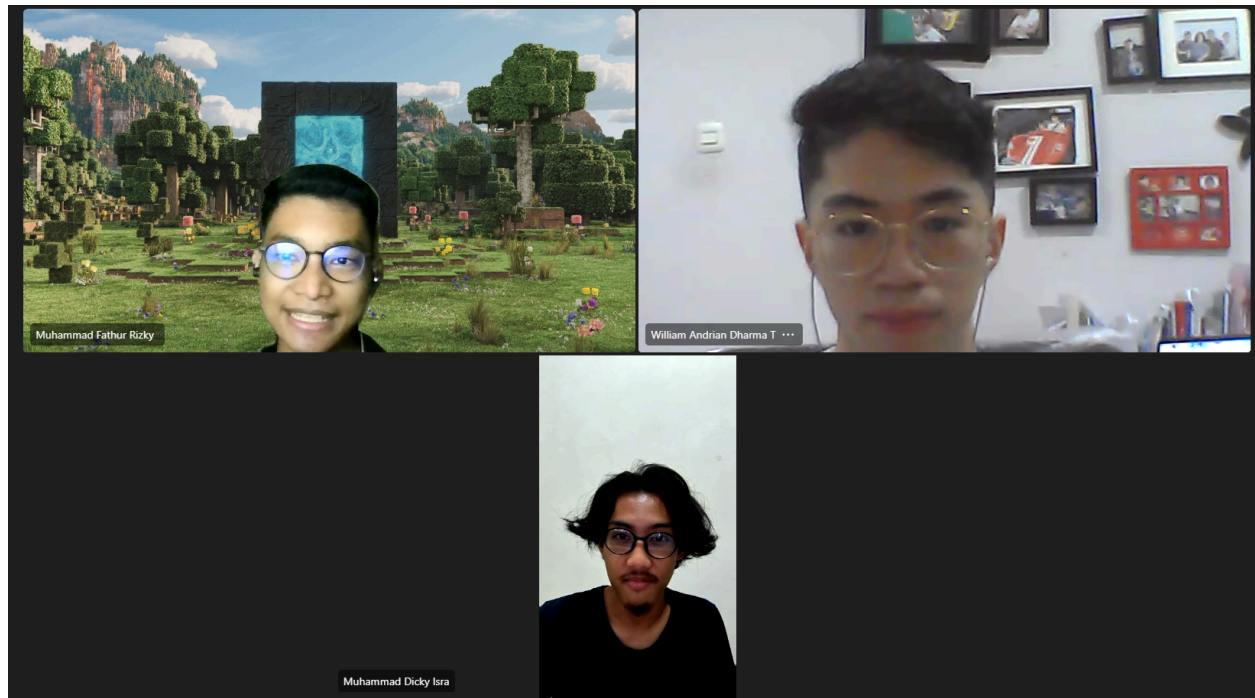


**Laporan Tugas Besar I Strategi Algoritma**  
**Robocode Tank Royale**  
**PolicyGradientStrategist**



William Andrian Dharma T  
13523006

[13523006@std.stei.itb.ac.id](mailto:13523006@std.stei.itb.ac.id)

Muhammad Dicky Isra  
13523075

[13523075@std.stei.itb.ac.id](mailto:13523075@std.stei.itb.ac.id)

Muhammad Fathur Rizky  
13523105

[13523105@std.stei.itb.ac.id](mailto:13523105@std.stei.itb.ac.id)

**Program Studi Teknik Informatika**  
**Sekolah Teknik Elektro dan Informatika**  
**Institut Teknologi Bandung**  
**2025**

## **BAB I: Deskripsi Tugas**

Robocode adalah permainan pemrograman di mana pemain merancang kode untuk bot berbentuk tank virtual yang bertarung satu sama lain di arena. Pertempuran dalam Robocode berlangsung hingga hanya tersisa satu bot sebagai pemenang, mirip dengan konsep Battle Royale, sehingga permainan ini juga disebut Tank Royale. Nama Robocode sendiri merupakan singkatan dari Robot Code, yang merujuk pada versi asli permainan ini.

Versi terbaru, Robocode Tank Royale, menghadirkan evolusi permainan dengan memungkinkan bot untuk bertarung melalui jaringan atau Internet. Dalam permainan ini, pemain berperan sebagai programmer yang mengembangkan kecerdasan buatan bagi bot mereka, tanpa kendali langsung selama pertempuran. Pemain bertugas menulis kode yang mengatur strategi dan logika bot, termasuk cara bergerak, mendeteksi lawan, menembak, serta merespons berbagai situasi di medan perang.

Pada tugas besar pertama ini, mahasiswa diminta untuk membuat sebuah bot yang akan dipertandingkan satu sama lain. Strategi utama yang harus digunakan dalam pengembangan bot ini adalah strategi greedy.

### **Komponen-Komponen Permainan**

#### **1. Rounds dan Turns**

Pertempuran dalam Robocode dapat terdiri dari beberapa rounds. Secara default, satu pertempuran berisi 10 rounds, di mana setiap round memiliki pemenang dan yang kalah. Setiap round terbagi dalam beberapa turns, yang merupakan unit waktu terkecil dalam permainan. Satu turn adalah satu ketukan waktu dan satu putaran permainan. Jumlah turns dalam satu round bergantung pada durasi hingga tersisa satu bot terakhir yang bertahan.

Pada setiap turn, sebuah bot dapat:

- Menggerakkan bot, memindai musuh, dan menembakkan senjata.
- Bereaksi terhadap peristiwa seperti tertembak, bertabrakan dengan bot lain, atau menabrak dinding.
- Mengirimkan perintah untuk bergerak, berputar, memindai, dan menembak ke server.

API (Application Programming Interface) bot resmi akan secara otomatis mengirimkan niat bot ke server, sehingga pemain tidak perlu mengaturnya sendiri, kecuali jika menggunakan API bot yang dibuat sendiri.

Pada setiap turn, bot akan menerima informasi terbaru tentang:

- Posisi dan orientasinya di medan perang.
- Keberadaan bot musuh jika terdeteksi oleh pemindai.

Catatan:

Game engine yang digunakan dalam tugas besar ini tidak sepenuhnya mengikuti aturan default terkait Rounds & Turns.

## 2. Batas Waktu Giliran (Turn Timeout)

Setiap bot memiliki batas waktu tertentu untuk setiap turn, biasanya 30-50 ms (bergantung pada aturan pertempuran). Jika bot tidak mengirimkan perintah dalam batas waktu yang ditentukan, maka perintahnya tidak akan dikirim ke server, menyebabkan bot melewatkan turn tersebut. Akibatnya, bot tidak bisa bergerak atau menembak hingga turn berikutnya.

## 3. Energi

- Semua bot memulai permainan dengan 100 poin energi.
- Bot akan kehilangan energi jika tertembak atau bertabrakan dengan bot musuh.
- Bot juga kehilangan energi saat menembakkan meriamnya.
- Jika peluru bot mengenai musuh, bot akan mendapatkan energi sebanyak 3 kali lipat dari energi yang digunakan untuk menembak.
- Bot dengan energi nol akan dinonaktifkan dan tidak bisa bergerak. Jika terkena serangan dalam kondisi ini, bot akan hancur.

## 4. Peluru

- Semakin besar energi yang digunakan untuk menembak, semakin berat dan lambat pelurunya.
- Peluru yang lebih berat menghasilkan lebih banyak kerusakan dan memberikan lebih banyak energi jika mengenai musuh.
- Peluru yang lebih ringan bergerak lebih cepat, lebih mudah mengenai target, tetapi menghasilkan lebih sedikit poin energi.

