

TUGAS BESAR 1
IF2211 STRATEGI ALGORITMA
Pemanfaatan Algoritma *Greedy* dalam Pembuatan *Bot* Permainan
Robocode Tank Royale



Kelompok 48
sucipto

Disusun Oleh:

Rafen Max Alessandro	13523031
Aloisius Adrian Stevan Gunawan	13523054

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

DAFTAR ISI	1
DAFTAR GAMBAR	3
DAFTAR TABEL	4
BAB 1	
DESKRIPSI MASALAH	5
BAB 2	
LANDASAN TEORI	13
2.1. Dasar Teori Algoritma Greedy	13
BAB 3	
APLIKASI STRATEGI GREEDY	19
3.1. Mapping	19
3.2. Alternatif Heuristik Sebagai Dasar Strategi Greedy	20
3.2.1. Attacking First-Scanned	20
a. Mapping Elemen Strategi Greedy	21
b. Efisiensi dan Efektivitas	21
3.2.2. Locking onto a Target	21
a. Mapping Elemen Strategi Greedy	22
b. Efisiensi dan Efektivitas	22
3.2.3. Minimum Distance Targeting	22
a. Mapping Elemen Strategi Greedy	23
b. Efisiensi dan Efektivitas	23
3.2.4. Dynamic movement	23
a. Mapping Elemen Strategi Greedy	24
b. Efisiensi dan Efektivitas	24
3.2.5. Staying Alive	24
a. Mapping Elemen Strategi Greedy	25
b. Efisiensi dan Efektivitas	25
3.3. Pembentukan Strategi Greedy	25
3.3.1. Suniper2	26
3.3.2. NearWall	26
3.3.3. CircleImpact	27
3.3.4. TRIPIN	28
BAB 4	
IMPLEMENTASI DAN PENGUJIAN	29
4.1. Implementasi Algoritma Greedy	29
4.1.1. Suniper2	29
4.1.2. NEARWALL	34

4.1.3. CircleImpact	38
4.1.4. TRIPIN	42
4.2. Analisis dan Pengujian	44
4.2.1. Suniper2	44
4.2.2. NearWall	45
4.2.3. CircleImpact	45
4.2.4. TRIPIN	46
BAB 5	
KESIMPULAN DAN SARAN	48
1. Kesimpulan	48
2. Saran	48
LAMPIRAN	50
1. Tautan repository	50
2. Tautan video	50
3. Checklist	50
DAFTAR PUSTAKA	51

DAFTAR GAMBAR

Gambar 01. Permainan pemrograman Robocode.	5
Gambar 02. Bagian tubuh tank	9
Gambar 03. Sudut pemindaian radar	11
Gambar 04. Sudut pemindaian radar setelah tidak bergerak dalam suatu turn	11
Gambar 05. Contoh fungsi yang disediakan oleh API Robocode	15
Gambar 06. Interface pemilihan bot ke dalam arena	16
Gambar 07. interface terminal menyatakan tindakan yang diambil bot setiap turn	17
Gambar 08. Interface penampilan arena dalam permainan	17
Gambar 09. Tabel tabulasi skor hasil akhir permainan	18
Gambar 10. Nilai akhir pengujian bot Suniper2	44
Gambar 11. Nilai akhir pengujian bot NearWall	45
Gambar 12. Nilai akhir pengujian bot CircleImpact	45
Gambar 13. Nilai akhir pengujian bot TRIPIN	46
Gambar 14. Nilai akhir pengujian seluruh bot sucipto	47

