LAPORAN TUGAS BESAR 1

IF2211 STRATEGI ALGORITMA

Pemanfaatan Algoritma Greedy dalam Pembuatan Bot Permainan Robocode Tank Royale

Blitzkrieg (Kelompok 39):

Shanice Feodora Tjahjono – 13523097

Brian Ricardo Tamin – 13523126

Andrew Tedjapratama – 13523148

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

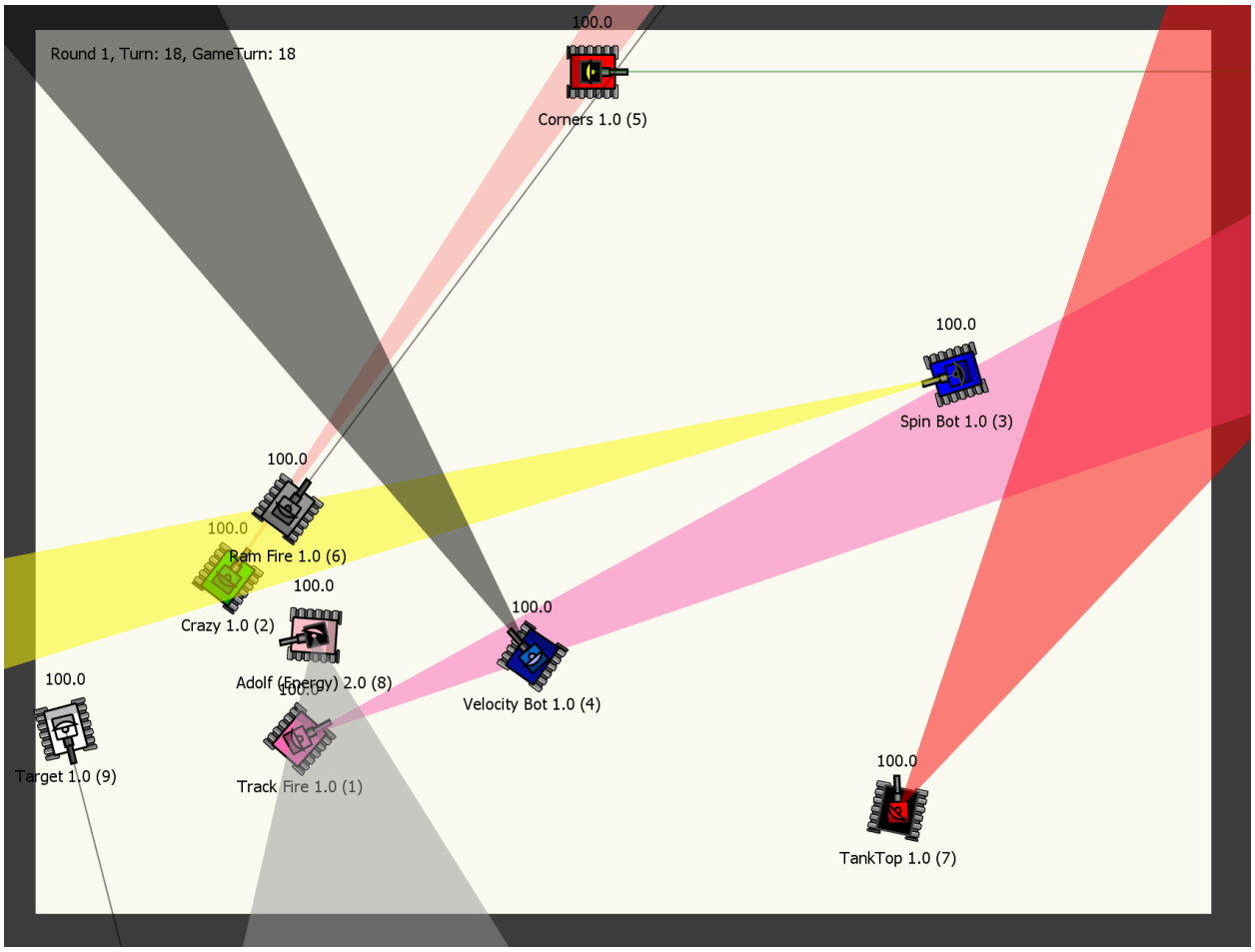
INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

BAB I

DESKRIPSI TUGAS



Gambar 1 Robocode Tank Royale

Robocode adalah permainan pemrograman yang bertujuan untuk membuat kode bot dalam bentuk tank virtual untuk berkompetisi melawan bot lain di arena. Pertempuran Robocode berlangsung hingga bot-bot bertarung hanya tersisa satu seperti permainan Battle Royale, karena itulah permainan ini dinamakan Tank Royale. Nama Robocode adalah singkatan dari "Robot code," yang berasal dari versi asli/pertama permainan ini. Robocode Tank Royale adalah evolusi/versi berikutnya dari permainan ini, di mana bot dapat berpartisipasi melalui Internet/jaringan. Dalam permainan ini, pemain berperan sebagai programmer bot dan tidak memiliki kendali langsung atas permainan. Pemain hanya bertugas untuk membuat program

yang menentukan logika atau "otak" bot. Program yang dibuat akan berisi instruksi tentang cara bot bergerak, mendeteksi bot lawan, menembakkan senjatanya, serta bagaimana bot bereaksi terhadap berbagai kejadian selama pertempuran.

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

Komponen-komponen dari permainan ini antara lain:

1. Rounds dan Turns

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

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

   Pada setiap turn, sebuah bot dapat:

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

   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. Game engine yang akan digunakan pada tugas besar ini tidak mengikuti aturan default mengenai komponen Round & Turns.

2. Batas Waktu Giliran

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

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

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

3. Energi

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

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

4. Peluru

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

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

5. Panas Meriam (Gun Heat)

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

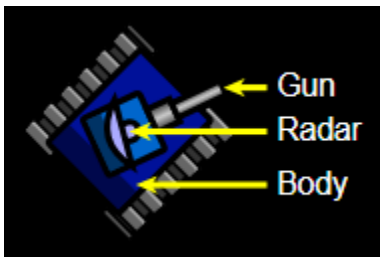
6. Tabrakan

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

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

7. Bagian Tubuh Tank

   Tubuh tank terdiri dari 3 bagian:

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

8. Pergerakan

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

9. Berbelok

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

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

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

10. Pemindaian

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

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

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

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

11. Skor

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

- **Bullet Damage:** Bot mendapatkan **poin sebesar *damage*** yang dibuat kepada bot musuh menggunakan peluru.
- **Bullet Damage Bonus:** Apabila peluru berhasil membunuh bot musuh, bot mendapatkan **poin sebesar 20% dari *damage*** yang dibuat kepada musuh yang terbunuh.
- **Survival Score:** Setiap ada bot yang mati, bot lainya yang masih bertahan pada ronde tersebut mendapatkan **50 poin.**

- **Last Survival Bonus:** Bot terakhir yang bertahan pada suatu ronde akan mendapatkan **10 poin** dikali dengan banyaknya musuh.
- **Ram Damage:** Bot mendapatkan **poin sebesar 2 kalinya *damage*** yang dibuat kepada bot musuh dengan cara menabrak.
- **Ram Damage Bonus:** Apabila musuh terbunuh dengan cara ditabrak, bot mendapatkan **poin sebesar 30% dari *damage*** yang dibuat kepada musuh yang terbunuh.

Skor akhir bot adalah akumulasi dari 6 komponen diatas. Perlu diperhatikan bahwa game akan menampilkan berapa kali suatu bot meraih peringkat 1, 2, atau 3 pada setiap ronde. Namun, hal ini tidak dihitung sebagai komponen skor maupun untuk perangkingan akhir. Bot yang dianggap menang pertempuran adalah bot dengan akumulasi skor tertinggi.

BAB II
LANDASAN TEORI

2.1 Algoritma Greedy

Algoritma Greedy merupakan algoritma pemecahan masalah heuristik yang mengikuti pendekatan “kerakusan” atau ketamakan yang berfokus pada pengambilan keputusan lokal untuk setiap langkah, dengan harapan akan mencapai solusi optimal secara keseluruhan. Algoritma ini memiliki dua karakteristik utama. Pertama, pada setiap langkah, algoritma ini mengambil pilihan yang terbaik yang dapat diperoleh di saat itu tanpa mempertimbangkan konsekuensi kedepannya (*local optimum*). Pilihan ini bersifat final dan tidak dapat diubah kembali (tidak ada *backtracking*). Kedua, sebuah solusi optimum global diharapkan dapat tercapai dengan memilih optimum lokal di setiap langkah sebelumnya.

Algoritma Greedy seringkali efisien dalam hal waktu dan memori karena keputusan diambil langsung tanpa perlu menelusuri semua kemungkinan. Namun, algoritma ini hanya menghasilkan solusi terbaik pada masalah tertentu yang memiliki sifat khusus. Tidak semua masalah bisa diselesaikan dengan baik menggunakan pendekatan "ambil yang terbaik sekarang" ini.. Beberapa contoh persoalan umum yang diselesaikan dengan algoritma Greedy, antara lain masalah penukaran uang (*coin change problem*), pohon merentang minimum (*minimum spanning tree*), dan kode Huffman untuk kompresi data.