## 5. Panas Meriam (Gun Heat)

- Saat menembak, meriam akan memanaskan. Peluru yang lebih berat menyebabkan panas lebih tinggi.
- Jika meriam terlalu panas, bot tidak dapat menembak hingga suhu meriam turun ke nol.
- Meriam sudah dalam kondisi panas di awal ronde dan memerlukan waktu pendinginan sebelum dapat digunakan pertama kali.

## 6. Tabrakan

- Bot akan menerima kerusakan jika menabrak dinding (wall damage).

- Jika bot bertabrakan dengan bot lain, keduanya akan menerima kerusakan.
- Ramming (menabrak lawan dengan sengaja) memberikan sedikit tambahan skor bagi bot yang menyerang.

## 7. Bagian Tubuh Tank

Bot terdiri dari tiga bagian utama:

- Body: Bagian utama tank yang digunakan untuk bergerak.
- Gun: Meriam yang digunakan untuk menembak dan bisa berputar secara independen dari tubuh.
- Radar: Alat untuk mendeteksi musuh, yang juga bisa berputar secara independen dari tubuh dan meriam.

## 8. Pergerakan & Berbelok

- Bot dapat bergerak maju dan mundur dengan kecepatan maksimum tertentu.
- Percepatan maksimum: 1 unit per giliran.
- Perlambatan maksimum: 2 unit per giliran.
- Kecepatan belok tubuh bot bergantung pada kecepatan gerakannya. Semakin cepat bot bergerak, semakin lambat belokannya.

Kecepatan Putar Maksimum per Giliran:

- Radar: 45 derajat (dapat berputar 360 derajat dalam 8 giliran).
- Turret & Meriam: 20 derajat.
- Tubuh Tank: 10 derajat (dipengaruhi oleh kecepatan bot).

Catatan:

Bot tidak mengonsumsi energi saat bergerak atau berbelok.

## 9. Pemindaian (Radar Scanning)

- Radar dapat mendeteksi musuh dalam jangkauan hingga 1200 piksel.
- Pemindaian hanya terjadi dalam sudut pemindaian (scan arc), yang merupakan area dari arah radar sebelumnya ke arah radar saat ini dalam satu giliran.
- Jika radar tidak bergerak, sudut pemindaian akan nol derajat, dan bot tidak akan dapat mendeteksi musuh. Oleh karena itu, radar harus selalu bergerak untuk memindai musuh.

## 10. Skor & Ranking

Pada akhir pertempuran, setiap bot akan diberi peringkat berdasarkan total skor yang diperoleh. Tujuan utama tugas besar ini adalah membuat bot dengan skor setinggi mungkin.

Komponen Skor:

- Bullet Damage: Skor sebesar damage yang diberikan kepada musuh dengan peluru.
- Bullet Damage Bonus: Jika peluru membunuh musuh, bot mendapatkan 20% dari damage yang diberikan ke musuh yang terbunuh.
- Survival Score: Setiap kali bot lain mati, bot yang bertahan mendapatkan 50 poin.
- Last Survival Bonus: Bot terakhir yang bertahan mendapatkan 10 poin dikali jumlah musuh.
- Ram Damage: Skor sebesar 2 kali damage yang diberikan dengan menabrak musuh.
- Ram Damage Bonus: Jika musuh terbunuh karena ditabrak, bot mendapatkan 30% dari damage yang diberikan.

Skor akhir adalah akumulasi dari keenam komponen di atas.

Catatan:

Game juga akan menampilkan berapa kali bot meraih peringkat 1, 2, atau 3 di setiap ronde, tetapi ini tidak berpengaruh pada ranking akhir. Bot dengan total skor tertinggi dianggap sebagai pemenang pertempuran.

## BAB II: Landasan Teori

Algoritma greedy didefinisikan sebagai metode yang membangun solusi secara bertahap dengan selalu memilih pilihan yang memberikan manfaat paling jelas dan segera pada setiap tahap. Pendekatan ini berfokus pada optimasi lokal dengan harapan bahwa pilihan tersebut akan mengarah pada solusi optimal secara global. Menurut [Wikipedia: Greedy Algorithm](#), algoritma ini mengikuti heuristik pemecahan masalah dengan membuat pilihan optimal lokal pada setiap tahap, meskipun tidak selalu menghasilkan solusi optimal global.

Dalam pengembangan bot untuk game Robocode Tank Royale, API yang disediakan sangat memudahkan para pengembang dalam mengintegrasikan logika dan strategi ke dalam bot mereka. API ini memberikan serangkaian metode dan antarmuka (*interface*) yang memungkinkan bot untuk melakukan interaksi secara langsung dengan lingkungan permainan. Misalnya, melalui API tersebut, bot dapat mengakses data sensor untuk mendeteksi posisi musuh, mengukur jarak, serta menentukan kondisi medan tempur secara real time.

Selain itu, dokumentasi resmi API—seperti yang dapat diakses melalui [tautan berikut](#)—memberikan panduan lengkap mengenai fungsi-fungsi yang tersedia, cara pemanggilannya, serta contoh kode yang relevan untuk mengoperasikan bot. Dengan demikian, para pengembang tidak hanya dapat memahami bagaimana cara kerja API, tetapi juga mendapatkan inspirasi untuk mengembangkan strategi pertahanan atau serangan yang lebih canggih. API tersebut mendukung fleksibilitas dalam pengembangan, sehingga memungkinkan penyesuaian dan peningkatan fitur sesuai dengan kebutuhan permainan dan preferensi pengguna.

## BAB III: Aplikasi Strategi Greedy

### 3.1 Fungsi Objektif Umum

Poin pada Tank Royale secara garis besar dihitung dengan rumus sebagai berikut.

$$f = survival + surv.bonus + bullet dmg + bullet.bonus + ramdmg + ram.bonus$$

Terlihat bahwa terdapat enam komponen yang bisa dimaksimalkan atau dengan kata lain  $f$  adalah fungsi objektif yang memaksimalkan jumlah poin dari keenam komponen tersebut.

### 3.2 Eksplorasi Solusi Greedy Alternatif

#### 3.2.1 Ellipsis: Maximizing the Survival Rate + Bullet Damage



Perhatikan bahwa untuk memaksimalkan Survival Rate, tingkat Ramming harus diperkecil atau tidak sama sekali. Intuisi ini terbangun karena berdasarkan pengalaman empiris, bot saat ramming akan mudah dikenakan peluru sehingga terlalu berisiko. Karena mudah dikenakan peluru, maka terciptalah bot yang memaksimalkan poin survival dan kerusakan peluru yang dikenakan kepada musuh. Bot ini memiliki karakteristik untuk menghindari pertempuran jarak dekat dengan gerakan mengorbit atau lebih preventif terhadap risiko tak terduga.

#### a. Mapping Elemen

- 1) Himpunan Kandidat: Semua aksi yang tersedia.
- 2) Himpunan Solusi: Aksi yang terpilih.
- 3) Fungsi Solusi: Memeriksa apakah aksi yang dipilih membuat bot memaksimalkan survival rate dan bullet damage.
- 4) Fungsi Seleksi: Memilih aksi yang tidak terlalu nafsu dalam melakukan pertempuran.

- 5) Fungsi Kelayakan: Memeriksa apakah aksi yang dipilih memenuhi persyaratan dasar untuk bertahan hidup sambil memaksimalkan bullet damage.
- 6) Fungsi Objektif: Memaksimalkan poin survival, surv.bonus, bullet damage, dan bullet bonus.