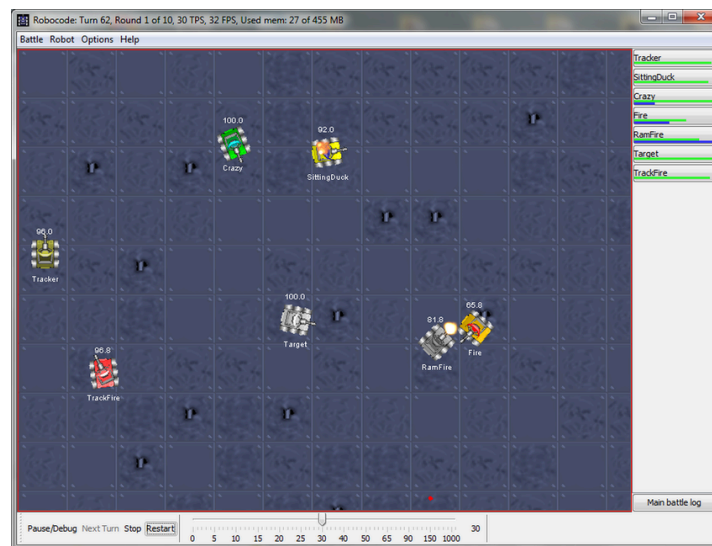
DAFTAR TABEL

Tabel 1. Hasil analisis pengujian setiap algoritma greedy	47
Tabel 2. Checklist	50

BAB 1

DESKRIPSI MASALAH

Tugas besar 1 mata kuliah IF2211 Strategi Algoritma memberikan persoalan berupa penggunaan strategi *greedy* dalam pembentukan sebuah *bot* pada permainan pemrograman Robocode. Dalam permainan Robocode, *bot* diimplementasikan dalam bentuk tank virtual untuk berkompetisi melawan bot lain di suatu arena. Pertempuran Robocode berlangsung hingga *bot-bot* bertarung satu dengan yang lain hingga hanya tersisa satu. Sistem *battle royale* ini menjadi alasan mengapa permainan ini dinamakan Tank Royale. Dalam permainan ini, pemain berperan sebagai *programmer* dari *bot* yang tidak memiliki kendali langsung atas permainan, melainkan terhadap program yang menentukan logika tindakan *bot*, atau dengan kata lain “otak” dari *bot*. Program yang dibuat berisikan instruksi mengenai bagaimana *bot* bergerak, mendeteksi *bot* musuh dalam arena, menembakkan senjatanya, serta interaksi *bot* terhadap berbagai kejadian yang terjadi selama pertempuran.



Gambar 01. Permainan pemrograman Robocode.

Menggunakan pemahaman mengenai strategi *greedy*, mahasiswa diminta untuk membentuk program yang diimplementasikan ke dalam *bot* berdasarkan strategi *greedy*, untuk kemudian dipertandingkan satu dengan yang lain.

Komponen-komponen dari permainan ini antara lain:

1. *Rounds* dan *Turns*

Pertempuran dapat terdiri dari beberapa *rounds*. Secara *default*, satu pertempuran terdiri atas 10 *rounds*, dengan setiap *round* 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 satu *bot* terakhir yang bertahan.

Pada setiap *turn*, sebuah *bot* dapat:

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

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

Pada setiap *turn*, *bot* akan secara otomatis menerima informasi terbaru tentang posisinya dan orientasinya di medan perang. *Bot* juga akan mendapatkan informasi tentang *bot* musuh ketika mereka terdeteksi oleh pemindai. Perlu diketahui bahwa *game engine* yang digunakan pada tugas besar ini tidak mengikuti aturan *default* mengenai komponen *rounds & turns*.

2. Batas Waktu Giliran

Penting untuk dicatat bahwa setiap *bot* memiliki batas waktu untuk setiap *turn* yang disebut dengan istilah *turn timeout*, umumnya berada pada kisaran 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. Daya tembak maksimum yang dapat dikeluarkan oleh *bot* adalah 3 dan daya tembak minimum adalah 0,1. Ketika peluru mengenai musuh, peluru memberikan *damage* kepada musuh sebesar

$$4 \times \text{daya tembak}$$

Jika daya tembak peluru lebih besar dari 1, peluru memberikan *damage* tambahan sebesar

$$2 \times (\text{daya tembak} - 1)$$

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. Kecepatan peluru ditentukan menggunakan perhitungan berikut

$$20 - (3 \times \text{daya tembak})$$

Sehingga kecepatan maksimum yang dapat dimiliki oleh sebuah peluru adalah 19,7 unit per *turn* untuk daya tembak minimum sebesar 0,1, dan kecepatan minimum adalah 11 unit per *turn* untuk daya tembak maksimum sebesar 3.

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. Jumlah panas meriam yang dihasilkan setelah peluru ditembakkan adalah

$$\frac{1 + \text{daya tembak}}{5}$$

dan setiap *round* diawali dengan nilai panas meriam 3 untuk setiap meriam.