2.2 Elemen Algoritma Greedy

Terdapat enam elemen Greedy, antara lain:

1. Himpunan Kandidat (C): berisi kandidat tersedia yang dapat dipilih pada setiap langkah.
2. Himpunan Solusi (S): berisi kandidat yang sudah dipilih dan diambil untuk solusi akhir.
3. Fungsi Solusi (*solution function*): menentukan apakah himpunan solusi sudah membentuk solusi lengkap terhadap permasalahan.
4. Fungsi Seleksi (*selection function*): strategi heuristik untuk memilih kandidat berdasarkan strategi *greedy* tertentu.
5. Fungsi Kelayakan (*feasibility function*): memeriksa apakah kandidat yang dipilih layak dimasukkan ke dalam himpunan solusi tanpa melanggar batasan yang ada.
6. Fungsi Objektif (*objective function*): mengevaluasi seberapa optimal solusi yang diperoleh.

2.3 Cara Kerja Bot Robocode Tank Royale

Perilaku bot tidak dikendalikan secara manual, tetapi melalui pemrograman. Bot menerima data dari game engine, lalu mengirimkan kembali command melalui API yang ada. Dengan kata lain, API yang menjembatani ‘komunikasi’ antara game server dengan bot. Pada awal setiap giliran (turn) permainan, bot menerima informasi mengenai position dan orientation (body, gun, radar), energy level, gun heat, musuh, dan event. Dengan informasi yang diterima, bot menentukan arah gerak, apakah radar diaktifkan dan berputar, menembak atau tidak, dan tanggapan terhadap event. Keputusan-keputusan ini diambil sejalan dengan program bot.

2.4 Alur Pengembangan Bot

Suatu folder bot terdiri dari 5 file dan disiapkan dengan urutan sebagai berikut,

1. JSON config (.json)
   Berisi metadata mengenai bot.

2. C# source code (.cs)
   Berisi algoritma greedy serta fungsi lainnya untuk mengatur warna, gerak, dan reaksi bot dalam permainan.

3. Build config (.csproj)
   Project file yang berisi versi .NET dan API yang sesuai.

4. Execution script (.cmd)
   Untuk Windows. Execution script dibuat untuk menangani command build dan run.

5. Execution script (.sh)
   Execution script untuk Linux/macOS.

Dalam menambahkan algoritma greedy pada bot, algoritma ini bertujuan untuk mendapatkan poin secara maksimal. Implementasi algoritma greedy bisa didasari berbagai jenis pendekatan atau strategi, antara lain dalam aspek kecepatan, jarak, energi, dan lain sebagainya.

2.5 Alur Menjalankan Program

1. Menjalankan game engine
   Game engine di-*run* dengan *command* sebagai berikut:

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

2. Melakukan setup konfigurasi booter
   Klik “Config” - “Bot Root Directories”, lalu masukkan directory yang berisi folder bot-bot.

3. Menjalankan battle
   Klik “Battle” - “Start Battle”. Bot-bot yang telah dimasukkan pada langkah setup konfigurasi akan muncul pada bagian “Bot Directories (local only)”.

4. Boot dan tambahkan bot yang ingin dipermainkan
   Pilih bot yang ingin dimainkan, lalu di-boot. Pilih bot-bot yang sudah di-boot, lalu ditambahkan sebagai battle participants.

5. Mulai permainan dengan klik “Start Battle”.

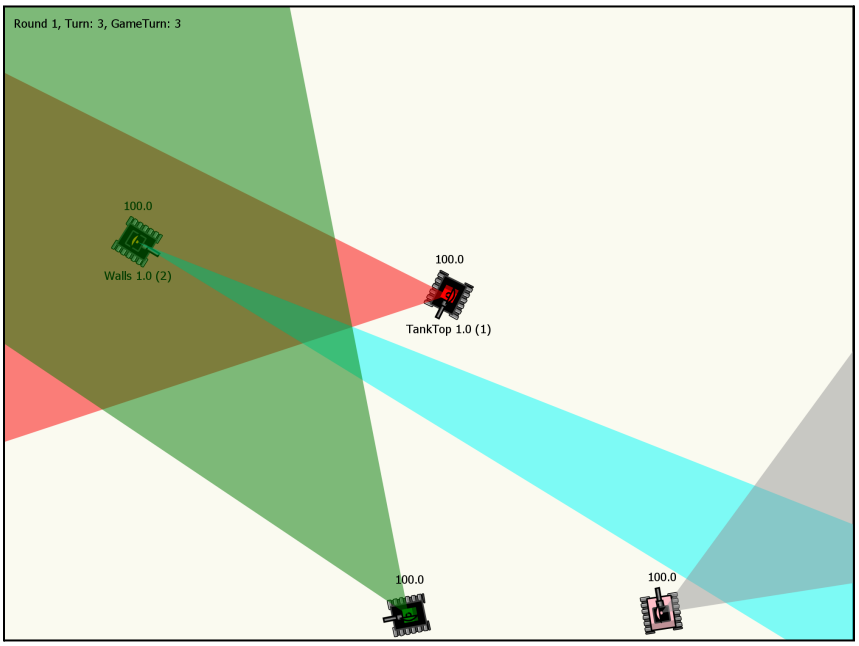
BAB III

APLIKASI STRATEGI GREEDY

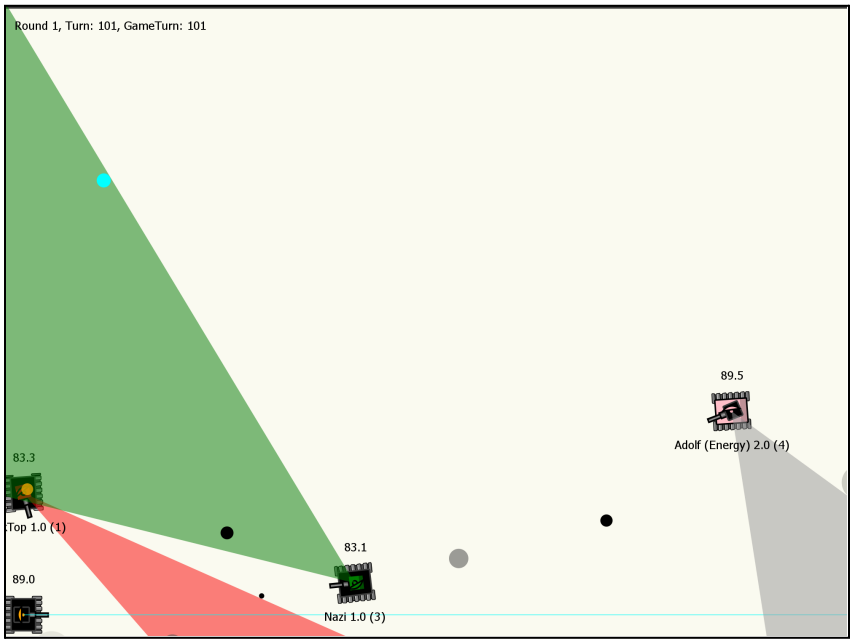
3.1 Eksplorasi Alternatif Solusi *Greedy*

3.1.1 Greedy by Distance

Greedy by Distance adalah pendekatan untuk mencari tank terdekat sebelum melakukan penyerangan untuk mengoptimalkan strategi. Strategi ini memanfaatkan jarak sebagai variabel inti untuk menyerang. Strategi ini tidak hanya optimal dalam penyerangan jarak jauh, tetapi juga sangat optimal dalam penyerangan jarak dekat (ramming).



Gambar 3.1.1.1 Ilustrasi bot (TankTop) melakukan scanning jarak terdekat



Gambar 3.1.1.2 Ilustrasi bot (TankTop) melakukan penyerangan terhadap Target (Walls) dengan jarak terdekat

a. Mapping Elemen *Greedy*:
   i. Himpunan Kandidat : distance, coordinate, speed , delta angle , energy, direction
   ii. Himpunan Solusi : distance, coordinate
   iii. Fungsi Solusi : tank lawan yang memiliki jarak terdekat dengan bot dan berada pada radius toleransi + 100, toleransi merupakan jarak terdekat terakhir kali tank melakukan scanning
   iv. Fungsi Seleksi : pilih tank lawan yang memiliki jarak terdekat dengan bot
   v. Fungsi Kelayakan: memeriksa apakah tank berada dalam radius toleransi, dan belum ada tank yang meninggal selama penyerangan dalam jarak radius
   vi. Fungsi Objektif : melakukan penyerangan agresif terhadap tank terdekat (shoot & ramming)

b. Analisis Efisiensi Solusi
   Dengan menggunakan strategi ini, bot akan mencari tank terdekat dengan koordinat yang sudah disimpan dalam sebuah class, untuk dikunjungi setelah selesai melakukan scanning 1 rotasi penuh. Strategi ini bergantung pada cara

pengurutan / *sorting* pencarian tank yang memiliki jarak terdekat dengan membandingkan setiap tank yang sudah di scan dengan jarak tank terendah pada saat itu. Pendekatan ini memiliki kompleksitas waktu algoritma:

$$T(n) = O(n)$$

n = jumlah tank yang terdeteksi setiap 1x rotasi scan

c. Analisis Efektivitas Solusi,
Strategi *Greedy by Distance* akan tereksekusi dengan baik apabila :

- Tank musuh berada dalam jangkauan scanning yaitu 1200 pixel
- Tidak ada tank lain yang berada terlalu dekat dalam radius *tolerance* + 100 pixel, dimana *tolerance* merupakan jarak terdekat tank terakhir kali bot melakukan scanning