#### **b. Analisis Efisiensi Solusi**

Bot Ellipsis melakukan perhitungan secara real-time pada setiap turn dengan operasi dasar seperti perbandingan, aritmatika, dan fungsi trigonometri yang semuanya memiliki kompleksitas waktu konstan, yakni  $O(1)$ . Pada loop utama, bot mengevaluasi kondisi pergerakan menggunakan perhitungan sederhana berdasarkan turn counter, sedangkan saat terjadi event seperti mendeteksi bot lain melalui event OnScannedBot, bot memperbarui target atau melakukan prediksi posisi dengan operasi matematika yang tetap berlangsung dalam waktu konstan. Begitu pula pada event lain seperti terkena peluru, menabrak dinding, atau bertabrakan dengan bot lain, hanya dilakukan penyesuaian terhadap kecepatan atau sudut dengan perhitungan sederhana. Dengan demikian, meskipun perhitungan dilakukan setiap siklus turn, beban komputasi tetap ringan dan memastikan respons cepat dalam pertandingan dinamis karena setiap turn hanya memerlukan waktu komputasi  $O(1)$ .

#### **c. Analisis Efektivitas Solusi**

Maximizing the survival rate dan bullet damage akan sangat efektif jika terdapat minimal 2 bot yang memiliki ide lock 1 target dan melakukan ramming. Hal ini akan sangat menguntungkan mengingat bullet akan sampai kepada target lebih pasti. Kepastian ini disebabkan bot yang melakukan ramming tidak terlalu lincah atau dalam kata lain diam sehingga kepastian lebih terjamin.

Bot ini tidak terlalu efektif jika musuh sangat lincah sehingga akan menyulitkan dan menghabiskan energi bot itu sendiri akibat dari peluru yang terbuang sia-sia (tidak mengenai target). Selain itu, terdapat kasus spesial di mana bot ini dikunci oleh musuh yang tidak terdeteksi oleh bot sehingga tidak bisa bergerak dan akan mati tanpa perlawanan.

### **3.2.2 Hebi: Maximizing Ramming Damage and Bullet Damage**

Bot Hebi (へび・蛇), sesuai namanya yang berarti “Ular” dalam Bahasa Jepang, berfungsi dengan cara mendekat ke musuh hingga jika bisa menempel agar bisa memaksimalkan poin *ramming* dan *bullet*. Strategi ini memanfaatkan sistem penilaian dimana damage *ram* bernilai 2 kali lipat, dan dengan posisi musuh yang dekat, peluru dengan *firepower* 3 menjadi lebih efektif yang mengakibatkan poin *bullet* juga naik.

#### **a. Mapping Elemen**

- 1) Himpunan Kandidat : Semua aksi yang tersedia.
- 2) Himpunan Solusi : Aksi yang dipilih.
- 3) Fungsi Solusi: Memeriksa apakah aksi yang dilakukan menghasilkan jarak terkecil dengan musuh dan memaksimalkan bullet damage
- 4) Fungsi Seleksi: Memilih aksi yang membuat bot bisa sedekat mungkin dengan musuh
- 5) Fungsi Kelayakan: Memeriksa apakah aksi yang dilakukan menghasilkan jarak terkecil dengan musuh.
- 6) Fungsi Objektif: Memaksimalkan ram damage dan bullet damage



## b. Analisis Efisiensi Solusi

Algoritma yang terdapat di *bot* ini dapat dibagi menjadi 2 yaitu *Movement* dan *Shooting*. Algoritma *Movement* mengandung logika pengejaran *bot* dan memiliki kompleksitas konstan  $O(1)$  karena hanya menggunakan operasi-operasi matematika dasar. Namun, algoritma *Shooting* memiliki kompleksitas  $O(n)$  dimana  $n$  bernilai jarak ke bot musuh dibagi dengan kecepatan peluru yang akan ditembak, untuk melakukan prediksi posisi musuh berdasarkan posisi, arah, dan kecepatannya, yang sesuai kecepatan peluru yang akan ditembak. Secara memori, bot ini efisien karena hanya menyimpan beberapa variabel primitif dan satu *array* posisi musuh yang diprediksi.

## c. Analisis Efektivitas Solusi

Bot Hebi yang memiliki strategi *ramming* paling efektif ketika musuh-musuhnya tidak memiliki algoritma menghindar yang cukup lincah atau acak, karena dapat mendekat dengan sangat cepat dan mendapat poin yang banyak. Bot Hebi juga efektif jika dapat menjadi pihak ketiga dari sebuah pertempuran diantara bot karena dapat memanfaatkan kekacauannya untuk mendekat ke bot manapun yang sedang tidak memperhatikannya.

Namun, bot Hebi kurang efektif melawan musuh yang memiliki algoritma menghindar cepat terutama dalam kondisi 1v1 karena *bot* ini tidak bisa belajar dari aksi yang dilakukan dirinya atau musuhnya, sehingga jika kondisi menguntungkan bot musuh, kondisi tersebut kemungkinan besar tidak bisa ditanggapi karena tidak adanya perubahan *behaviour*, kecuali algoritma bot musuh membuat kondisi berganti. Lalu, karena merupakan *bot ramming*, bot Hebi kemungkinan besar tidak akan selamat hingga akhir round untuk mendapatkan poin *survival* yang banyak.

### 3.2.3 LowestEnergyChaser: Maximizing Killing Score

Perhatikan bahwa dengan sistem skor yang tersedia akan diberikan bonus skor jika bot membunuh musuh. Bonus tersebut diberikan sebagai berikut, yaitu jika bot membunuh dengan peluru maka akan diberikan tambahan skor sebesar 20% dari total *damage* yang diberikan. Selain itu akan diberikan tambahan skor sebesar 30% dari total damage yang diberikan jika berhasil membunuh dengan cara *ramming*. Sehingga pendekatan ini adalah pendekatan dengan usaha untuk mencari bot dengan *energy* terkecil untuk memaksimalkan bonus membunuh musuh.

#### a. Mapping Elemen

- 1) Himpunan Kandidat: Semua aksi yang tersedia.
- 2) Himpunan Solusi: Aksi yang terpilih.
- 3) Fungsi Solusi: Memeriksa apakah aksi yang dipilih membuat bot mengincar bot dengan Energy terendah.
- 4) Fungsi Seleksi: Memilih aksi yang memaksimalkan skor yang bisa didapat.
- 5) Fungsi Kelayakan: Memeriksa apakah aksi yang dipilih memenuhi persyaratan dasar untuk bertahan hidup sambil memaksimalkan skor.
- 6) Fungsi Objektif: Memaksimalkan bullet damage, bullet bonus, dan ram bonus.

### **b. Analisis Efisiensi Solusi**

Bot ini memiliki kompleksitas waktu dan ruang yang cukup efisien. Search musuh dilakukan dalam waktu komputasi dengan kompleksitas  $O(n)$ , di mana  $n$  adalah jumlah bot yang ditemukan dalam satu putaran. Karena hanya melibatkan operasi matematika sederhana, proses prediksi posisi musuh dan pengambilan keputusan untuk pergerakan dan serangan memiliki kompleksitas  $O(1)$ . Dalam hal penggunaan memori, bot hanya menyimpan informasi tentang target yang terkunci, seperti posisi, energi, kecepatan, dan arah. Seberapa sering bot ini mendeteksi dan merespons musuh menentukan sumber dayanya, tetapi karena perhitungan yang cukup ringan di setiap putaran, penggunaannya tetap optimal.