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. *Wall damage* memberikan *damage* kepada *bot* sesuai dengan kecepatan *bot* dalam menabrak dinding. *Damage* yang diberikan didasarkan terhadap perhitungan

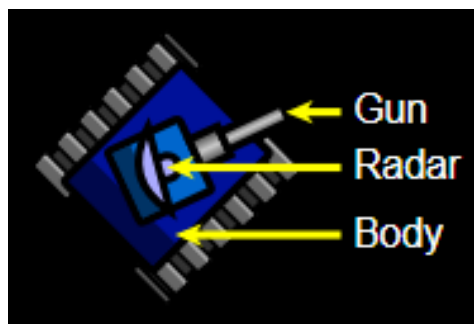
$$\frac{|v|}{2} - 1$$

Dengan v melambangkan kecepatan *bot* saat menabrak dinding dan *damage* negatif akan diubah menjadi nol.

Jika *bot* menabrak *bot* musuh dengan bergerak maju, ini disebut *ramming* (menabrak dengan sengaja), yang akan memberikan sedikit skor tambahan bagi *bot* yang menyerang. Setiap tabrakan memberikan *damage* sebesar 0,6 kepada *bot*, baik yang ditabrak maupun yang menabrak.

7. Bagian Tubuh Tank

Tubuh tank terdiri dari 3 bagian:



Gambar 02. Bagian tubuh tank

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*. Sedangkan *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 *turn* untuk mencapai kecepatan maksimum. *Bot* dapat mengalami percepatan maksimum sebesar 1 unit per *turn* dan pengereman dengan perlambatan maksimum 2 unit per *turn*. Percepatan dan perlambatan maksimum tidak bergantung pada kecepatan *bot* saat itu. Kecepatan maksimum yang dapat dimiliki *bot* adalah 8 unit per *turn*.

9. Berbelok

Seperti yang disebutkan sebelumnya, bagian tubuh, *turret* (meriam), dan radar dapat berputar secara independen antara satu dengan yang lain. Jika *turret* (meriam) 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 *turn*, yang berarti dapat berputar 360 derajat dalam 8 *turn*. *Turret* (meriam) dapat berputar hingga 20 derajat per *turn*.

Bagian paling lambat adalah tubuh tank, yang dalam kondisi terbaik dapat berputar hingga 10 derajat per *turn*. Namun, ini bergantung pada kecepatan *bot* saat ini. Semakin cepat *bot* bergerak, semakin lambat kemampuannya untuk berbelok. Kecepatan perputaran *bot* dihitung berdasarkan

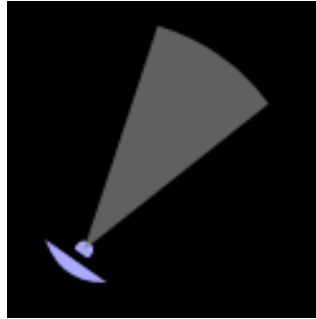
$$10 - \frac{3}{4}|v|$$

dengan v melambangkan kecepatan *bot*. Dengan demikian, jika sebuah *bot* memiliki kecepatan maksimal sebesar 8 unit per *turn*, *bot* tersebut hanya dapat berputar dengan kecepatan perputaran maksimal sebesar 4 derajat per *turn*. Perlu diperhatikan bahwa tidak ada energi yang dikonsumsi saat *bot* bergerak atau berbelok.

10. Pemindaian

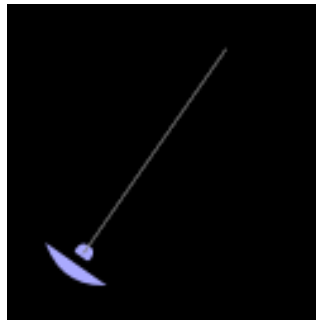
Aspek penting dalam Robocode adalah memindai *bot* musuh menggunakan radar. Radar dapat mendeteksi *bot* dalam jangkauan hingga 1200 piksel. Musuh yang berada lebih dari 1200 piksel dari *bot* tidak dapat terdeteksi atau dipindai oleh radar.

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 *turn*.