Strategi *Greedy by Distance* akan tereksekusi dengan buruk apabila:

- Ketika tank lain selain tank terdekat yang berada dalam radius *tolerance* + 100, tepat setelah bot melakukan *scanning* 1 rotasi penuh / setelah tank_list berhasil di clear setelah 1 rotasi.
- Tank musuh berada diluar jangkauan scanning yaitu 1200 pixel
- Terdapat tank musuh yang meninggal akibat tank lain, pada saat tank bot melakukan penyerangan terhadap tank target (akan mereset tolerance ke default value 1200)

3.1.2 Greedy by Energy

Greedy by energy adalah pendekatan yang mengoptimalkan aksi dan pilihan berdasarkan energi, baik energi sendiri maupun target lawan. Strategi optimalisasi ini memanfaatkan energi untuk:

- Menyerang lebih agresif dengan cara mendekati target lawan saat memiliki keunggulan energi terhadap lawan.
- Mengatur kekuatan tembakan untuk efektivitas serangan maupun efisiensi dan penghematan energi.
- Meningkatkan prioritas gerakan menghindar saat energi rendah dan tidak memiliki keunggulan terhadap energi lawan.

a. Mapping Elemen Greedy:
   i. Himpunan kandidat: bot energy, target energy, firepower, distance, coordinate, movement, direction

   ii. Himpunan solusi: bot energy, target energy

   iii. Fungsi solusi: mengelola energi untuk menyesuaikan tembakan (firepower) dan movement (aggressive/defensive). Rasio keunggulan energi (energyAdvantageRatio) dipakai sebagai patokan keunggulan energi bot terhadap target.

   iv. Fungsi seleksi: memilih target tank lawan yang energinya rendah dan tidak unggul. terhadap bot untuk diprioritaskan.

   v. Fungsi kelayakan: Memeriksa apakah sisa energi cukup untuk melakukan tembakan tertentu. Memeriksa apakah rasio energy bot dan target mencukupi rasio keunggulan energi terhadap musuh, dengan *energyAdvantageRatio* = 1.6 agar bot menyesuaikan keputusan gerakan agresif/defensif.

   vi. Fungsi Objektif: memaksimalkan efektifitas serangan dan pertahanan dengan pemanfaatan *energyAdvantageRatio*, menyeimbangkan antara konservasi energi dan agresivitas berdasarkan kondisi energi. Tujuan utamanya adalah memaksimalkan damage per unit energi ketika memiliki keunggulan, dan memaksimalkan *survivability* dalam posisi energi tidak menguntungkan.

b. Analisis Efisiensi Solusi
   Dengan menggunakan strategi ini, bot akan menyesuaikan strategi penembakan dan pergerakan berdasarkan energi bot maupun energi lawan. Jika energi bot lebih tinggi dibanding energi lawan, maka energi bot memiliki keunggulan direpresentasikan dengan rasio $\frac{ownEnergy}{targetEnergy}$ yang digunakan sebagai pembanding terhadap *energyAdvantageRatio* sebesar 1.6, yang berarti bot dianggap unggul ketika energi bot 1.6x lebih besar dari energi lawan. Rasio ini digunakan sebagai dasar dalam menentukan strategi-strategi bot lainnya.

   Penargetan lawan dilakukan melalui scanning dan kemudian memproses energi dan jarak posisi target tersebut untuk diperiksa rasio keunggulannya. Jika bot unggul, firepower akan disesuaikan dengan mempertimbangkan heuristik jarak dengan tujuan menembak target yang lebih lemah dan dekat dengan firepower lebih besar guna meningkatkan peluang mengenai target dengan damage yang

tinggi sekaligus menghemat energi (karena menembak peluru besar membuang energi lebih banyak). Sebaliknya, untuk target jauh atau sangat energi bot menipis, tembakan tetap dilakukan dengan menurunkan firepower seolah-olah sebagai "*lifesteal*" guna mempertahankan energi lebih lama karena jika peluru berhasil mengenai target, energi bot akan kembali bertambah. Pendekatan strategi penembakan ini menerapkan prinsip *Greedy by energy* di mana konservasi energi menjadi faktor utama dalam strategi menyerang.

Greedy by energy juga diterapkan dalam strategi bertahan, dengan menyesuaikan tingkat agresif bot berdasarkan keunggulan energi. Jika bot memiliki keunggulan energi yang signifikan, bot akan lebih agresif dalam mendekati lawan dengan tujuan menekan pergerakan lawan dan memanfaatkan kesempatan menyerang dengan damage lebih besar. Sebaliknya, jika energi bot lebih rendah atau hampir abis, bot akan menghindari pertempuran jarak dekat, bergerak secara acak untuk mengurangi presisi tembakan musuh, serta menembak dengan firepower rendah agar tetap dapat bertahan lebih lama.

Selain itu, mekanisme gerakan bot juga dioptimalkan untuk efisiensi energi dengan melakukan kombinasi pergerakan dan perubahan arah acak setiap interval tertentu agar tidak mudah diprediksi lawan. Jika bot mendeteksi lawan dengan energi rendah dan lebih rendah dari energi sendiri, maka bot akan mendekati secara agresif terhadap lawan tersebut guna memaksimalkan peluang eliminasi lawan. Namun, jika lawan memiliki energi tinggi, bot akan lebih fokus dalam menjaga jarak dan menghindari serangan agar berada dalam posisi yang lebih menguntungkan sebelum melakukan keputusan serangan berikutnya. Dengan kombinasi strategi ini, bot mampu mempertahankan keseimbangan antara serangan dan pertahanan secara optimal berdasarkan kondisi dan keunggulan energi, dengan tambahan pertimbangan jarak untuk konservasi energi.

Perlu diperhatikan juga pada bot *Greedy by energy* yang telah dibuat, proses scanning bot tidak dilakukan secara menyeluruh terhadap semua lawan sebelum memilih target dengan energi paling rendah di seluruh arena. Sebaliknya, scanning dilakukan secara berkelanjutan dan langsung menargetkan lawan yang tidak unggul energinya terhadap energi bot. Pendekatan ini dipakai sebab rencana pengembangan bot dilakukan dengan pertimbangan pergerakan bot yang cepat dan acak sehingga heuristik energi akan optimal dengan scanning secara langsung dibanding melakukan scan secara menyeluruh sebelum memutuskan target yang ingin diserang.

Pendekatan ini membuat bot lebih adaptif dalam menyerang dan bertahan, memanfaatkan momen-momen ketika lawan dengan energi rendah terdeteksi tanpa harus menyaring seluruh musuh terlebih dahulu. Dengan demikian, strategi

ini memanfaatkan keunggulan energi, memungkinkan bot untuk menghabisi lawan lebih cepat tanpa risiko kehilangan terlalu banyak energi. Pendekatan ini memiliki kompleksitas waktu algoritma:

$$T(n) = O(1)$$

n = jumlah tank yang terdeteksi setiap 1x rotasi scan

c. Analisis Efektivitas Solusi
Strategi *Greedy by energy* akan tereksekusi dengan baik apabila:

- Lawan memiliki energi lebih rendah, dan bot dianggap unggul (energi 1.6x lebih besar musuh), dan memanfaatkan keunggulan tersebut untuk menyerang dengan menghabisi lawan lebih cepat.
- Target berada dalam jarak dekat untuk efisiensi penggunaan energi dan peningkatan firepower dengan konsumsi energi optimal.
- Jumlah lawan yang terdeteksi tidak terlalu banyak. Jika hanya ada sedikit lawan, maka bot dapat fokus menyerang dan menghabisi target yang lebih lemah tanpa harus sering berpindah target.

Strategi *Greedy by Energy* akan tereksekusi dengan buruk apabila:

- Lawan memiliki energi tinggi dan lebih unggul dibanding bot. Bot tidak dapat bermain secara agresif dan serangan balasan lawan akan lebih kuat.
- Jarak target terlalu jauh. Pada jarak yang jauh, akurasi tembakan berkurang dan akan mengkonsumsi energi secara boros jika tidak mengenai target.
- Banyak lawan yang harus diprioritaskan secara bersamaan, karena strategi agresif dan penghindaran bot difokuskan sesuai 1 target saja.