### **c. Analisis Efektivitas Solusi**

Metode ini kurang efektif untuk konteks bot umum. Bot ini lemah jika diserang oleh bot rammer karena tidak memperhatikannya. Selain itu karena bot ini terlalu *greedy* dengan hanya memperhatikan 1 bot dalam satu waktu maka bot ini rawan diserang oleh berbagai arah. Sehingga proses meraih skor untuk mengincar bonus tidak optimal.

## **3.2.4 Kawakaze: Minimizing Risk and Maximizing Bullet Score**

Bot Kawakaze menerapkan strategi *Minimum Risk Movement* untuk mengurangi probabilitas terjebak dalam sebuah posisi yang tidak menguntungkan. Perhatikan bahwa terdapat bonus *survival* sehingga meminimalkan risiko dalam pergerakan diharapkan bisa menaikkan *survival rate* dari bot, sambil mengumpulkan poin lewat menembak bot lain.

### **a. Mapping Elemen**

- 1) Himpunan Kandidat: Semua aksi yang tersedia.
- 2) Himpunan Solusi: Aksi yang terpilih.
- 3) Fungsi Solusi: Memeriksa apakah aksi yang dipilih membuat bot berada di titik dengan risiko terendah dalam sebuah area
- 4) Fungsi Seleksi: Memilih aksi yang meminimalkan risiko dalam posisi tersebut
- 5) Fungsi Kelayakan: Memeriksa apakah aksi yang dipilih membuat bot pergi ke titik dengan risiko paling rendah dalam sebuah area
- 6) Fungsi Objektif: Meminimalkan risiko dan memaksimalkan *bullet damage*

### **b. Analisis Efisiensi Solusi**

Bot ini memiliki dua komponen penting, algoritma *Minimal Risk Movement* dan *Shooting*. Algoritma *Minimal Risk Movement* memiliki parameter konstan 200 titik yang dievaluasi tingkat risikonya, ditambah dengan jumlah bot musuh yang hidup. Sehingga kompleksitas waktunya menjadi  $O(n)$  dengan  $n$  jumlah bot musuh yang hidup. Namun, algoritma *Shooting* memiliki kompleksitas  $O(n)$  dimana  $n$  bernilai jarak ke bot musuh dibagi dengan kecepatan peluru yang akan ditembak, untuk melakukan prediksi posisi musuh berdasarkan posisi, arah, dan kecepatannya, yang sesuai kecepatan peluru yang akan ditembak. Secara memori, bot ini memiliki komponen yang dapat dibayangkan cukup boros yaitu *Hashtable* dari musuh-musuh yang ada, yang menyimpan sebuah *class enemyBot* untuk menyimpan keadaan untuk perhitungan.

### **c. Analisis Efektivitas Solusi**

Bot ini efektif jika musuh memiliki algoritma *shooting* yang kurang tanggap karena pergerakan bot ini tidak mengikuti sebuah pola fix. Namun bot ini akhirnya dinilai kurang efektif karena kurang cepat dalam melakukan perpindahan dan arah pergerakannya yang masih terlalu lurus. Sehingga walaupun tidak mengikuti sebuah pola, bot musuh yang cukup tanggap dapat menembak dan mengenai bot ini. Lalu, bot ini juga lemah terhadap bot *rammer* karena tidak bisa dengan tanggap menghindari bot jenis tersebut.

### **3.3 Strategi Greedy yang Dipilih**

Berdasarkan solusi-solusi yang telah dipaparkan pada subbab 3.2, terdapat kelebihan dan kekurangan dari masing-masing solusi karena pada setiap round, game memiliki kondisi lawan yang tidak menentu. Maka dari itu, kami memilih algoritma yang dapat lebih general (tidak terlalu *overfitting*) dalam menghadapi berbagai permasalahan. Kami memutuskan untuk menggunakan bot Ellipsis sebagai main bot yang akan dilakukan kompetisi karena

## BAB IV: Implementasi dan Pengujian

### A. Implementasi 3 Solusi Alternatif

#### Hebi

```
BEGIN
    // Inisialisasi
    bot ← new HebiBot
    Load bot configuration from "Hebi.json"
    Set BodyColor, TurretColor, RadarColor, BulletColor, ScanColor,
TracksColor, GunColor
    Set GunTurnRate ← MaxGunTurnRate
    Set RadarTurnRate ← MaxRadarTurnRate
    enemy ← [ArenaWidth/2, ArenaHeight/2]    // Titik tengah arena sebagai
target awal
    AdjustGunForBodyTurn ← TRUE
    AdjustRadarForGunTurn ← TRUE
    AdjustRadarForBodyTurn ← TRUE

    // Persiapan awal radar dan meriam
    turnDirection ← (if RadarBearingTo(enemy) > 0 then 1 else -1)
    SetTurnRadarLeft(360 * turnDirection)
    SetTurnGunLeft(GunBearingTo(enemy))

    // Loop utama selama pertandingan
    WHILE IsRunning DO
        turnsSinceShot ← turnsSinceShot + 1
        angle ← BearingTo(enemy)

        // Tetapkan arah pergerakan
        turnDirection ← 1
        IF turnDirection ≠ lastDirection THEN
            changeDirection ← TRUE
        ENDIF

        IF changeDirection AND (ABS(Speed) ≥ 2) THEN
            MaxSpeed ← 0.0001    // Kurangi kecepatan untuk perubahan arah
cepat
        ELSE
            MaxSpeed ← 9        // Atur kecepatan normal
            changeDirection ← FALSE
        ENDIF
        lastDirection ← turnDirection

        SetForward(999 * turnDirection)

        // Atur pergerakan radar atau belokan bot tergantung waktu sejak
tembakan terakhir
        IF turnsSinceShot > 10 THEN
            SetTurnRadarLeft(360 * turnDirection)
        ELSE
            SetTurnLeft(angle)
        ENDIF

        SetRescan()    // Lakukan pemindaian ulang
```

```

        Go()          // Eksekusi perintah turn ini
    ENDWHILE

    // Event handler ketika bot musuh terdeteksi
    ON_SCANNED_BOT(event):
        turnsSinceShot ← 0
        angleToEnemy ← BearingTo(event.X, event.Y)
        SetTurnRadarLeft(RadarBearingTo(event.X, event.Y))

        // Inisialisasi prediksi posisi musuh
        preds ← [event.X, event.Y]
        deltaTime ← 0

        // Tentukan daya tembak berdasarkan jarak
        IF DistanceTo(event.X, event.Y) < 100 THEN
            currFirepower ← 3
        ELSE IF DistanceTo(event.X, event.Y) < 200 THEN
            currFirepower ← 2
        ELSE
            currFirepower ← 1.1
        ENDIF

        bulletSpeed ← CalcBulletSpeed(currFirepower)

        // Prediksi posisi musuh menggunakan pergerakan linear
        WHILE ( (deltaTime + 1) * bulletSpeed < DistanceTo(preds[0], preds[1])
) DO
            deltaTime ← deltaTime + 1
            preds[0] ← preds[0] + SIN(event.Direction) * event.Speed
            preds[1] ← preds[1] + COS(event.Direction) * event.Speed

            // Periksa apakah prediksi keluar dari batas arena
            IF (preds[0] < 18 OR preds[1] < 18 OR preds[0] > ArenaWidth - 18
OR preds[1] > ArenaHeight - 18) THEN
                preds[0] ← MIN( MAX(18, preds[0]), ArenaWidth - 18 )
                preds[1] ← MIN( MAX(18, preds[1]), ArenaHeight - 18 )
                BREAK
            ENDIF
        ENDWHILE

        SetTurnGunLeft( GunBearingTo(preds[0], preds[1]) )
        SetFire(currFirepower)
        enemy ← COPY(preds)    // Update posisi musuh sebagai target
    END_ON_SCANNED_BOT

    // Fungsi pembantu untuk menentukan arah samping
    FUNCTION SideAngle(angle):
        RETURN (if angle > 0 then 1 else -1)
    END_FUNCTION
END