Gambar 03. Sudut pemindaian radar

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



Gambar 04. Sudut pemindaian radar setelah tidak bergerak dalam suatu *turn*

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* lainnya yang masih bertahan pada *round* tersebut mendapatkan **50 poin**;
- **Last Survival Bonus:** *Bot* terakhir yang bertahan pada suatu *round* 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 permainan akan menampilkan berapa kali suatu *bot* meraih peringkat 1, 2, atau 3 pada setiap *round*. Namun, hal ini tidak dihitung sebagai komponen skor maupun untuk perangkingan akhir. *Bot* yang dianggap menang pertempuran adalah *bot* dengan akumulasi skor tertinggi.

BAB 2

LANDASAN TEORI

2.1. Dasar Teori Algoritma Greedy

Algoritma *greedy* merupakan salah satu pendekatan algoritma yang populer untuk digunakan dalam persoalan optimasi sederhana. Menggunakan algoritma *greedy*, dapat ditemukan solusi yang relatif optimal, baik dalam konteks maksimasi (*maximization*) maupun minimasi (*minimization*). Algoritma ini didasarkan terhadap konsep “*taking what can be taken now*”, mengambil pilihan terbaik di setiap langkah, dengan harapan memberikan solusi optimal di akhir penyelesaian.

Strategi yang digunakan dalam algoritma *greedy* tidak mempertimbangkan konsekuensi jangka panjang secara keseluruhan, tetapi mengutamakan perolehan terbesar yang paling mungkin dalam suatu langkah. Strategi ini memberikan karakteristik algoritma yang cepat dan efisien secara waktu karena tidak mengevaluasi semua kemungkinan yang mungkin, tidak seperti algoritma *brute force*.

Walaupun algoritma *greedy* tidak dapat diandalkan untuk memberikan solusi terbaik secara mutlak setiap saat, algoritma tetap dapat digunakan untuk menghasilkan solusi hampiran (*approximation*) yang mendekati solusi optimal pada suatu persoalan, memberikan keuntungan waktu dibandingkan menggunakan algoritma yang kebutuhan waktunya eksponensial untuk menghasilkan solusi yang eksak. Sebagai contoh, untuk menyelesaikan persoalan *traveling salesperson*, daripada menggunakan algoritma *brute force* yang membutuhkan waktu komputasi lama, lebih baik menggunakan algoritma *greedy* untuk memberikan hampiran solusi optimal.

Untuk mendefinisikan sebuah algoritma *greedy*, diperlukan definisi eksplisit mengenai enam komponen elemen berikut:

1. Himpunan kandidat

Himpunan berisi kandidat yang dapat dipilih pada setiap langkah, misalnya simpul dalam sebuah graf, koin, benda, dsb.

2. Himpunan solusi

Himpunan berisi kandidat yang telah dipilih, telah dianggap sebagai pilihan optimal pada langkah sebelumnya.

3. Fungsi solusi

Fungsi yang menentukan apakah himpunan solusi telah memberikan solusi secara keseluruhan sesuai dengan persoalan.

4. Fungsi seleksi (*selection function*)

Fungsi yang digunakan untuk memilih kandidat berdasarkan sebuah strategi *greedy* yang bersifat heuristik, yaitu didasarkan berdasarkan aturan praktis untuk setiap langkahnya, bukan perhitungan optimal secara menyeluruh.

5. Fungsi kelayakan (*feasible*)

Fungsi yang memeriksa apakah kandidat yang dipilih memenuhi kondisi kelayakan untuk dimasukkan ke dalam himpunan solusi.

6. Fungsi obyektif

Fungsi yang menyatakan tujuan dari algoritma, apakah untuk memaksimumkan atau meminimumkan.

Menggunakan pendekatan algoritma *greedy*, dapat diterapkan alur logika pengendalian *bot* dalam permainan pemrograman Robocode. Strategi *greedy* digunakan sebagai dasar pengambil keputusan oleh *bot* dalam sebuah *turn* berdasarkan keuntungan yang terlihat langsung, yaitu memberikan poin paling besar, tanpa mempertimbangkan dampak jangka panjang terhadap dinamika permainan secara keseluruhan.