3.1.3 Greedy by Shooting Efficiency

Greedy by *Shooting Efficiency* adalah pendekatan untuk mencari tank dengan *composite value* terendah dimana , *composite value* adalah gabungan dari *delta angle* dan *distance* dengan rasio 1 : 6. Strategi ini memanfaatkan selisih sudut badan dan jarak tembak sebagai variabel inti untuk menyerang. Strategi ini sangat optimal dalam penembakan jarak jauh, dengan harapan strategi ini bukan hanya memaksimalkan keakurasian penembakan tetapi meminimalkan miss shoot yang lumayan fatal.

a. Mapping Elemen *Greedy*:
   i. Himpunan Kandidat : distance, coordinate, speed , delta angle, energy, direction
   ii. Himpunan Solusi : delta angle , distance
   iii. Fungsi Solusi : tank lawan yang memiliki selisih sudut tubuh terhadap bot dan jarak terpendek (*Composite Value*).
   iv. Fungsi Seleksi : pilih tank lawan yang memiliki delta angle terkecil dan jarak terdekat dengan bot
   v. Fungsi Kelayakan: memeriksa apakah tank berjarak kurang dari sama dengan 350 pixel untuk melakukan *shooting* dan memiliki *Composite Value* terkecil dalam sekali rotasi *scanning.*
   vi. Fungsi Objektif : melakukan penyerangan terstruktur terhadap tank dengan *CompositeValue* terkecil.

b. Analisis Efisiensi Solusi
   Algoritma ini mencari tank target yang memiliki selisih sudut badan dan jarak tembak terdekat dengan rasio 1:6. Rasio ini dipertimbangkan dengan maximum value dari delta angle dan distance dimana jangkauan tembak maximum setelah melakukan scanning adalah 1200 pixel dan normalized delta angle dengan range value [0-179]. Dalam hal ini kasus terburuk yang dapat dipilih terjauh adalah putaran sejauh 179 derajat dengan jarak 1200, dengan menggunakan rasio 1: 6, didapat jarak = 1200 dan value putaran = 1074. Dari *value* tersebut, didapat perbandingan ~ (1:1) untuk maksimum value , dimana pada pertimbangan rasio ini, adalah jika terdapat selisih sangat kecil setelah pengalian bilangan *integer k* terhadap value putaran, diantara kedua *value* diutamakan bobot terberat pada value jarak. Pendekatan ini memiliki kompleksitas waktu algoritma

$$T(n) = O(n)$$

n = jumlah tank yang terdeteksi setiap 1x rotasi scan(360^o)

c. Analisis Efektivitas Solusi
   Strategi *Greedy by Shooting Efficiency* akan tereksekusi dengan baik apabila :
   - Tank musuh berada dalam jangkauan scanning yaitu 1200 pixel
   - Tank musuh tidak melakukan pembelokan tajam dengan kecepatan tinggi

Strategi *Greedy by Shooting Efficiency* akan tereksekusi dengan buruk apabila:

- Ketika tank musuh berada diluar jangkauan scanning yaitu 1200 pixel
- Ketika tank memiliki kecepatan tinggi dan melakukan pembelokan tajam

3.1.4 Greedy by Potential Points

Strategi G*reedy by Potential Points* merupakan pendekatan yang memprioritaskan target berdasarkan nilai skor potensial yang bisa diperoleh dari membunuh setiap lawan dan bertahan hidup. Nilai ini dihitung berdasarkan HP lawan, potensi damage dari tembakan atau tabrakan (ramming), serta berbagai bonus skor, seperti *Bullet Damage*, *Ram Damage*, dan *Survival Bonus*. Dengan strategi ini, bot akan selalu memilih target dengan skor tertinggi pada saat itu, tanpa mempertimbangkan kondisi jangka panjang, sehingga berfokus pada pencapaian skor maksimum dalam waktu sesingkat mungkin.

a. Mapping Elemen *Greedy*:

   i. Himpunan kandidat: target energy, coordinate, distance, direction, score potential

   ii. Himpunan solusi: score potential

   iii. Fungsi solusi: memilih target dengan skor potensial tertinggi yang dapat memberikan poin maksimal

   iv. Fungsi seleksi: mengurutkan lawan berdasarkan nilai skor potensial, kemudian memilih yang tertinggi sebagai lockedTarget.

   v. Fungsi kelayakan: memastikan bahwa bot memiliki cukup energi untuk menyerang, atau jika energi rendah, memilih strategi menghindar untuk bertahan lebih lama.

   vi. Fungsi objektif: bot akan memaksimalisasi skor dengan cara menyerang lawan yang memberikan potensial poin tertinggi. Potensial skor tersebut dihitung dari penyeimbangan antara damage dari tembakan dan tabrakan (*ramming*). Setelah itu, aspek *survivability* juga diprioritaskan saat bot

memiliki energi rendah dengan tidak menembak dan fokus dalam menghindar untuk meningkatkan perolehan poin dari bertahan hidup.

b. Analisis Efisiensi Solusi

Dengan strategi ini, bot akan melakukan scan terhadap seluruh robot terlebih dahulu, dan mencari tank lawan dengan skor potensial (*ScorePotential*) tertinggi. Skor potensial tersebut dikomputasi berdasarkan energi atau HP lawan, *bullet damage,* dan *kill bonus* dari *bullet* maupun *ramming*. Efisiensi strategi ini ditentukan oleh berbagai faktor. Pertama, bot hanya perlu menentukan satu fokus berdasarkan target terbaik (*bestTarget*) saat itu. Hal ini mengurangi beban komputasi dalam pemilihan target, karena bot hanya perlu membandingkan skor potensial dari lawan yang terdeteksi dalam satu rotasi scanning.

Efisiensi pendekatan *Greedy by Potential Points* dalam penyerangan dapat dilihat ketika bot memprioritaskan target dengan HP rendah. Dengan pendekatan ini, bot akan melakukan penyerangan yang diprioritaskan untuk menghabiskan musuh yang energinya sudah rendah sehingga mendapatkan poin bonus lebih. Strategi ini memungkinkan bot untuk memaksimalkan damage dan penghabisan musuh dengan energi konsumsi energi minimum.

Jika lawan memiliki HP atau energi rendah (<10), maka lawan menjadi target potensial yang bisa dikalahkan. Jika lawan dapat dikalahkan dengan *ramming*, bot akan memanfaatkan *Ram Damage Bonus* sebesar 30% dari damage yang dibuat, sehingga meningkatkan perolehan skor. Jika menggunakan peluru, bot akan mendapatkan *Bullet Damage Bonus* sebesar 20% dari damage yang diberikan jika musuh berhasil dieliminasi.

Jika energi bot rendah dan tidak cukup untuk bertarung, bot akan beralih ke strategi menghindar dan memprioritaskan survival agar tetap memperoleh poin. Dengan bertahan lebih lama di arena, bot masih berpeluang mendapatkan Survival Score sebesar 50 poin setiap kali ada bot lain yang mati dan *Last Survival Bonus* sebesar 10 poin dikalikan jumlah musuh jika menjadi bot terakhir yang bertahan. Mekanisme bertahan ini memastikan bahwa bot tetap dapat mengumpulkan poin meskipun dalam kondisi energi yang tidak menguntungkan, memenuhi prinsip *Greedy by Potential Points*. Pendekatan ini memiliki kompleksitas waktu algoritma:

$$T(n) = O(n)$$

n = jumlah tank yang terdeteksi setiap 1x rotasi scan

c. Analisis Efektivitas Solusi

Strategi *Greedy by Potential Points* akan tereksekusi dengan baik apabila :

- Lawan dengan potensial skor tertinggi mudah dijangkau dan memiliki HP yang rendah, sehingga bot bisa langsung menyerang dan memperoleh poin maksimal

- Bot memiliki energi cukup untuk menyerang. Jika HP bot masih cukup, bot dapat menembak atau menabrak lawan dengan skor potensial tertinggi tanpa resiko habis energi.

Strategi *Greedy by Potential Points* akan tereksekusi dengan buruk apabila:

- Lawan dengan potensial skor tertinggi sulit dijangkau dan memiliki HP yang tinggi. Bot akan terlalu fokus terhadap satu target sehingga mengabaikan ancaman dan membuang energi secara tidak efektif.

- Bot memiliki energi rendah, sehingga hanya dapat kabur dan memiliki resiko mati terhadap lawan.

3.2 Strategi Greedy Terpilih

Berdasarkan variasi strategi greedy yang telah dijelaskan sebelumnya, terdapat kelebihan dan kekurangan dari masing-masing strategi, dikarenakan pada setiap ronde game memiliki *starting position* dan potensi terjadinya *error sending per turn* ke server. Maka dari itu kami menentukan tank bot dengan strategi yang paling optimal untuk menghadapi situasi tersebut dari 4 tank yang sudah kami buat.

Pada implementasi strategi tank kami, kami memutuskan untuk menggunakan TankTop sebagai tank utama yang mengimplementasikan dengan urutan prioritas dari tertinggi hingga terendah sebagai berikut:

1. Mengecek 360 derajat tank-tank sekitar

2. Melakukan sorting terhadap tank dari distance terkecil ke terbesar

3. Jika distance lebih jauh dari 120 pixel, tank menuju X,Y locked target (yaitu tank terdekat) sambil menembak menggunakan DistanceShoot().

4. Jika distance sudah lebih kecil dari 120 pixel, tank menembak dengan menggunakan fungsi AggressiveShoot() dan maju dengan kecepatan yang lebih cepat (ramming) dengan radius tolerance + 100, dimana tolerance adalah jarak tank terdekat sebelumnya jika ada.

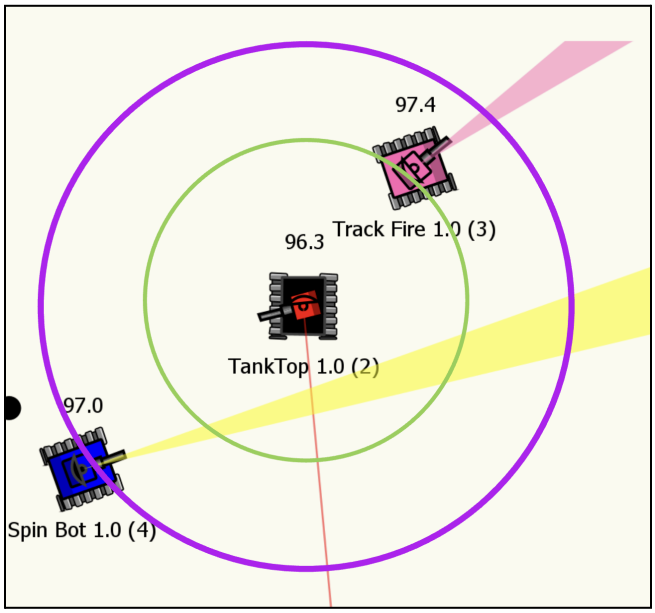
5. Jika terdapat tank yang meninggal selama penembakan, tolerance akan di reset ke default state 1200.