```

## Kawakaze

```
BEGIN
// Inisialisasi Bot Kawakaze
bot ← new KawakazeBot
bot.Start()

// CLASS enemyBot untuk menyimpan data musuh
CLASS enemyBot:
    position ← [0, 0]
    direction ← 0
    speed ← 0
    isAlive ← FALSE
    energy ← 0
END_CLASS

// Deklarasi variabel global
random ← new Random()
enemies ← new Hashtable() // Menyimpan objek enemyBot dengan key
sebagai ID musuh
target ← new enemyBot() // Target musuh utama
nextSpot ← [0, 0]
lastSpot ← [0, 0]

// Inisialisasi warna dan properti tampilan bot
BodyColor ← Color(0xe0, 0xde, 0xe1) // White
TurretColor ← Color(0x29, 0xb9, 0xdb) // Light Blue
RadarColor ← Color(0x1d, 0x1b, 0x43) // Dark Blue
BulletColor ← Color(0x29, 0x27, 0x26) // Black
ScanColor ← Color(0x8c, 0x4f, 0x61) // Blue
TracksColor ← Color(0x6e, 0x6a, 0x68) // Dark
GunColor ← Color(0x6e, 0x6a, 0x68)

// Inisialisasi turn rate dan pengaturan decoupling
GunTurnRate ← MaxGunTurnRate
RadarTurnRate ← MaxRadarTurnRate
AdjustGunForBodyTurn ← TRUE
AdjustRadarForGunTurn ← TRUE
AdjustRadarForBodyTurn ← TRUE
SetTurnRadarRight(very large number) // Putar radar terus menerus
nextSpot ← [X, Y] // Titik tujuan awal = posisi
saat ini
lastSpot ← [X, Y]
target ← new enemyBot()

// Loop utama selama pertandingan
WHILE IsRunning DO
    IF target.isAlive AND (TurnNumber > 10) THEN
        CALL Operations()
    ENDIF
    Go()
ENDWHILE

// FUNCTION: Operations - mengatur pergerakan dan operasi senjata
FUNCTION Operations():
```

```

// Operasi penembakan
dis ← DistanceTo(target.position[0], target.position[1])
IF Energy > 1 THEN
    currFire ← MIN( MIN(Energy / 6, 300 / dis), target.energy / 3 )
    CALL Shooting(currFire)
ENDIF

nextDis ← DistanceTo(nextSpot[0], nextSpot[1])
IF nextDis < 16 THEN
    modifier ← 1 - random.Next( (int)( random.NextDouble()^EnemyCount
) )

    FOR i FROM 0 TO 199 DO
        testAngle ← random.NextDouble() * 2 * PI
        testDis ← random.NextDouble() * 200
        testSpot[0] ← X + MIN(dis * 0.8, 100 + testDis) *
SIN(testAngle)
        testSpot[1] ← Y + MIN(dis * 0.8, 100 + testDis) *
COS(testAngle)
        IF (testSpot[0] > 30 AND testSpot[0] < ArenaWidth - 30 AND
            testSpot[1] > 30 AND testSpot[1] < ArenaHeight - 30) THEN
            IF RiskScoring(testSpot, modifier) < RiskScoring(nextSpot,
modifier) THEN
                nextSpot ← testSpot
            ENDIF
        ENDIF
    ENDFOR
    lastSpot[0] ← X
ELSE
    angle ← BearingTo(nextSpot[0], nextSpot[1])
    frontBack ← SideAngle(angle) // 1 jika depan, -1 jika belakang
    IF frontBack = 1 THEN
        SetTurnLeft(angle)
    ELSE
        SetTurnRight( NormalizeRelativeAngle(angle + 180) )
    ENDIF
    SetForward(999 * frontBack)
    MaxSpeed ← (ABS(angle) > 1) ? 0.01 : 8
ENDIF
END_FUNCTION

// FUNCTION: Shooting - menembak target dengan prediksi posisi
FUNCTION Shooting(currFirepower):
    deltaTime ← 0
    bulletSpeed ← CalcBulletSpeed(currFirepower)
    preds ← [target.position[0], target.position[1]]
    WHILE ((deltaTime + 1) * bulletSpeed < DistanceTo(preds[0], preds[1]))
DO
        deltaTime ← deltaTime + 1
        preds[0] ← preds[0] + SIN(target.direction) * target.speed
        preds[1] ← preds[1] + COS(target.direction) * target.speed
        IF preds[0] < 18 OR preds[1] < 18 OR preds[0] > ArenaWidth - 18 OR
preds[1] > ArenaHeight - 18 THEN
            preds[0] ← CLAMP(preds[0], 18, ArenaWidth - 18)
            preds[1] ← CLAMP(preds[1], 18, ArenaHeight - 18)
            BREAK
        ENDIF
    ENDIF

```

```

        ENDWHILE
        SetTurnGunLeft( GunBearingTo(preds[0], preds[1]) )
        SetFire(currFirepower)
    END_FUNCTION

    // FUNCTION: RiskScoring - menghitung skor risiko untuk sebuah titik
    FUNCTION RiskScoring(p, modifier):
        risk ← modifier * 0.08 / ((CalcDistance(p[0], p[1], lastSpot[0],
lastSpot[1]))^2)
        FOR EACH enemy IN enemies.Values DO
            IF enemy.isAlive THEN
                risk ← risk + MIN(enemy.energy / Energy, 2) *
                    (1 + ABS( COS( CalcBearingOf(p[0], p[1], X, Y) -
CalcBearingOf(p[0], p[1], enemy.position[0], enemy.position[1]) ) ))
            ENDIF
        ENDFOR
        RETURN risk
    END_FUNCTION

    // EVENT: OnScannedBot - menangani deteksi bot musuh
    EVENT OnScannedBot(e):
        enemy ← enemies[e.ScannedBotId]
        IF enemy IS NULL THEN
            enemy ← new enemyBot()
            enemies.Add(e.ScannedBotId, enemy)
        ENDIF
        enemy.energy ← e.Energy
        enemy.direction ← e.Direction
        enemy.speed ← e.Speed
        enemy.isAlive ← TRUE
        enemy.position ← [e.X, e.Y]
        IF (NOT target.isAlive) OR (DistanceTo(e.X, e.Y) <
DistanceTo(target.position[0], target.position[1])) THEN
            target ← enemy
        ENDIF
        IF EnemyCount = 1 THEN
            SetTurnRadarLeft(RadarTurnRemaining)
        ENDIF
    END_EVENT

    // EVENT: OnBotDeath - menangani kematian bot musuh
    EVENT OnBotDeath(e):
        enemy ← enemies[e.VictimId]
        enemy.isAlive ← FALSE
    END_EVENT

    // FUNCTION: CalcBearingOf - menghitung bearing relatif antara dua titik
    FUNCTION CalcBearingOf(x1, y1, x2, y2):
        RETURN NormalizeRelativeAngle( NormalizeAbsoluteAngle(180 * ATAN2(x2 -
x1, y2 - y1) / PI) )
    END_FUNCTION

    // FUNCTION: CalcDistance - menghitung jarak antara dua titik
    FUNCTION CalcDistance(x1, y1, x2, y2):
        RETURN SQRT((x1 - x2)^2 + (y1 - y2)^2)
    END_FUNCTION