2.2. Cara Kerja Robocode

Secara umum, Robocode menggunakan algoritma dan alur logika yang diatur secara unik untuk setiap *bot*. *Bot* diprogram untuk melakukan berbagai bentuk tindakan yang diizinkan dalam suatu permainan, seperti bergerak ke arah tertentu, menembak dengan kekuatan peluru tertentu, menghindari lawan, dan berbagai macam pilihan untuk dilakukan dalam suatu *turn*. *Bot* juga dapat mendasarkan jalur logika berdasarkan kondisi yang terjadi di arena, seperti ketika *bot* menabrak dinding, peluru dari *bot* mengenai musuh, atau *bot* menerima serangan dari musuh.

Menggunakan pemacu-pemacu untuk tindakan tersebut, sebuah *bot* dapat diprogram untuk memberikan kemenangan di setiap *round*, mengupayakan kemenangan di akhir permainan yang umumnya terdiri atas 10 *round*.

Pengguna dapat menggunakan API Robocode yang menyediakan banyak fungsi-fungsi untuk digunakan dalam mengatur algoritma dan alur logika setiap *bot*. Maka, penerapan implementasi strategi *greedy* ke dalam *bot* dilakukan dengan mengaplikasikan fungsi-fungsi yang sesuai berdasarkan dasar heuristik strategi *greedy*.

[View Source](#) | [Edit this page](#)

Forward(double)

Moves the bot forward until it has traveled a specific distance from its current position, or it is moving into an obstacle. The speed is limited by [MaxSpeed](#).

When the bot is moving forward, the [Acceleration](#) determine the acceleration of the bot that adds 1 additional unit to the speed per turn while accelerating. However, the bot is faster at braking. The [Deceleration](#) determines the deceleration of the bot that subtracts 2 units from the speed per turn.

This call is executed immediately by calling [Go\(\)](#) in the code behind. This method will block until it has been completed, which can take one to several turns. New commands will first take place after this method is completed. If you need to execute multiple commands in parallel, use *setter* methods instead of this blocking method.

This method will cancel the effect of prior calls to [TargetSpeed](#), [SetForward\(double\)](#), and [SetBack\(double\)](#) methods.

Declaration

```
public void Forward(double distance)
```

Parameters

Type	Name	Description
double	distance	Is the distance to move forward. If negative, the bot will move backward. If PositiveInfinity the bot will move forward infinitely. If NegativeInfinity the bot will move backward infinitely.

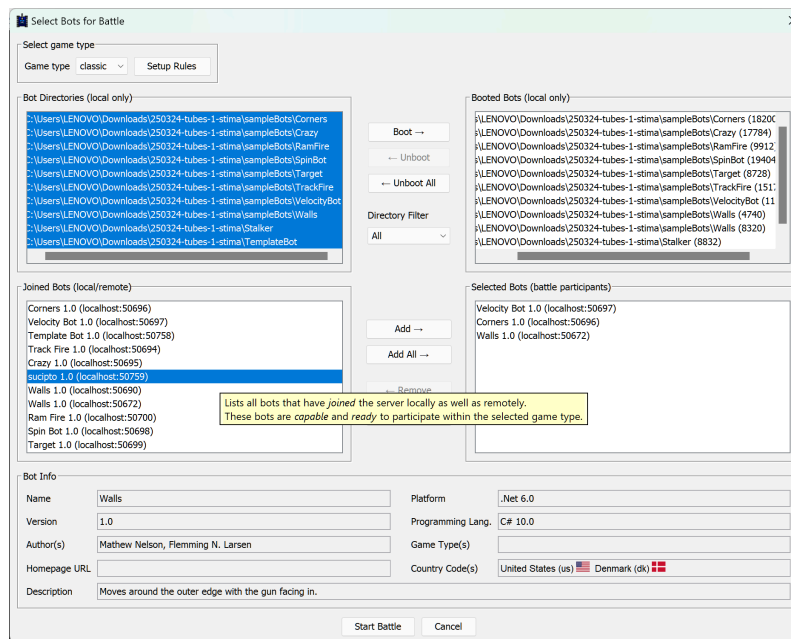
See Also

- [SetForward\(double\)](#)
- [SetBack\(double\)](#)
- [Back\(double\)](#)
- [DistanceRemaining](#)
- [TargetSpeed](#)

