;`
`public double y;`
`}`
	Struktur data Point2D digunakan untuk memudahkan proses menulis kode yang berkaitan dengan manipulasi titik.

3. Implementasi Fungsi dan Prosedur

Nama Fungsi/Prosedur	Penjelasan
`public override void Run()`	Prosedur ini melakukan inisialisasi *member attributes* robot dan melakukan *scan* awal 360 derajat untuk mengumpulkan informasi semua musuh lalu mencari sudut paling aman berdasarkan rata-rata jarak ke semua musuh.
`public override void OnTick(TickEvent e)`	Prosedur yang dijalankan setiap tick permainan dan bertanggung jawab untuk pergerakan bot berdasarkan strategi yang telah ditentukan. Ini termasuk pergerakan sinusoidal menuju sudut teraman serta pergerakan berputar (osilasi). Selain itu, prosedur ini juga menangani penargetan musuh baik untuk satu musuh maupun banyak musuh.
`public override void OnScannedBot(ScannedBotEvent e)`	Prosedur ini dipanggil ketika bot berhasil mendeteksi musuh. Ini menyimpan data musuh yang terdeteksi dan memutuskan

	apakah akan mengunci target atau memilih musuh terdekat untuk di-*lock*.
`public override void OnBotDeath(BotDeathEvent e)`	Prosedur ini dipanggil ketika bot mati untuk menghapus data musuh tersebut dan mengaktifkan mode 1 vs 1 jika tersisa satu musuh.
`private void Oscillate()`	Prosedur ini membuat bot bergerak dalam pola melingkar sambil berosilasi tegak lurus dengan arah gerak.
`private bool WallAvoidance()`	Fungsi ini menangani penghindaran dinding dengan menggunakan teknik *walking stick*. Bot akan memeriksa apakah ada dinding di depannya dan mengkalkulasi sudut apit *stick* depan-kiri dan depan-kanan lalu pergi ke arah dengan sudut terkecil untuk menghindari dinding.
`private bool IsACorner(double x, double y, double margin)`	Fungsi ini memeriksa apakah posisi (x, y) berada di dekat salah satu sudut arena dengan margin yang diberikan.
`private bool IsCloseToWall()`	Fungsi ini memeriksa apakah bot berada terlalu dekat dengan dinding arena.
`private bool IsOutsideArena(double x, double y)`	Fungsi ini memeriksa apakah posisi (x, y) berada di luar batas arena.
`private Point2D SafestCorner()`	Fungsi ini mencari sudut arena yang paling aman berdasarkan jarak rata-rata musuh dari setiap sudut. Sudut yang memiliki jarak rata-rata terjauh dari musuh akan dipilih sebagai sudut teraman.
`private void MoveTo(double x, double y, double vel = 8)`	Prosedur ini menggerakkan bot menuju posisi (x, y) dengan kecepatan yang ditentukan (vel). Bot hanya bergerak ke arah depan sehingga harus memutar balik apabila tujuannya berada di belakangnya.
`private void TrackScanAt(double x, double y)`	Prosedur yang memutarkan radar di sekitar posisi masukan.

`private int SelectTargetEnemy()`	Fungsi yang mengembalikan ID robot musuh dengan jarak terdekat dengan robot ini.
`private double CalcFirePower(double targetX, double targetY)`	Fungsi yang menghitung daya tembak robot berdasarkan jarak musuh.
`private void CircularTargetingConditional(int enemyId, double firePower)`	Prosedur yang melakukan penargetan secara melingkar atau *circular targeting* dengan memperhitungkan kecepatan sudut musuh. Prosedur ini akan memanggil *LinearTargeting* atau *HeadOnTargeting* jika kecepatan sudut musuh tidak memenuhi.
`private void LinearTargeting(double targetX, double targetY, double targetSpeed, double targetDirection, double firePower)`	Prosedur yang melakukan penargetan secara linear atau *linear targeting* yang memprediksi posisi musuh jika musuh bergerak secara konstan pada arah yang sama.
`private void HeadOnTargeting(double targetX, double targetY, double firePower)`	Prosedur yang melakukan penargetan secara langsung atau *head-on targeting*, yakni menargetkan posisi musuh secara langsung.
`private (Point2D, Point2D) CalcWalkingStick()`	Fungsi yang menghitung posisi *stick* kiri dan *stick* kanan untuk menghindari tabrakan dengan dinding.
`private Point2D CalcStickEnd(double angle, double heading, double length = STICK_LENGTH, Point2D center = null)`	Fungsi yang menghitung posisi ujung *stick* berdasarkan sudut, arah bot, dan panjang *stick*. Fungsi ini digunakan untuk menghindari tabrakan dengan dinding.
`private double DegreesToRadians(double degrees)`	Fungsi pembantu yang mengembalikan radian berdasarkan masukan derajat.
`private double RadiansToDegrees(double radians)`	Fungsi pembantu yang mengembalikan derajat berdasarkan masukan radian.
`private Point2D rotatePoint(Point2D point, Point2D center, double angle)`	Fungsi yang merotasikan suatu titik terhadap suatu koordinat pusat.