6. Jika selama proses greedy terdapat tank yang menabrak, scan dihentikan dan tank berbalik ke arah penabrak dan melakukan ramming.

Ketika round dimulai , TankTop akan melakukan scanning 1 putaran penuh atau 360 derajat dan melakukan *storing* ke dalam suatu array of class yaitu EnemyInfo. Kemudian dari array tersebut, diambil komponen-komponen seperti distance, direction, X, Y. Dari informasi tersebut, TankTop kemudian melakukan sorting dalam fungsi sortPriorityTankDistance() , dimana pada fungsi ini, dilakukan sorting berdasarkan jarak saat melakukan scanning, dan mereturn tank info dengan jarak terendah. Setelah TankTop mendapatkan informasi tank, TankTop akan menuju koordinat dari lawan dengan distance terdekat hasil sorting, dengan menuju ke koordinat X, Y.

Ketika ternyata tank target memiliki jarak lebih dari 120 pixel, maka TankTop akan menembak dengan menggunakan fungsi DistanceShoot(), dimana pada fungsi ini, TankTop akan membesarkan Fire Power jika tank cukup dekat, dan berlaku sebaliknya jika tank cukup jauh, dalam kasus ini penembakan dilakukan dengan pendekatan *Greedy by distance Shooting*. Tetapi jika lockedTank memiliki jarak di bawah 120 pixel, maka TankTop akan menembak dengan menggunakan fungsi AggressiveShoot(), dimana AggressiveShoot() ini melakukan penembakan dengan Fire Power maksimum yaitu 3, sambil melihat kondisi energy sendiri, jika tidak memungkinkan menembak dengan FirePower 3 dengan jarak di bawah 120 pixel, maka tank akan menurunkan tembakannya, dalam kasus ini penembakan terjadi dengan pengimplementasian *Greedy by Self Energy Shooting*.

Terdapat kasus-kasus tertentu dimana ada kemungkinan TankTop melakukan penyerangan terhadap tank yang bukan memiliki jarak terpendek. Hal ini disebabkan oleh *error difference* pada radius toleransi total , dimana pada kasus ini , tank lain berada pada radius toleransi total, yang mana toleransi total merupakan jarak terpendek terakhir kali ditambah dengan 100 pixel untuk mencegah kasus tank tidak terscan lawan akibat selisih waktu pergantian turn. Variable tolerance sendiri diciptakan dengan tujuan agar TankTop tidak melakukan penyerangan terhadap tank yang berada diluar radius terakhir tank terdekat (mempertahankan greedy by distance). Jadi diciptakan batas radius batas penyerangan setelah *scanning*. Toleransi total adalah tolerance + 100, 100 untuk mencegah tank gagal melakukan scanning terhadap sekitar akibat batas yang terbuat. Pencegahan itulah yang menciptakan potensi *miss* dalam penyerangan tank dengan *distance* terendah.



Gambar 3.2.1 Ilustrasi *tolerance* lingkaran hijau dan *total tolerance* lingkaran ungu

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Bot dalam *Pseudocode*

Greedy by Distance

```
Class TankTop : Bot
    turn_direction : integer
    tolerance : double
    error_count : integer
    tank_list : array of EnemyInfo
    lockedTarget : EnemyInfo
    previousLockedTarget : EnemyInfo

    Main:
        Create a new TankTop instance and call Start()

    Constructor:
        Initialize the base class with configuration file "TankTop.json"

    Procedure Run:
        tolerance ← 1200
```

```
            error_count ← 0
            EnemyInfo.NEFF ← 0
            BodyColor ← Red? (or set color as defined)
            TurretColor ← Red
            RadarColor ← Red
            BulletColor ← Black
            ScanColor ← Red
            AdjustGunForBodyTurn ← true
            lockedTarget ← null
            previousLockedTarget ← null
            tank_list ← new array of EnemyInfo (size 1000)

            While IsRunning do:
                TurnRadarRight(360)
                SetRescan()
                ClearList(tank_list)
            End While

        Procedure OnScannedBot(event):
            If error_count equals 2 then:
                previousLockedTarget ← null
                error_count ← 0
            End If

            newTank ← new EnemyInfo(event.X, event.Y, DistanceTo(event.X,
ent.Y), event.Direction)
            tank_list[EnemyInfo.NEFF] ← newTank
            Increment EnemyInfo.NEFF

            lockedTarget ← sortPriorityTankDistance(tank_list)

            If previousLockedTarget is not null then:
                If absolute(DistanceTo(previousLockedTarget.X,
eviousLockedTarget.Y) -          DistanceTo(lockedTarget.X, lockedTarget.Y))
tolerance then:
                    error_count ← error_count + 1
                    Return from procedure
                End If
            Else:
                previousLockedTarget ← lockedTarget
                Return from procedure
            End If

            tolerance ← (if DistanceTo(previousLockedTarget.X,
eviousLockedTarget.Y) < DistanceTo(lockedTarget.X, lockedTarget.Y)
                        then DistanceTo(previousLockedTarget.X,
eviousLockedTarget.Y) + 100
                        else DistanceTo(lockedTarget.X, lockedTarget.Y) +
0)
            previousLockedTarget ← lockedTarget

            If lockedTarget is null then:
                Return from procedure
            End If
```

```
            // Greedy predictive angle movement
            heading ← lockedTarget.direction
            gunRotationAdjustment ← 15
            distance ← DistanceTo(lockedTarget.X, lockedTarget.Y)

            SetTurnLeft(BearingTo(lockedTarget.X, lockedTarget.Y))

            If heading > 15 then:
                SetTurnGunLeft(GunBearingTo(lockedTarget.X -
nRotationAdjustment, lockedTarget.Y))
            Else If heading < -15 then:
                SetTurnGunLeft(GunBearingTo(lockedTarget.X +
nRotationAdjustment, lockedTarget.Y))
            Else:
                SetTurnGunLeft(GunBearingTo(lockedTarget.X, lockedTarget.Y))
            End If

            SmartShoot(Energy, distance)

            // Movement mechanics
            If distance ≤ 120 then:
                If distance > 80 then:
                    SetForward(500)
                    SetTurnLeft(turn_direction * 1000)
                Else:
                    SetForward(1000)
                End If
            Else:
                SetForward(distance / 2.5)
            End If

        Procedure OnHitBot(event):
            SetTurnGunLeft(GunBearingTo(event.X, event.Y))
            AggressiveShoot(Energy)
            SetForward(40)

        Procedure AggressiveShoot(health):
            If health > 16 then:
                SetFire(3)
            Else If health > 10 then:
                SetFire(2)
            Else If health > 4 then:
                SetFire(1)
            Else If health > 2 then:
                SetFire(0.5)
            Else If health > 0.4 then:
                SetFire(0.1)
            End If

        Procedure DistanceShoot(distance):
            distanceAdjustedShoot ← 3.0 - (0.008 * distance)
            SetFire(distanceAdjustedShoot)

        Procedure SmartShoot(health, distance):
            If distance < 20 then:
```

```
            AggressiveShoot(health)
        Else:
            DistanceShoot(distance)
        End If

    Procedure ClearList(list):
        For i from 0 to EnemyInfo.NEFF - 1 do:
            list[i] ← null
        End For
        EnemyInfo.NEFF ← 0

    Procedure sortPriorityTankDistance(list):
        find_tank ← null
        temp ← maximum possible double value
        For i from 0 to EnemyInfo.NEFF - 1 do:
            If list[i] is not null and list[i].distance < temp then:
                temp ← list[i].distance
                find_tank ← list[i]
            End If
        End For
        Return find_tank

    Procedure OnBotDeath(event):
        tolerance ← 1200
        previousLockedTarget ← null

    Procedure OnHitWall(event):
        turn_direction ← turn_direction * -1
        SetBack(-200)

End Class

Class EnemyInfo
    Static NEFF : integer = 0
    X : double
    Y : double
    distance : double
    direction : double

    Constructor(x, y, dist, direct):
        X ← x
        Y ← y
        distance ← dist
        direction ← direct
End Class
:
```