Gambar 05. Contoh fungsi yang disediakan oleh API Robocode

Karena algoritma *greedy* mengutamakan untuk mengambil setiap optimum lokal (*local optimum*) yang paling menguntungkan di setiap langkah, dibandingkan mengusahakan spekulasi logis untuk mengutamakan solusi optimum global (*global optimum*), maka pembentukan algoritma dan alur logika dalam program cenderung sederhana, dengan setiap pilihan hanya didasarkan terhadap data yang diperoleh pada *turn* itu dan tidak mengandalkan data sebelumnya. Namun, menggunakan perhitungan lebih lanjut, dapat dilakukan prediksi sederhana menggunakan data yang diperoleh dalam satu *turn* untuk memperkirakan kondisi pada *turn* berikutnya, sehingga dasar yang digunakan untuk mengambil langkah algoritma *greedy* berikutnya menjadi lebih maju daripada sekadar mengandalkan data dalam suatu *turn* saja.

Setelah program dapat di-*compile* dan dipastikan berjalan, menggunakan perintah “dotnet build”, *bot* dapat di-*compile* ke dalam Robocode untuk dimasukkan ke dalam suatu permainan. Jika tahapan *compile* dilakukan dengan benar, *bot* akan masuk ke dalam *local bot directories* yang dapat di-*boot* secara lokal, untuk kemudian masuk ke dalam bagian *joined bots* yang berisikan kumpulan *bot* yang telah di-*boot* baik secara lokal maupun *remote*. Dengan memilih *bot* pada bagian *joined bots*, *bot* tersebut akan dimasukkan ke dalam arena untuk permainan selanjutnya.



Gambar 06. Interface pemilihan *bot* ke dalam arena

Dengan menekan tombol ‘Start Battle’, pengguna akan memulai permainan baru dengan seluruh *bot* yang telah dimasukkan ke dalam arena. Permainan dimulai pada *round* 1 dan *turn* 1, dengan setiap *bot* mengambil pilihan setiap langkah pada sebuah *turn* dan melangsungkan pilihan tersebut dalam beberapa *turn* kedepannya, sesuai dengan algoritma dan alur logika yang mendasarinya.

Dengan menekan tombol ‘Start Battle’, pengguna akan memulai permainan baru dengan seluruh *bot* yang telah dimasukkan ke dalam arena. Permainan dimulai pada *round* 1 dan *turn* 1, dengan setiap *bot* mengambil pilihan setiap langkah pada sebuah *turn* dan melangsungkan pilihan tersebut dalam beberapa *turn* kedepannya, sesuai dengan algoritma dan alur logika yang mendasarinya. *Interface* permainan juga menampilkan nama, energi, dan skor total yang dimiliki