```



```

    // FUNCTION: SideAngle - menentukan apakah sudut menghadap depan (1) atau
    belakang (-1)
    FUNCTION SideAngle(angle):
        RETURN (angle < -90 OR angle > 90) ? -1 : 1
    END_FUNCTION

    // FUNCTION: CLAMP - membatasi nilai dalam rentang min dan max
    FUNCTION CLAMP(value, min, max):
        RETURN MAX(min, MIN(value, max))
    END_FUNCTION
END

```

## LowestEnergyChaser

```

BEGIN
    // Inisialisasi Bot lowestEnergyChaser
    bot <- new lowestEnergyChaserBot
    bot.Start()

    // Deklarasi variabel global
    locked <- FALSE
    lockedTargetId <- -1
    lockedTargetX <- 0
    lockedTargetY <- 0
    lockedTargetEnergy <- ∞
    lockedTargetDistance <- ∞
    lockedTargetVelocity <- 0
    lockedTargetHeading <- 0
    turnCounter <- 0
    lastSeenTurn <- 0

    // Konstanta
    LockTimeout <- 10
    CloseRangeDistance <- 150.0
    EnemyRammingThreshold <- 20.0
    BulletSpeedFactor <- 20.0

    // Loop utama selama pertandingan
    WHILE IsRunning DO
        turnCounter <- turnCounter + 1

        IF locked AND (turnCounter - lastSeenTurn > LockTimeout) THEN
            locked <- FALSE
            lockedTargetId <- -1
        ENDIF

        GunTurnRate <- MaxGunTurnRate
        RadarTurnRate <- MaxRadarTurnRate

        IF locked THEN

```

```

    bearingToTarget <- BearingTo(lockedTargetX, lockedTargetY)
    TurnRate <- Clamp(bearingToTarget, -MaxTurnRate, MaxTurnRate)

    IF lockedTargetEnergy < EnemyRammingThreshold THEN
        SetForward(1000)
    ELSE
        distance <- lockedTargetDistance
        moveDistance <- (distance > CloseRangeDistance) ? (distance -
CloseRangeDistance) : -10
        SetForward(moveDistance)
    ENDIF

    // Prediksi posisi musuh
    CALL PredictEnemyPosition(predictedX, predictedY)

    gunBearing <- NormalizeRelativeAngle(GunBearingTo(predictedX,
predictedY))
    GunTurnRate <- Clamp(gunBearing, -MaxGunTurnRate, MaxGunTurnRate)

    radarBearing <-
NormalizeRelativeAngle(RadarBearingTo(lockedTargetX, lockedTargetY))
    RadarTurnRate <- Clamp(radarBearing, -MaxRadarTurnRate,
MaxRadarTurnRate)

    firePower <- (lockedTargetEnergy < EnemyRammingThreshold) ? 3 : 1
    SetFire(firePower)
ELSE
    TurnRate <- MaxTurnRate
ENDIF

Go()
ENDWHILE

// EVENT: OnScannedBot - saat mendeteksi bot musuh
EVENT OnScannedBot(e):
    scannedDistance <- DistanceTo(e.X, e.Y)
    enemyEnergy <- e.Energy

    IF locked AND (e.ScannedBotId = lockedTargetId) THEN
        lockedTargetX <- e.X
        lockedTargetY <- e.Y
        lockedTargetDistance <- scannedDistance
        lockedTargetEnergy <- enemyEnergy
        lockedTargetVelocity <- e.Speed
        lockedTargetHeading <- e.Direction
        lastSeenTurn <- turnCounter
    ELSE IF (NOT locked) OR (enemyEnergy < lockedTargetEnergy) THEN
        locked <- TRUE
        lockedTargetId <- e.ScannedBotId
        lockedTargetX <- e.X
        lockedTargetY <- e.Y
        lockedTargetDistance <- scannedDistance
        lockedTargetEnergy <- enemyEnergy
        lockedTargetVelocity <- e.Speed
        lockedTargetHeading <- e.Direction
        lastSeenTurn <- turnCounter
    
```

```

        ENDIF
    END_EVENT

    // EVENT: OnHitByBullet - saat terkena peluru
    EVENT OnHitByBullet(e):
        TurnRate <- 5
    END_EVENT

    // EVENT: OnHitWall - saat menabrak dinding
    EVENT OnHitWall(e):
        SetForward(-100)
    END_EVENT

    // EVENT: OnHitBot - saat terjadi tabrakan dengan bot lain
    EVENT OnHitBot(e):
        IF locked AND (e.VictimId = lockedTargetId) AND (lockedTargetEnergy <
EnemyRammingThreshold) THEN
            SetForward(50)
        ELSE
            escapeBearing <- NormalizeRelativeAngle(BearingTo(e.X, e.Y) + 180)
            TurnRate <- Clamp(escapeBearing, -MaxTurnRate, MaxTurnRate)
            SetForward(-50)
        ENDIF
    END_EVENT

    // FUNCTION: PredictEnemyPosition - memprediksi posisi musuh berdasarkan
kecepatan dan arah
    FUNCTION PredictEnemyPosition(out predictedX, out predictedY):
        bulletSpeed <- BulletSpeedFactor - (3 * 3)
        timeToImpact <- lockedTargetDistance / bulletSpeed

        predictedX <- lockedTargetX + (lockedTargetVelocity * timeToImpact *
COS(DegreesToRadians(lockedTargetHeading)))
        predictedY <- lockedTargetY + (lockedTargetVelocity * timeToImpact *
SIN(DegreesToRadians(lockedTargetHeading)))

        predictedX <- Clamp(predictedX, 0, ArenaWidth)
        predictedY <- Clamp(predictedY, 0, ArenaHeight)
    END_FUNCTION

    // FUNCTION: DegreesToRadians - mengkonversi derajat ke radian
    FUNCTION DegreesToRadians(degrees):
        RETURN degrees * (PI / 180)
    END_FUNCTION

    // FUNCTION: Clamp - membatasi nilai antara min dan max
    FUNCTION Clamp(value, min, max):
        RETURN MAX(min, MIN(value, max))
    END_FUNCTION
END

```

## B. Penjelasan dari Solusi Greedy yang Dipilih

### B.1 Inisialisasi dan Main Loop

```
MAIN:
    bot <- new EllipsisBot
    bot.Start()

KONSTRUKTOR (EllipsisBot):
    botInfo <- Load("Ellipsis.json")
```

Pada bagian inisialisasi, bot Ellipsis dibuat dengan cara memanggil konstruktor yang memuat informasi konfigurasi dari file "Ellipsis.json". Setelah inisialisasi, bot dijalankan melalui metode utama yang memulai loop eksekusi. Proses ini memastikan bahwa semua parameter yang diperlukan telah disiapkan sebelum bot mulai bergerak dan merespons lingkungan.

### B.2 Loop Utama (Run Method)

```
RUN():
    WHILE IsRunning DO
        RadarTurnRate <- MaxRadarTurnRate

        IF NOT locked THEN
            IF (turnCounter MOD 64 = 0) THEN
                TurnRate <- 5
                TargetSpeed <- MaxSpeed
            ELSE IF (turnCounter MOD 64 = 32) THEN
                TargetSpeed <- -MaxSpeed
            ENDIF
            turnCounter <- turnCounter + 1
        ENDIF

        Go() // Eksekusi pergerakan untuk turn ini
    ENDWHILE
```

Loop utama atau metode **Run()** bertanggung jawab untuk mengontrol pergerakan bot di setiap turn selama pertandingan berlangsung. Pada setiap iterasi, bot mengatur kecepatan putar radar ke nilai maksimal untuk memastikan pendeteksian target yang optimal. Jika tidak ada target yang terkunci, bot akan menggunakan pola pergerakan orbit dengan mengubah kecepatan dan arah secara periodik berdasarkan nilai turn counter. Logika ini memastikan bahwa bot tetap bergerak dan selalu siap untuk mendeteksi bot musuh, sambil menjaga kestabilan dan kelincihan pergerakan.