Greedy by Energy

```
Class Adolf : Bot
    Private variables:
        turnDirection: integer
        movementCounter: integer
```

```
        movingForward: boolean
        energyAdvantageRatio: constant double <- 1.6

    Main:
        Create new Adolf instance and call Start()

    Constructor:
        Initialize base class with config file "Adolf.json"

    procedure Run:
        AdjustRadarForBodyTurn <- true
        AdjustGunForBodyTurn <- true
        AdjustRadarForGunTurn <- true

        BodyColor <- Black
        TurretColor <- Black
        RadarColor <- Black
        BulletColor <- Black
        ScanColor <- Black
        GunColor <- Black
        TracksColor <- Black

        movingForward <- true

        while IsRunning
            TurnRadarRight(360)
        end while

    procedure OnScannedBot(event):
        targetX <- event.X
        targetY <- event.Y
        targetEnergy <- event.Energy
        distance <- CalculateDistanceTo(targetX, targetY)
        bearingFromGun <- CalculateGunBearingTo(targetX, targetY)

        movementCounter <- movementCounter + 1
        if movementCounter mod 10 = 0 then
            turnDirection <- random(0 or 1) * 2 - 1
        end if

        if event.Energy < 20 and Energy > event.Energy *
energyAdvantageRatio then
            TurnToFaceTarget(targetX, targetY, event.Direction)
            SetForward(150 + random(0 to 60))
        else
            SetTurnLeft(40 * turnDirection)
            SetForward(100 + random(0 to 60))
        end if

        TurnGunLeft(bearingFromGun)
        OptimizedEnergyShoot(Energy, targetEnergy, distance)
```

```
        SetRescan()

    procedure OnHitBot(event);

    procedure OnHitWall(event):
        ReverseDirection()

    procedure TurnToFaceTarget(targetX, targetY, Direction):
        bearing <- CalculateBearingTo(targetX, targetY)
        if Direction > 180 and Direction < 360 then
            turnDirection <- 1
        else if Direction < 180 and Direction > 0 then
            turnDirection <- -1
        end if
        TurnLeft(bearing)

    procedure OptimizedEnergyShoot(ourEnergy, targetEnergy,
distance):
        energyRatio <- ourEnergy / targetEnergy

        if energyRatio >= energyAdvantageRatio and distance < 150
then
            SetFire(minimum(2.5, ourEnergy / 4))
        else if targetEnergy < 15 and distance < 100 then
            SetFire(minimum(2.8, ourEnergy / 5))
        else if ourEnergy > 20 then
            if distance < 200 then
                SetFire(2)
            else if distance < 300 then
                SetFire(1.8)
            else
                SetFire(1.5)
            end if
        else if ourEnergy > 10 then
            SetFire(1.5)
        else if ourEnergy > 5 then
            SetFire(1)
        else if ourEnergy > 2 then
            SetFire(0.5)
        else if ourEnergy > 1 then
            SetFire(0.3)
        else if ourEnergy > 0.3 then
            if targetEnergy < 4 then
                SetFire(0.3)
            else
                SetFire(0.1)
            end if
        end if

    procedure ReverseDirection:
```

```
            if movingForward then
                SetBack(40000)
                movingForward <- false
            else
                SetForward(40000)
                movingForward <- true
            end if
    end class
```

Greedy by Shooting Efficiency

```
    Class Nazi : Bot
        tank_list : array of EnemyInfo
        allScanned : boolean
        scanned_count : integer
        lockedTarget : EnemyInfo

        Main:
            Create a new Nazi instance and call Start()

        Constructor:
            Initialize the base class with configuration file "Nazi.json"

        Procedure Run:
            allScanned ← false
            EnemyInfo.NEFF ← 0
            BodyColor ← Black
            TurretColor ← Green
            RadarColor ← Green
            BulletColor ← Black
            ScanColor ← Green
            AdjustGunForBodyTurn ← true
            lockedTarget ← null
            tank_list ← new array of EnemyInfo (size 1000)

            While IsRunning do:
                TurnRadarRight(360)
                SetRescan()
            End While

        Procedure OnScannedBot(event):
            If allScanned equals false then:
                newTank ← new EnemyInfo(event.X, event.Y,
    DistanceTo(event.X, event.Y), DirectionTo(event.X, event.Y))
                tank_list[EnemyInfo.NEFF] ← newTank
                Increment EnemyInfo.NEFF
                scanned_count ← scanned_count + 1
                If scanned_count > EnemyCount - 1 then:
                    scanned_count ← 0
                    allScanned ← true
                End If
                Return from procedure
            End If
```

```
            lockedTarget ← sortCompositeValue(tank_list)
            distance ← DistanceTo(lockedTarget.X, lockedTarget.Y)
            gunBearing ← GunBearingTo(lockedTarget.X, lockedTarget.Y)

            SetTurnLeft(BearingTo(lockedTarget.X, lockedTarget.Y))
            SetTurnGunLeft(gunBearing)

            SmartShoot(Energy, distance)
            ClearList(tank_list)
            allScanned ← false

            // Movement Mechanics
            If distance ≤ 350 then:
                If distance > 300 then:
                    SetForward(10)
                Else:
                    SetBack(25)
                End If
            Else:
                SetForward(distance / 3.5)
            End If

        Procedure OnHitBot(event):
            SetTurnLeft(500)
            SetBack(500)

        Procedure OnHitWall(event):
            SetForward(500)

        Procedure EnergyAdjustedShoot(health):
            If health > 1.0 then:
                SetFire(0.8)
            Else:
                SetFire(0.2)
            End If

        Procedure DistanceShoot(distance):
            adjustShoot ← distance / 140
            SetFire(3.0 - adjustShoot)

        Procedure sortCompositeValue(list):
            find_tank ← null
            temp ← maximum possible double value
            For i from 0 to EnemyInfo.NEFF - 1 do:
                If list[i] is not null and list[i].CompositeValue < temp
    then:
                    find_tank ← list[i]
                    temp ← list[i].CompositeValue
                End If
            End For
            Return find_tank

        Procedure ClearList(list):
            For i from 0 to EnemyInfo.NEFF - 1 do:
                list[i] ← null
```

```
            End For
            EnemyInfo.NEFF ← 0

        Procedure SmartShoot(health, distance):
            If health < 3 then:
                EnergyAdjustedShoot(health)
            Else:
                DistanceShoot(distance)
            End If

    End Class

    Class EnemyInfo
        Static NEFF : integer = 0
        X : double
        Y : double
        distance : double
        deltaAngle : double
        CompositeValue : double

        Constructor(x, y, dist, delta_angle):
            X ← x
            Y ← y
            distance ← dist
            deltaAngle ← delta_angle mod 180
            CompositeValue ← distance + (6 * deltaAngle)

    End Class
```

Greedy by Potential Points

```
Class EnemyInfo
   Constants:
         static variable NEFF = 0
         variables X, Y, HP, Distance, ScorePotential, Direction

   Constructor:
         X <- x
         Y <- y
         HP <- hp
         Distance <- distance
         Direction <- direction
         ScorePotential <- 0

{Main tank bot class}
Class BlitzKrieg4 : Bot
    Private variables:
         bestTarget: EnemyInfo
         lockedTarget: EnemyInfo
         tank_list: EnemyInfo[1000]

    Main:
         Create new GreedyBastard instance and call Start()
```

```
    Constructor:
        Initialize base class with config file "GreedyBastard.json"

    procedure Run:
        AdjustRadarForBodyTurn <- true
        AdjustGunForBodyTurn <- true
        AdjustRadarForGunTurn <- true
        BodyColor <- Purple

        Initialize tank_list with size 1000

        while IsRunning
            ClearList(tank_list)
            TurnRadarRight(360)
            lockedTarget <- sortPriorityScorePotential(tank_list)

            if lockedTarget exists then
                if Energy < 10 AND Energy < lockedTarget.HP then
                    evadeSurvival()
                else
                    AttackTarget(lockedTarget)
                end if
            end if

            TurnRadarRight(360)
        end while

    procedure OnScannedBot(event):
        targetX <- event.X
        targetY <- event.Y
        targetEnergy <- event.Energy
        direction <- event.Direction
        distance <- CalculateDistanceTo(targetX, targetY)
        gunBearing <- CalculateGunBearingTo(targetX, targetY)

        TurnGunLeft(gunBearing)

        Create enemy = new EnemyInfo(targetX,targetY,
targetEnergy,distance,direction
        ComputeScorePotential(enemy)

        if NEFF < tank_list length then
            add enemy to tank_list at index NEFF
            NEFF <- NEFF + 1
        end if

        if lockedTarget exists then
            lockedTarget <- enemy
        end if
```

```
            if bestTarget is null or enemy.ScorePotential
    > bestTarget.ScorePotential then
                  bestTarget <- enemy
            end if

            if bestTarget exists and lockedTarget is null then
                  if Energy < 10 and Energy < bestTarget.HP then
                        evadeSurvival()
                  else
                        AttackTarget(bestTarget)
                  end if
            end if

            SetRescan()

      procedure OnHitBot(event):
            MoveBack(100)
            TurnRight(45)
            MoveForward(150)

      procedure OnHitWall(event):
            MoveBack(50)
            TurnRight(90)
            MoveForward(100)

      procedure ComputeScorePotential(enemy):
            bulletDamage <- 100 - enemy.HP
            ramDamage <- 2 * (100 - enemy.HP)

            if enemy.HP < 10 then
                  bulletKillBonus <- 0.2 * bulletDamage
                  ramKillBonus <- 0.3 * ramDamage
            else
                  bulletKillBonus <- 0
                  ramKillBonus <- 0
            end if

            enemy.ScorePotential <- bulletDamage+ramDamage+
    bulletKillBonus+ramKillBonus

      procedure ClearList(list):
            for i = 0...NEFF - 1
                  list[i] <- null
            end for
            NEFF <- 0

      function sortPriorityScorePotential(list):
            if NEFF <= 0 then
                  -> null
            end if
```

```
            highestScoreTank <- list[0]
            for i = 1...NEFF - 1
                  if list[i].ScorePotential >
    highestScoreTank.ScorePotential then
                        highestScoreTank <- list[i]
                  end if
            end for

            -> highestScoreTank

      procedure AttackTarget(target):
            distance <- target.Distance

            if target.HP > 10 then
                  TurnToFaceTarget(target.X, target.Y, target.Direction)
                  MoveForward(80)
            else
               TurnToFaceTarget(target.X, target.Y, target.Direction)
               MoveForward(300)
            end if

            TurnToFaceTarget(target.X, target.Y, target.Direction)

            if Energy > 20 then
                  if distance < 50 and target.HP < 20 then
                        Fire(3)
                  else if distance < 100 and target.HP > 20 then
                        Fire(2.8)
                  else if distance < 100 and target.HP < 10 then
                        Fire(2.5)
                  else
                        Fire(2)
                  end if
            end if

            bestTarget <- null

      procedure TurnToFaceTarget(targetX, targetY, Direction):
            bearing <- CalculateBearingTo(targetX, targetY)
            TurnLeft(bearing)

      procedure evadeSurvival:
            direction <- random(0 or 1) * 2 - 1
            TurnLeft(90 * direction + random(-20 to 20))
            MoveForward(300)
    end class
```

4.2 Struktur Data, Fungsi, dan Prosedur yang Digunakan

Program bot memanfaatkan beberapa struktur data inti untuk mendukung strategi greedy berdasarkan jarak musuh terdekat dan penguncian target. Struktur-struktur ini dibentuk dari kombinasi atribut, objek, serta integrasi dengan API Robocode Tank Royale.