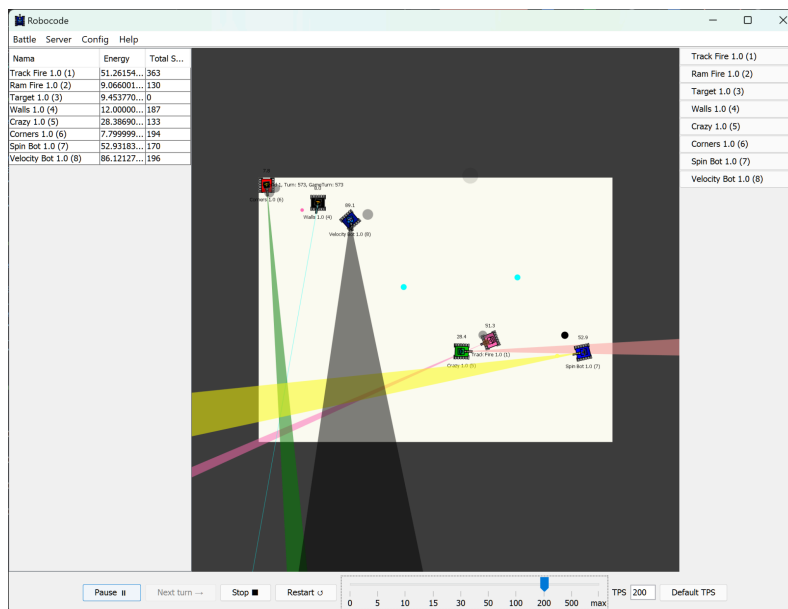
oleh suatu *bot* ketika *round* berlangsung, sehingga keberlangsungan permainan dapat diperhatikan oleh pengguna secara *real-time*.

```

sucipto 1.0 (2)
Console Properties Events
419 preparing to shoot
423 preparing to shoot
435 preparing to shoot
449 preparing to shoot
459 preparing to shoot
462 preparing to shoot
474 Making circles rn, going right
476 Making circles rn, going right
Bot found at (419,5115790463175, 77,28811811286006)
490 calling moveToCenter()
Going to the center, turning: -141,9179854020827 degrees
506 Bot found at (341,94929751457835, 45,03765325433058)
563 preparing making circle
571 Ouch! I hit a wall, must turn back!
581 Ouch! I hit a wall, must turn back!
599 preparing to shoot
630 calling moveToCenter()
Going to the center, turning: -81,3432645815224 degrees
OK Clear Copy to clipboard

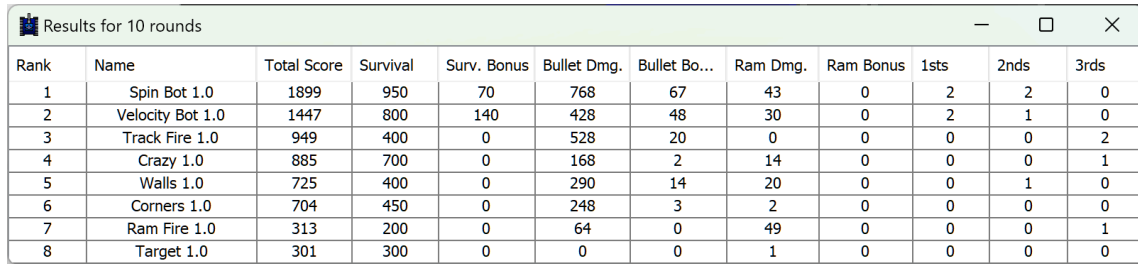
```

Gambar 07. *interface* terminal menyatakan tindakan yang diambil *bot* setiap *turn*



Gambar 08. *Interface* penampilan arena dalam permainan

Satu permainan selesai setelah *round* ke-10 telah memberikan pemenang, dengan hasil dari permainan ditampilkan dalam bentuk tabel yang menabulasikan data-data selama permainan berlangsung, serta berapa kali sebuah *bot* mendapatkan peringkat satu, dua, atau tiga dalam sebuah *round*.



Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Spin Bot 1.0	1899	950	70	768	67	43	0	2	2	0
2	Velocity Bot 1.0	1447	800	140	428	48	30	0	2	1	0
3	Track Fire 1.0	949	400	0	528	20	0	0	0	0	2
4	Crazy 1.0	885	700	0	168	2	14	0	0	0	1
5	Walls 1.0	725	400	0	290	14	20	0	0	1	0
6	Corners 1.0	704	450	0	248	3	2	0	0	0	0
7	Ram Fire 1.0	313	200	0	64	0	49	0	0	0	1
8	Target 1.0	301	300	0	0	0	1	0	0	0	0

Gambar 09. Tabel tabulasi skor hasil akhir permainan

Dalam permainan Robocode, hasil dari permainan dapat diterima secara mentah sebagai hasil akhir. Namun, untuk tujuan membentuk strategi *greedy* terbaik, hasil ini dapat diolah secara semantik untuk melihat kelebihan dan kekurangan dari algoritma dan alur logika yang telah diimplementasikan. Interaksi menang dan kalah terhadap *bot* musuh dapat menjadi tolak ukur faktor-faktor yang dapat diperbaiki dalam sebuah *bot*, sehingga iterasi permainan dapat meningkatkan kualitas dan kinerja dari algoritma *greedy* yang dibentuk.