c. Schmelly

1. Pseudocode

```
procedure Run()
      scan(360) // Lakukan scan terus-menerus hingga menemukan musuh

procedure OnTick(TickEvent e)
      if (enemyDetected) then
            enemyDetected <- false
            if (isNearWall()) then
                  moveToCenter() // Gerak mendekati tengah beberapa px
            else
                  antiGravity() // Gerak menjauhi musuh yang di-scan

procedure OnScannedBot(ScannedBotEvent e)
      enemyDetected <- true
      enemyData = new EnemyData(e) // Simpan data musuh
      // Putar radar menuju posisi musuh
      setRadarTurn(e.X, e.Y)
      // Atur daya tembak peluru
      setGunFire(e.X, e.Y)
      // Lakukan targeting secara linear
      LinearTarget(e.X, e.Y, e.Speed, e.Direction, e.Energy)
```

2. Implementasi Struktur Data

	Penjelasan

`public struct EnemyData {`
`  public Point2D coordinate;`
`  public double enemyDirection;`
`  public double enemySpeed;`
`  public double enemyEnergy;`
`}`
	Struktur data *EnemyData* berisi data-data yang didapatkan dari musuh. Data tersebut berisi *coordinate* atau koordinat musuh dalam bentuk struktur Point2D, *enemyDirection* yakni arah gerak musuh, *enemySpeed* dan *enemyEnergy* yakni energi yang dimiliki musuh yang di-*scan*.
	Struktur ini digunakan pada pengolahan vektor gaya dari musuh yang ditemukan. Data musuh selalu diperbarui setiap kali dilakukan *scan* terhadap musuh (dapat dilihat pada prosedur *OnScannedBot* pada *pseudocode* di atas.

`public struct Point2D {`
`  public double X;`
`  public double Y;`
`}`
	Struktur data *Point2D* berisi koordinat musuh pada sumbu-X dan sumbu-Y. Struktur ini digunakan pada struktur data *EnemyData* untuk menyimpan koordinat musuh dengan lebih mudah.

3. Implementasi Fungsi dan Prosedur

Nama Fungsi/Prosedur	Deskripsi
`public override void Run()`	Prosedur utama yang menjalankan robot. Pada prosedur ini juga radar robot diatur agar selalu berputar ke kiri dan setiap komponen diatur agar dapat bergerak secara independen.
`public override void OnTick(TickEvent e)`	Prosedur yang dijalankan pada setiap *tick* atau *turn*. Berisi pilihan pergerakan robot jika terdeteksi musuh. Dapat dilihat pada *pseudocode* di atas.
`public override void OnScannedBot(ScannedBotEvent e)`	Prosedur yang dijalankan jika robot berhasil memindai sebuah tank musuh. Berisi pencatatan data musuh, pengaturan radar dan daya tembak peluru, serta penargetan musuh secara linear. Dapat dilihat pada *pseudocode* di atas.
`public double getEnemyBearing(double enemyX, double enemyY)`	Fungsi yang mengembalikan *bearing* atau arah musuh terhadap robot.
`public bool isNearWall()`	Fungsi yang mengembalikan *true* jika posisi robot dekat dengan dinding (dihitung dari margin sebesar 100 piksel).
`public void setRadarTurn(double enemyX, double enemyY)`	Prosedur yang mengarahkan radar kepada musuh yang sebelumnya tercatat.
`public void setGunFire(double enemyX, double enemyY)`	Prosedur yang mengatur daya tembak peluru berdasarkan jarak musuh.
`public double getEnemyDistance(EnemyData enemyData)`	Fungsi yang mengembalikan jarak musuh dengan masukan berupa struktur data *EnemyData*.
`public void LinearTarget(double targetX, double targetY, double targetSpeed, double targetDirection, double firePower)`	Prosedur yang melakukan targeting secara linear dengan melakukan prediksi posisi musuh berdasarkan data yang dimiliki sekarang.
`public void moveToCenter(double vel)`	Prosedur yang menggerakkan robot supaya bergerak menuju pusat arena. Prosedur ini guna untuk menghindari tabrakan dengan dinding.