Bagian ini memuat pembahasan kelas-kelas yang terdapat dalam program bot setelah mengetahui konfigurasi terkait medan permainan Robocode Tank Royale serta interaksi bot dengan arena melalui API. Dalam pengembangan bot, kami menerapkan pendekatan pemrograman berorientasi objek (OOP), di mana variabel dan data yang berkaitan dengan kondisi permainan dienkapsulasi ke dalam kelas-kelas tertentu. Selain itu, kami menambahkan beberapa metode tambahan pada kelas bot yang kami rancang untuk mendukung implementasi strategi greedy.

Secara umum, semua bot kami merupakan turunan dari class Bot milik API Robocode Tank Royale. Dalam program, terdapat beberapa kelas penting yang digunakan secara umum pada seluruh bot, yaitu kelas bot itu sendiri (misal TankTop, Nazi, Adolf, GreedyBastard) serta class tambahan seperti EnemyInfo. Kelas-kelas ini menyediakan data penting seperti posisi, energi, dan arah bot musuh yang akan digunakan sebagai dasar pengambilan keputusan greedy.

1. Class Bot (Super Class)

   Kelas ini menyediakan fungsi-fungsi penting seperti pergerakan bot, pengendalian radar, menembak, dan event handler. Beberapa fungsi yang dimiliki kelas ini, antara lain,

   a. TurnRadarRight(degree): Menggerakkan radar sejauh derajat tertentu.
   b. SetForward(distance), SetBack(distance): Mengatur gerakan maju atau mundur.
   c. TurnLeft(degree), TurnRight(degree): Mengatur arah gerakan.
   d. Fire(power): Menembakkan peluru dengan kekuatan tertentu.
   e. SetRescan(): Memerintahkan radar untuk melakukan pemindaian ulang.
   f. Event handler: OnScannedBot(), OnHitWall(), OnHitBot().

2. Class Bot TankTop, Adolf, Nazi, GreedyBastard

   Setiap bot memiliki implementasi strategi greedy berbeda, namun memiliki struktur data internal yang serupa sebagai turunan dari Class Bot. Berikut beberapa atribut umum yang dimiliki keempat kelas ini,

   a. turnDirection (integer): Arah rotasi radar atau badan bot.
   b. movingForward (boolean): Menentukan apakah bot sedang bergerak maju.

c. movementCounter (integer): *Counter* untuk mengatur pola gerak.
d. lockedTarget, bestTarget (EnemyInfo): Target musuh yang sedang dikunci atau menjadi prioritas serangan.
e. tank_list (EnemyInfo[]): Array berisi musuh-musuh yang terdeteksi dalam satu siklus radar, digunakan dalam proses seleksi target secara greedy.
f. Konstanta strategi: Contohnya, energyAdvantageRatio.

Selain itu, kelas ini juga memiliki beberapa metode umum yang serupa, antara lain,

a. Run(): Fungsi utama yang dieksekusi saat bot aktif.
b. OnScannedBot(event): Handler ketika musuh terdeteksi, proses utama algoritma greedy.
c. OnHitWall(event), OnHitBot(event): Handler ketika bot bertabrakan dengan dinding atau musuh.
d. TurnToFaceTarget(), AttackTarget(), evadeSurvival(), dll.: Fungsi tambahan sesuai strategi bot masing-masing.

3. Class EnemyInfo

Kelas ini bersifat tambahan, untuk menyimpan informasi bot musuh secara efisien agar dapat diproses dalam algoritma greedy, khususnya dalam bot yang menggunakan algoritma greedy berdasarkan poin potensial. Berikut beberapa atribut yang dimiliki kelas ini,

a. X, Y (double): Posisi musuh.
b. deltaAngle (double) : selisih sudut badan bot dengan koordinat musuh.
c. HP (double) : Energi musuh saat ini.
d. Distance (double) : Jarak musuh dari bot.
e. Direction (double) : Arah gerak musuh.
f. ScorePotential (double) : Estimasi skor yang dapat diperoleh jika menyerang musuh ini (digunakan dalam strategi greedy by score).
g. static NEFF (int) : Static variable untuk menghitung jumlah efektif elemen pada tank_list.

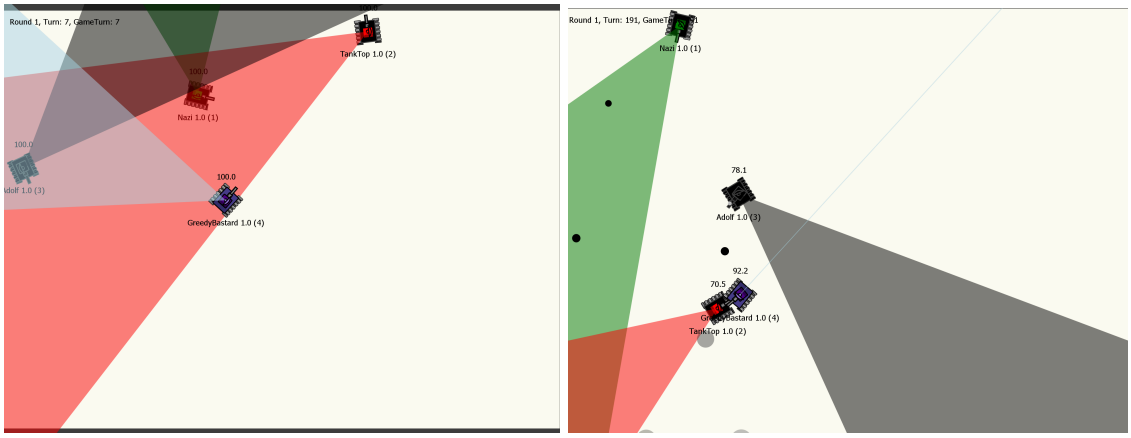
Kelas ini juga memiliki metode khusus, yaitu ComputeScorePotential() yang menghitung potensi skor dari musuh berdasarkan *damage* dan bonus.

4.3 Analisis dan Pengujian

Kami melakukan evaluasi dan pengujian permainan dengan mengintegrasikan bot kami, bersama dengan sample tank yang telah kami perbarui sedikit. Setelah menjalankan

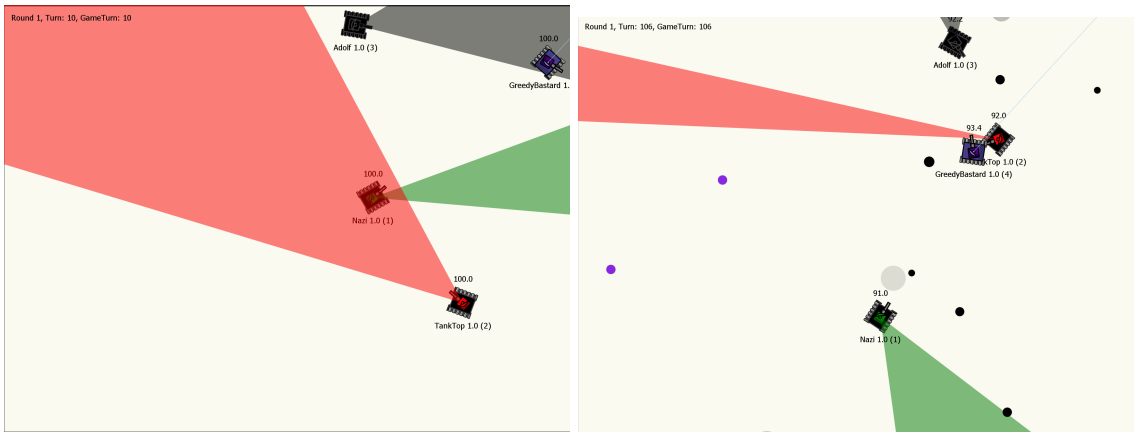
permainan, kami secara cermat memantau perilaku dan semua tindakan yang dilakukan oleh tank kami menggunakan visualisasi yang tersedia.