### B.3 Prediksi Posisi Target

```
PREDICT_POSITION():
    posX <- lockedTargetX + lockedTargetSpeed * SIN(lockedTargetDirection)
    posY <- lockedTargetY + lockedTargetSpeed * COS(lockedTargetDirection)
    RETURN [posX, posY]
```

Fungsi `PREDICT_POSITION()` digunakan untuk menghitung posisi perkiraan target berdasarkan data posisi terkini dan kecepatan serta arah target. Dengan menggunakan fungsi trigonometri, bot menambahkan offset pada posisi X dan Y target yang berasal dari kecepatan dan arah pergerakan target. Hal ini memungkinkan bot untuk memperkirakan posisi target pada turn-turn berikutnya, sehingga dapat menyesuaikan gerakan dan arah tembakan dengan lebih akurat. Fungsi ini merupakan bagian penting untuk meningkatkan efektivitas dalam menyerang target secara tepat waktu.

#### B.4 Penanganan Event Saat Bot Terdeteksi (OnScannedBot)

```
ON_SCANNED_BOT(event):
    scannedDistance <- DistanceTo(event.X, event.Y)

    IF locked AND (event.ScannedBotId = lockedTargetId) THEN
        // Update data target yang sudah terkunci
        lockedTargetX <- event.X
        lockedTargetY <- event.Y
        lockedTargetDistance <- scannedDistance
        lockedTargetSpeed <- event.Speed
        lockedTargetDirection <- event.Direction
    ELSE IF (NOT locked) OR (scannedDistance < lockedTargetDistance) THEN
        // Kunci target baru
        locked <- TRUE
        lockedTargetId <- event.ScannedBotId
        lockedTargetX <- event.X
        lockedTargetY <- event.Y
        lockedTargetDistance <- scannedDistance
        lockedTargetSpeed <- event.Speed
        lockedTargetDirection <- event.Direction
    ENDIF

    pos <- PREDICT_POSITION()

    // Atur gerakan bot berdasarkan posisi target
    bearing <- BearingTo(lockedTargetX, lockedTargetY)
    tangentBearing <- NormalizeRelativeAngle(bearing + 90)
    TurnRate <- CLAMP(tangentBearing, -MaxTurnRate, MaxTurnRate)

    IF ABS(TargetSpeed) < 4 THEN
        TargetSpeed <- 15
    ENDIF

    // Atur pergerakan meriam (gun)
    gunBearing <- NormalizeRelativeAngle(GunBearingTo(pos[0], pos[1]))
    GunTurnRate <- CLAMP(gunBearing, -MaxGunTurnRate, MaxGunTurnRate)

    // Atur pergerakan radar
    radarBearing <- NormalizeRelativeAngle(RadarBearingTo(lockedTargetX,
lockedTargetY))
    RadarTurnRate <- CLAMP(radarBearing, -MaxRadarTurnRate, MaxRadarTurnRate)

    // Tentukan daya tembak berdasarkan jarak target
    IF lockedTargetDistance < 150 THEN
        firePower <- 3
```

```

ELSE
    firePower <- 1
ENDIF

SetFire(firePower)

```

Saat bot mendeteksi keberadaan bot musuh melalui event OnScannedBot, bot mengevaluasi jarak bot musuh yang terdeteksi dan membandingkannya dengan target yang sudah terkunci (jika ada). Jika bot yang terdeteksi adalah target yang telah terkunci, data posisi, kecepatan, dan arah diperbarui. Namun, jika target belum terkunci atau bot musuh yang terdeteksi lebih dekat daripada target sebelumnya, maka bot akan mengunci bot musuh tersebut sebagai target baru. Selanjutnya, bot menghitung nilai-nilai seperti bearing, tangent bearing, dan sudut untuk meriam dan radar, serta menentukan daya tembak berdasarkan jarak target dengan tujuan memperoleh poin sebanyak-banyaknya.

## B.5 Penanganan Event Lain

```

ON_HIT_BY_BULLET(event) :
    TurnRate <- 5

ON_HIT_WALL(event) :
    TargetSpeed <- -TargetSpeed

ON_HIT_BOT(event) :
    TargetSpeed <- -ABS(TargetSpeed)
    PRINT "Collision detected. Executing escape maneuver."

```

Selain menangani event pemindaian bot musuh, bot juga memiliki mekanisme untuk merespons peristiwa lain seperti terkena peluru, menabrak dinding, atau bertabrakan dengan bot lain. Pada saat terkena peluru, bot hanya melakukan penyesuaian kecil pada kecepatan putarnya untuk mengubah arah secara minimal. Jika menabrak dinding, bot akan membalik kecepatan targetnya untuk menghindari kerusakan lebih lanjut. Sementara itu, ketika terjadi tabrakan dengan bot lain, bot akan mengurangi kecepatan secara drastis untuk menghindari kerusakan tambahan dan segera melaksanakan manuver pelarian. Respon cepat terhadap event-event ini membantu bot untuk bertahan dalam kondisi pertempuran yang dinamis dan penuh risiko.

## B.6 Fungsi Utilitas

```

CLAMP(value, min, max) :
    RETURN MAX(min, MIN(value, max))

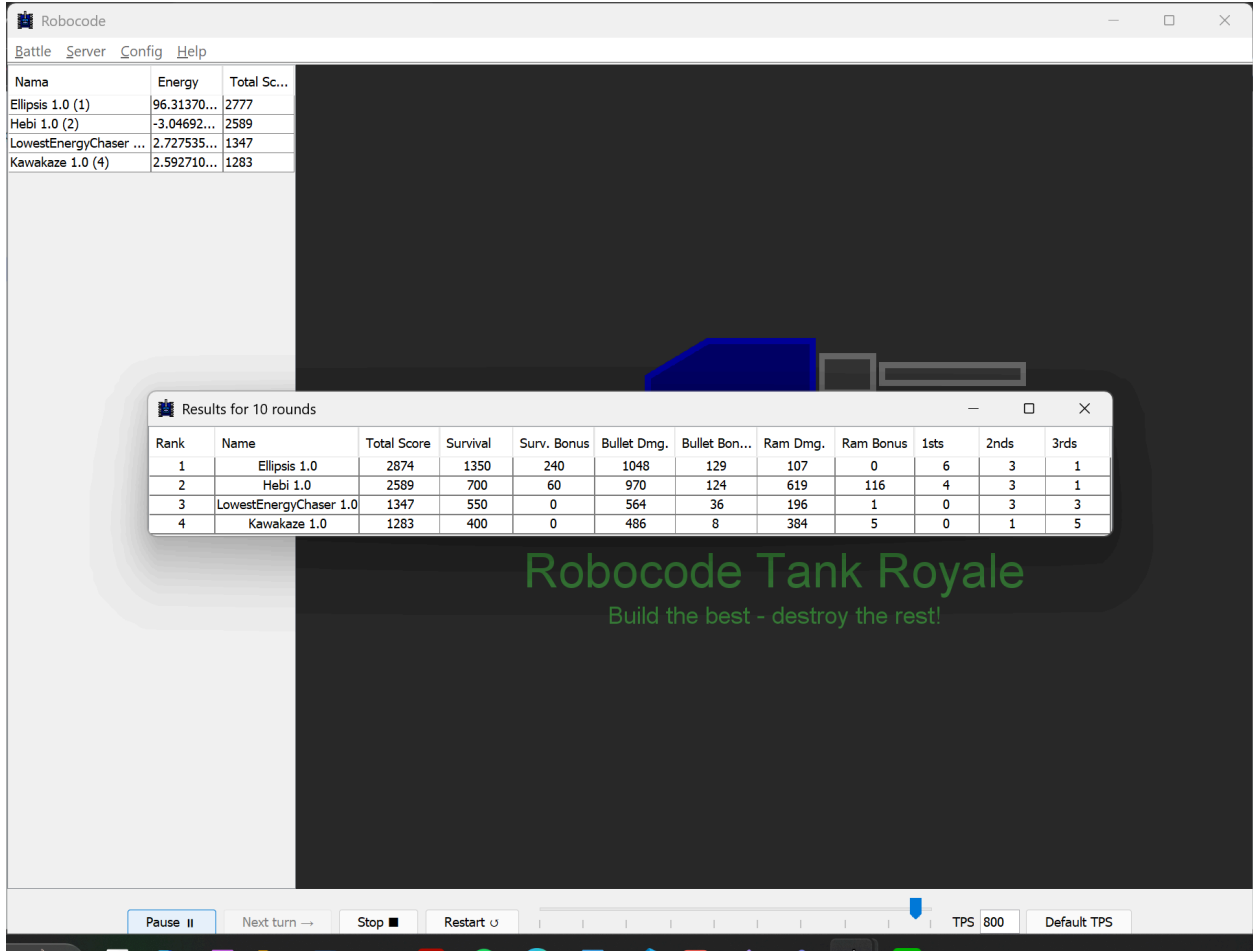
```