`public void antiGravity()`	Prosedur yang menggerakkan robot dengan menerapkan prinsip Anti-Gravity. Prosedur melakukan perhitungan terhadap "gaya" musuh pada arah yang berbalik dengan robot dan menggerakkan robot pada arah tersebut sehingga robot menjauhi musuh.
`public double degToRad(double degree)`	Fungsi pembantu yang mengembalikan derajat berdasarkan masukan radian.
`public double radToDeg(double radian)`	Fungsi pembantu yang mengembalikan radian berdasarkan masukan derajat.

d. Pffrrrhh

1. Pseudocode

```
procedure Run()
    Set bot body, gun, and radar to turn independently
    set bot color



procedure OnScannedBot(ScannedBotEvent e)
    // Hitung sudut radar untuk mengunci posisi musuh
    radarAngle <- PositiveInfinity *
NormalizeRelativeAngle(RadarBearingTo(e.X, e.Y))
    if radarAngle is not NaN then
        SetTurnRadarLeft(radarAngle)

    // Hitung daya tembak berdasarkan energi dan jarak ke musuh
    firePower <- max(3 * Energy / DistanceTo(e.X, e.Y), 0.1)
    if GunTurnRemaining equals 0 then
        SetFire(firePower)

    // Lakukan targeting linier
    LinearTargeting(e.X, e.Y, e.Speed, e.Direction, firePower)

    // Kalkulasi risiko untuk menentukan pergerakan (High Risk High Return)
    risk <- 0
    targetX <- e.X
    targetY <- e.Y
    for i from 0 to 359 do
        x <- X + 100 * cos(DegreesToRadians(i))
        y <- Y + 100 * sin(DegreesToRadians(i))
```

```
        tempRisk <- e.Energy / ((x - e.X)^2 + (y - e.Y)^2 + 1e-6)
        if tempRisk > risk then
            risk <- tempRisk
            targetX <- x
            targetY <- y

    // Gerak ke posisi dengan risiko tertinggi
    turn <- BearingTo(targetX, targetY) * (PI / 180)
    SetTurnLeft(tan(turn) * (180 / PI))
    SetForward(DistanceTo(targetX, targetY) * cos(turn))
```

2. Implementasi Struktur Data
   Bot ini tidak menggunakan struktur data.
3. Implementasi Fungsi dan Prosedur

Nama Fungsi/Prosedur	Deskripsi
`public override void Run()`	Prosedur utama yang menjalankan robot. Pada prosedur ini juga radar robot diatur agar selalu berputar ke kanan dan setiap komponen diatur agar dapat bergerak secara independen.
`public override void OnTick(TickEvent e)`	Prosedur yang dijalankan jika robot berhasil memindai sebuah tank musuh. Berisi pengaturan radar untuk *lock* pada musuh yang di-scan, pengaturan daya tembak senjata, penargetan secara linear, dan pengaturan pergerakan agar mendekati musuh.
`private void LinearTargeting(double targetX, double targetY, double targetSpeed, double targetDirection, double firePower)`	Prosedur yang melakukan targeting secara linear dengan melakukan prediksi posisi musuh berdasarkan data yang dimiliki sekarang.
`private double DegreesToRadians(double degrees)`	Fungsi pembantu yang mengembalikan radian berdasarkan masukan derajat.

4.2. Pengujian

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	woff 1.0	2608	1200	150	907	161	191	0	7	3	0
2	Schmelly 1.0	1770	900	90	686	36	58	0	1	3	5
3	Qwuck 1.0	1595	800	60	532	20	155	28	2	2	5
4	Pffrrrhh 1.0	1104	100	0	493	0	476	34	0	2	0

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	woff 1.0	2437	1250	210	707	94	175	0	5	4	0
2	Pffrrrhh 1.0	1982	400	0	797	59	610	115	2	4	2
3	Schmelly 1.0	1667	700	60	686	67	154	0	2	1	4
4	Qwuck 1.0	1289	650	30	448	13	148	0	1	1	4

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	woff 1.0	2356	1100	180	867	123	85	0	6	1	1
2	Schmelly 1.0	1876	1000	90	619	72	95	0	3	1	4
3	Pffrrrhh 1.0	1808	350	0	764	81	557	56	0	6	2
4	Qwuck 1.0	1638	550	30	643	64	352	0	1	2	3

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	woff 1.0	2550	1100	150	912	90	236	61	7	2	0
2	Qwuck 1.0	1764	1000	90	530	60	84	0	1	4	3
3	Pffrrrhh 1.0	1511	200	0	681	41	529	58	2	2	1
4	Schmelly 1.0	1429	700	60	445	35	157	31	0	2	6

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	woff 1.0	2476	1200	150	909	121	72	24	6	3	1
2	Qwuck 1.0	2146	1100	120	673	48	172	33	2	5	2
3	Schmelly 1.0	1353	550	30	526	17	229	0	1	0	5
4	Pffrrrhh 1.0	1323	150	0	590	62	521	0	1	2	2

Dari 5 kali pengujian yang dilakukan, dapat disimpulkan bahwa:

- Distribusi skor sangat bergantung kepada Pffrrrhh, karena siapapun yang menjadi target Pffrrrhh akan sangat mungkin menjadi korban pertama di round tersebut. Movement yang agresif dari Pffrrrhh dengan paduan *linear targeting* membuat bot yang menjadi target Pffrrrhh sulit untuk sintas dalam ronde tersebut.
- Schmelly unggul ketika mendapatkan posisi spawn yang strategis di awal. Hal itu menyebabkan *snowball effect* terhadap round tersebut. Ditambah lagi, *movement* dan *targeting* dari Schmelly yang sulit diprediksi membuat keberjalanan ronde tersebut lebih mudah untuk Schmelly.

- Qwuck menjadi sangat kuat ketika Pffrrrhh sudah mati. Movement dari Qwuck sangat sulit diprediksi oleh Schmelly dan Woff sehingga waktu dari ronde tersebut berlangsung sangat lama.
- Woff menjadi bot yang paling konsisten karena hampir semua strategi dari bot lain di-*counter* oleh Woff, seperti Anti-Gravity yang dapat menjauh dari Pffrrrhh dan Stop and Go yang menghindari bullet Head-on, Linear, dan Circular dari Qwuck dan Schmelly.

V. KESIMPULAN DAN SARAN

5.1. Kesimpulan

Robocode adalah sebuah permainan dengan domain persoalan yang dapat diselesaikan secara *greedy*. Tantangan utama dalam permainan ini adalah menentukan aksi optimal yang dapat dilakukan pada suatu momen singkat untuk memperoleh keuntungan sebesar-besarnya tanpa mengetahui konsekuensi di masa depan. Aksi dapat ditentukan berdasarkan informasi yang dapat diperoleh robot saat ini melalui *scan* radar lalu diimplementasikan melalui fungsi-fungsi yang disediakan API. Dengan menentukan serangkaian aksi yang efektif untuk menangkal strategi musuh, solusi akhir yakni kemenangan dengan poin terbanyak dapat diperoleh.

Terdapat 4 alternatif strategi *greedy* yang diimplementasikan pada bot dalam permainan Robocode dengan ciri khasnya masing-masing. Strategi yang diimplementasikan dibagi menjadi dua aspek yakni penargetan dan pergerakan. Strategi penargetan yang dieksplorasi meliputi *head-on*, *linear*, *circular*, dan *play it forward*. Di sisi lain, strategi pergerakan yang telah dieksplorasi meliputi *ramming*, *circle*, *random*, *osilasi*, *corner*, *anti-gravity*, dan *stop and go*. Berbagai strategi *greedy* yang dieksplorasi diimplementasikan pada 4 robot, yakni Woff, Qwuck, Schmelly, dan Pffrrrhh, untuk diuji performanya. Hasil yang diperoleh adalah Woff lebih unggul secara umum dibandingkan yang lain karena kemampuannya untuk beradaptasi terhadap strategi musuh yang beragam.

5.2. Saran

1. Kelompok menerapkan manajemen waktu yang lebih baik agar tidak mengerjakan tugas mendekati waktu tenggat.
2. Asisten lebih responsif di *spreadsheet* QnA dan menetapkan batasan yang lebih jelas di spesifikasi agar tidak ada miskomunikasi.
3. Sebaiknya asistensi dibuat wajib saja untuk meminimalisisasi kesalahpahaman spesifikasi.

VI. LAMPIRAN

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

Berikut adalah pranala menuju repositori GitHub:

https://github.com/karolyangqian/Tubes1_Nguyen

Berikut adalah pranala menuju video YouTube:

https://youtube.com/shorts/P3_IchA4bAc

VII. DAFTAR PUSTAKA

Programiz. (n.d). “Greedy Algorithm”. Programiz.com. Diakses pada 22 Maret 2025 melalui https://www.programiz.com/dsa/greedy-algorithm.

Munir, Rinaldi. (2025). “Algoritma Greedy (Bagian 1)”. Homepage Rinaldi Munir. Diakses pada 22 Maret 2025 melalui https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/04-Algoritma-Greedy-(2025)-Bag1.pdf.

Munir, Rinaldi. (2025). “Algoritma Greedy (Bagian 2)”. Homepage Rinaldi Munir. Diakses pada 22 Maret 2025 melalui https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/05-Algoritma-Greedy-(2025)-Bag2.pdf.

Robowiki. (17 Agustus 2017). “Targeting”. Robowiki.net. Diakses pada 25 Maret 2025 melalui https://robowiki.net/wiki/Targeting.

Robowiki. (25 Agustus 2014). “Targeting”. Robowiki.net. Diakses pada 25 Maret 2025 melalui https://robowiki.net/wiki/Movement.