Berikut adalah analisis yang kami lakukan terhadap suatu pertandingan bot kami:



Gambar 4.3.1 Pengujian strategi algoritma *greedy by distance*

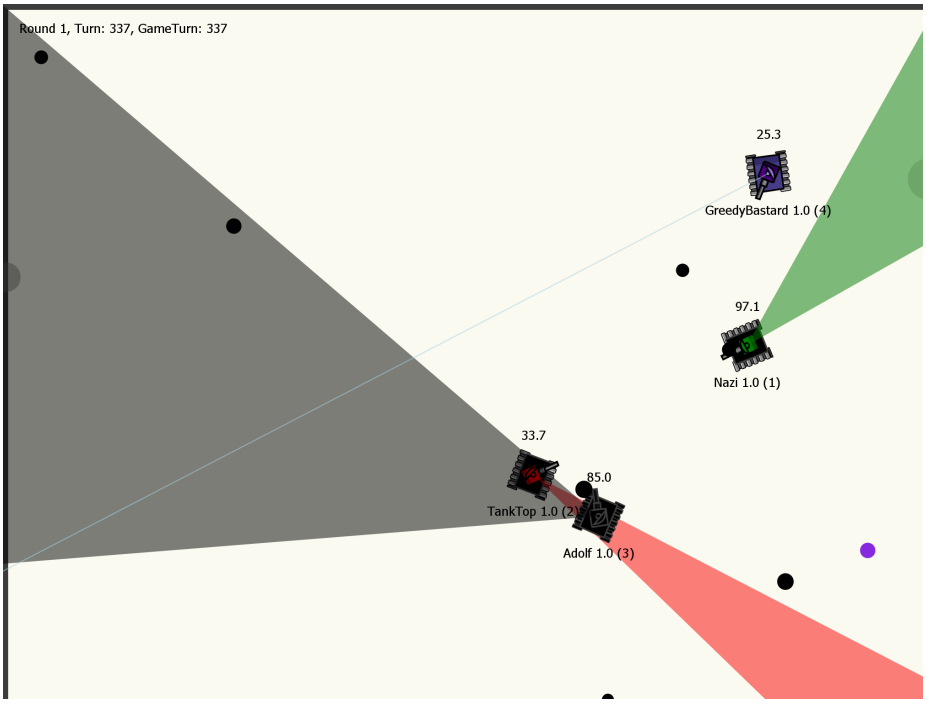
Berdasarkan gambar 4.3.1 dapat dilihat bahwa bot permainan yang telah dikembangkan menggunakan algoritma *greedy by distance* telah bergerak sesuai dengan harapan. Pada gambar 4.3.1 dapat dilihat bahwa tank melakukan scan 1 putaran penuh dan bergerak mengincar bot terdekat yaitu Nazi. Tetapi pada saat pengincaran, GreedyBastard mendekat , sehingga target penyerangan berubah.



Gambar 4.3.2 Pengujian strategi algoritma *greedy by shooting efficiency*

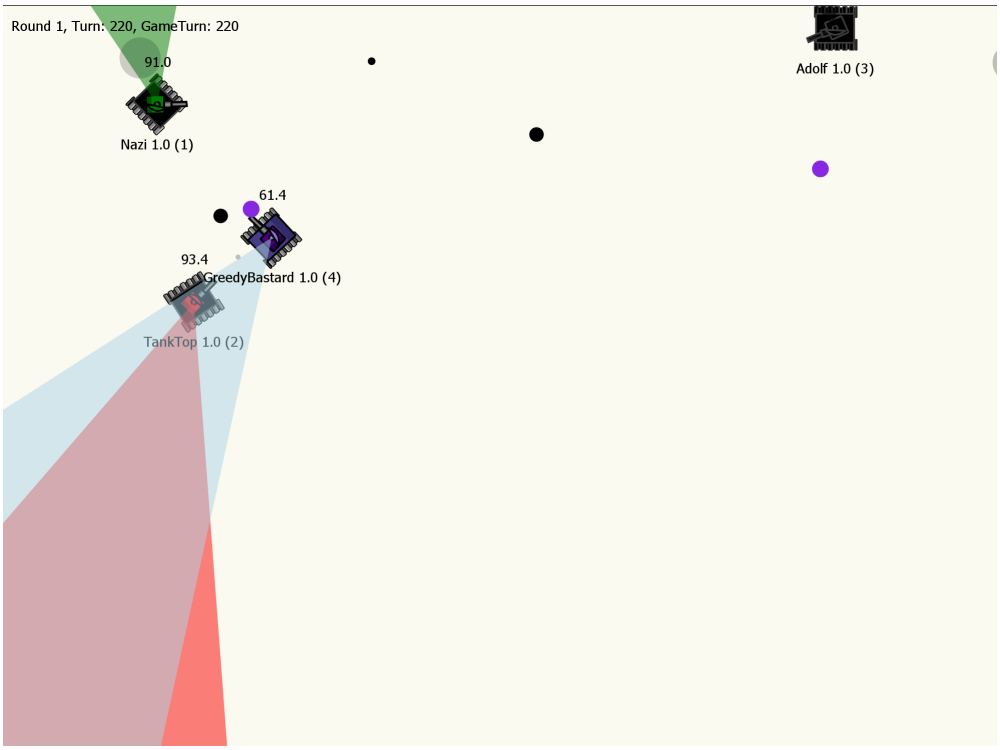
Berdasarkan gambar 4.3.2 dapat dilihat bahwa bot permainan yang telah dikembangkan menggunakan algoritma *greedy by shooting efficiency* telah melakukan aksinya sesuai harapan. Setelah melakukan scan penuh, Nazi menjaga jarak dengan tank lain sambil

mengukur keefisienan penembakan dengan menggunakan composite value dan menembak dari jarak yang lumayan jauh.



Gambar 4.3.3 Pengujian strategi algoritma *greedy by Energy*

Berdasarkan gambar 4.3.3 dapat dilihat bahwa bot permainan yang telah dikembangkan menggunakan algoritma *greedy by shooting Energy* telah melakukan aksinya sesuai harapan. Sambil melakukan gerakan mekaniknya, Adolf melakukan penyerangan berdasarkan Energinya dan Energi musuh dengan konstanta pengali 1,6x, dimana pada kondisi ini TankTop dengan nyawanya yang rendah dilakukan penyerangan dan ramming oleh Adolf setelah nyawa TankTop menipis dengan jarak yang sangat minim.



Gambar 4.3.4 Pengujian strategi algoritma *greedy by Potential Point*

Berdasarkan gambar 4.3.4 dapat dilihat bahwa bot permainan yang telah dikembangkan menggunakan algoritma *greedy by potential point* telah bergerak sesuai dengan harapan. Pada gambar 4.3.4 dapat dilihat bahwa tank GreedyBastard melakukan penyerangan terhadap Nazi yang memiliki jarak dekat dengan damage bullet yang besar. Hal ini disebabkan oleh *advantage points* yang dapat diperoleh oleh GreedyBastard dalam keputusannya untuk menyerang dengan FirePower yang lebih besar dalam jarak yang minim.

Tabel 4.3.1 Tabel Perbandingan *greedy* bot berdasarkan jumlah poin dengan 10 round setiap match

Match	TankTop	Nazi	Adolf	GreedyBastard
1	2330	1250	1654	615
2	2597	2214	1397	862
3	2006	1185	2250	981
4	2762	1680	1556	613
5	2621	1018	1605	996
6	2532	1200	2236	1321

7	2710	1432	2351	1215
8	2360	1132	2482	715
9	2164	1571	2092	1352
10	2095	1543	2154	1621
Average	2417.7	1422.5	1977.7	1029.1
STDEV	244.16	367.65	382.26	341.48
Total Win	7	0	3	0

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dalam tugas besar ini, kami berhasil mengembangkan beberapa bot Robocode Tank Royale menggunakan pendekatan algoritma greedy. Setiap bot menerapkan algoritma greedy yang berbeda, dengan tujuan untuk memaksimalkan skor dalam *battle* melalui pemilihan keputusan terbaik di setiap situasi berdasarkan informasi lokal. Melalui pengujian, dapat disimpulkan bahwa pendekatan dengan algoritma greedy cukup efektif untuk menghasilkan performa bot yang kompetitif, meskipun memiliki keterbatasan dalam menghadapi kondisi yang tidak terduga atau musuh yang menerapkan strategi lebih kompleks. Secara umum, pendekatan ini mampu memberikan solusi yang cepat dan efisien dalam pengambilan keputusan selama permainan berlangsung.

5.2 Saran

Untuk pengembangan bot Robocode kedepannya, disarankan agar pendekatan algoritma greedy yang digunakan dapat dieksplorasi secara lebih luas. Hal ini mencakup pengembangan heuristik yang lebih adaptif dan penyusunan strategi yang mampu menangani berbagai situasi permainan dengan lebih optimal. Selain itu, penting untuk melakukan penemuan lebih banyak *counterexample* atau skenario khusus yang dapat menguji batas efektivitas strategi greedy yang diterapkan. Dengan pemahaman yang lebih baik terhadap kondisi di mana strategi greedy berhasil ataupun gagal, pengembangan bot dapat diarahkan untuk menghasilkan solusi greedy yang lebih kuat, konsisten, dan efisien.

LAMPIRAN

Link Repository GitHub
https://github.com/brii26/Tubes1_Blitzkrieg

Link Video Tugas Besar
https://youtu.be/aaM3yIX5FFA

Tabel Penyelesaian Tugas Besar

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

R. Munir, “Algoritma Greedy (Bagian 1),” IF2211 Strategi Algoritma, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(202 1)-Bag1.pdf. [Accessed: Mar. 17, 2025].