Fungsi CLAMP merupakan fungsi utilitas yang berfungsi membatasi nilai dalam rentang yang telah ditentukan. Dengan menggunakan fungsi ini, nilai seperti kecepatan atau sudut putar akan diatur agar tidak melebihi batas minimum dan maksimum yang diperbolehkan. Fungsi ini sangat berguna untuk memastikan bahwa perhitungan dalam bot tidak menghasilkan nilai yang tidak valid atau ekstrem, sehingga menjaga kestabilan pergerakan dan penargetan selama pertandingan.

## C. Analisis dan Pengujian

### Pengujian

#### Pengujian 1



The image shows the Robocode Tank Royale game interface. A results window titled "Results for 10 rounds" is displayed over the game area. The window contains a table with the following data:

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bon...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Ellipsis 1.0	2874	1350	240	1048	129	107	0	6	3	1
2	Hebi 1.0	2589	700	60	970	124	619	116	4	3	1
3	LowestEnergyChaser 1.0	1347	550	0	564	36	196	1	0	3	3
4	Kawakaze 1.0	1283	400	0	486	8	384	5	0	1	5

The background shows the Robocode Tank Royale game area with the text "Robocode Tank Royale" and "Build the best - destroy the rest!". The bottom of the window shows game controls: "Pause II", "Next turn →", "Stop ■", "Restart ↺", a progress bar, "TPS 800", and "Default TPS".

## Pengujian 2

The screenshot shows the Robocode Tank Royale game interface. A results window titled "Results for 10 rounds" is displayed over the game area. The window contains a table with the following data:

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bon...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Hebi 1.0	2685	700	0	968	175	763	78	3	5	1
2	Ellipsis 1.0	2412	1200	240	724	100	148	0	5	3	0
3	Kawakaze 1.0	1999	500	60	705	58	630	46	2	1	4
4	LowestEnergyChaser 1.0	1330	600	0	432	13	281	3	0	1	5

The background game area shows a dark field with a blue tank and a grey tank. The text "Robocode Tank Royale" and "Build the best - destroy the rest!" is visible in green. The bottom of the window has a control bar with buttons: "Pause II", "Next turn →", "Stop ■", "Restart ↺", a progress bar, "TPS 800", and "Default TPS".



## Pengujian 3

The screenshot shows the Robocode Tank Royale interface. The main window displays a battle scene with a blue tank (Ellipsis 1.0) and a grey tank (Hebi 1.0) on a dark field. A menu bar at the top includes Battle, Server, Config, and Help. A table in the top-left corner shows the current battle status:

Nama	Energy	Total Sc...
Ellipsis 1.0 (1)	38.0	2840
Hebi 1.0 (2)	11.50000...	2034
LowestEnergyChaser ...	6.706922...	1619
Kawakaze 1.0 (4)	0.092672...	1687

Overlaid on the battle scene is a 'Results for 10 rounds' window showing a detailed performance table:

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bon...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Ellipsis 1.0	2952	1350	240	1176	177	8	0	7	2	1
2	Hebi 1.0	2034	450	0	781	113	610	80	2	2	4
3	Kawakaze 1.0	1687	500	60	639	47	408	32	1	3	2
4	LowestEnergyChaser 1.0	1619	700	0	780	51	88	0	0	3	3

Below the table, the text 'Robocode Tank Royale' is displayed in green, followed by the tagline 'Build the best - destroy the rest!'. At the bottom of the interface, there are controls for 'Pause II', 'Next turn -->', 'Stop ■', 'Restart ↺', a progress bar, and settings for 'TPS 800' and 'Default TPS'.

## Analisis

Bot Ellipsis berhasil memenangkan 2/3 pertandingan yang memberikan tingkat kemenangan bot Ellipsis sebesar 66,67%. Berdasarkan hasil analisis kami, algoritma greedy dari implementasi kami sudah berhasil dalam sebagian besar kasus dengan mencakup sebagian besar dari kasus yang mungkin terjadi. Namun, bot ellipsis juga gagal pada salah satu pertandingan, analisis kami akan hal ini terjadi ketika bot ellipsis dikepung dan disudutkan. Hal itu merupakan kelemahan bot ini sehingga bot ini sulit untuk mencari jalan keluar.

## BAB V: Kesimpulan dan Saran

### 5.1 Kesimpulan

Melalui Tugas Besar I Strategi Algoritma ini, kami telah berhasil membuat bot untuk permainan Robocode Tank Royale dengan pendekatan algoritma greedy. Program final yang kami buat

merupakan hasil kombinasi dari dua buah solusi greedy yang tersedia dan sudah kami uji untuk menghasilkan strategi terbaik yaitu greedy by distance and diamond type.

## **5.2 Saran**

Bot ini dapat dikembangkan lebih lanjut menggunakan ide-ide lanjutan, misalnya menggunakan algoritma prediksi dalam melakukan penembakan dan algoritma yang dapat melakukan dodge pada bullet yang sedang ditembakkan.

## Lampiran

Repositori Github: [https://github.com/fathurwithyou/Tubes1\\_PolicyGradientStrategist](https://github.com/fathurwithyou/Tubes1_PolicyGradientStrategist)

Video Bonus Pengerjaan: <https://youtu.be/OSRTFWCq1us?si=Axlly0U-XBRKYQ29>

Tabel I. Checklist Pengerjaan

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

## Daftar Pustaka

Robocode Developer. (n.d.). *Namespace Robocode.TankRoyale.BotApi*. API Documentation.

<https://robocode-dev.github.io/tank-royale/api/dotnet/api/Robocode.TankRoyale.BotApi.html>

Wikipedia. (2025, March 5). *Greedy Algorithm*. Wikimedia Foundation, Inc. Retrieved March 23, 2025, from [https://en.wikipedia.org/wiki/Greedy\\_algorithm](https://en.wikipedia.org/wiki/Greedy_algorithm)