BAB 3

APLIKASI STRATEGI *GREEDY*

3.1. Mapping

Pembentukan bot di robocode didasarkan terhadap beberapa pertimbangan atas banyaknya kejadian yang dapat dilakukan oleh sebuah *bot*. Maka dari itu, diperlukan *mapping* secara formal yang memetakan elemen-elemen strategi algoritma *greedy* terhadap komponen yang dimiliki dalam Robocode. Beberapa komponen yang dapat dimanfaatkan sebagai dasar pembentukan algoritma *greedy*, di antaranya:

- Bagaimana *bot* bergerak (maju, mundur, memutar *body*, dsb.);
- Bagaimana *bot* mendeteksi musuh (radar diputar terus-menerus, melakukan *scan* dalam keadaan tertentu, dsb.);
- Bagaimana *bot* menembak (arah *gun* menembak, kecepatan perputaran *gun*, dsb.);
- Manajemen energi (kekuatan menembak, *survivability*, mengatur posisi terhadap musuh dan dinding, dsb.); dan
- Strategi mencari poin (mengusahakan tembakan untuk mengenai musuh, melakukan *ramming*, mengutamakan untuk bertahan hidup, dsb.).
- dan berbagai macam komponen lainnya.

Jika dipetakan ke dalam elemen strategi *greedy*, secara umum setiap algoritma *greedy* akan membentuk pengelompokkan elemen sebagai berikut:

1. Himpunan kandidat

Dalam setiap *turn*, sebuah *bot* dapat memilih banyak langkah untuk dilakukan. Seperti yang telah dijelaskan sebelumnya, kandidat langkah yang dipilih terdiri atas menggerakkan *bot*, menggerakkan radar, menggerakkan *gun*, menembakkan peluru dari *gun*, melakukan kombinasi empat langkah tersebut, atau tidak mengambil langkah.

2. Himpunan solusi

Hasil dari langkah *bot* yang dianggap telah menjadi pilihan yang memberikan optimal lokal (*local optimum*) dalam suatu *turn* akan sesuai dengan algoritma dan alur logika yang diterapkan dalam strategi *greedy*.

3. Fungsi solusi

Memeriksa apakah untuk setiap *round* yang telah dilewati *bot*, strategi *greedy* telah berjalan sebagaimana mestinya dan memberikan optimum lokal (*local optimum*) sesuai implementasi algoritma dan alur logika.

4. Fungsi seleksi

Pemilihan tindakan yang dilakukan oleh *bot* dalam suatu *turn* akan sesuai dengan algoritma dan alur logika yang diterapkan dalam strategi *greedy*, yang turut menyatakan definisi optimum lokal (*local optimum*) untuk strategi *greedy* yang digunakan.

5. Fungsi kelayakan

Memeriksa apakah langkah yang dilakukan oleh *bot* sesuai dengan dasar strategi *greedy* yang diimplementasikan.

6. Fungsi obyektif

Mendapatkan skor tertinggi sebagai optimum global (*global optimum*) di akhir permainan dengan memanfaatkan metode-metode untuk mendapatkan skor dalam Robocode, seperti mengenai musuh dengan peluru, melakukan *ramming* hingga *bot* hancur, bertahan hidup selama mungkin, dan lain sebagainya.

Definisi untuk setiap elemen strategi *greedy* di atas masih bersifat umum dan abstrak. Oleh karena itu, dibutuhkan heuristik yang berperan sebagai dasar dari strategi *greedy* yang diimplementasikan ke dalam *bot* untuk memberikan definisi yang lebih spesifik dan definit mengenai setiap elemen.

3.2. Alternatif Heuristik Sebagai Dasar Strategi Greedy

Berdasarkan kajian, analisis, dan eksperimen yang telah kelompok lakukan, didapatkan beberapa heuristik berikut sebagai heuristik yang baik untuk diimplementasikan sebagai dasar dari strategi Greedy.

3.2.1. *Attacking First-Scanned*

Robocode menggunakan radar untuk mendapatkan informasi mengenai posisi *bot* musuh. Informasi ini didapatkan apabila radar diputar dan garis batas pandangan radar

menyentuh bot lawan. Lalu, Robocode juga memiliki variabel *gun heat* yang menyebabkan waktu jarak interval tembak untuk setiap besar peluru berbeda berdasarkan kekuatan peluru yang ditembak. Oleh karena itu, dapat dibentuk prinsip heuristik dalam bentuk ‘melakukan penyerangan terhadap *bot* musuh yang pertama ditemukan dengan kekuatan peluru minimal memberikan kesempatan menembak sebanyak mungkin.’.

a. *Mapping* Elemen Strategi Greedy

1. Himpunan kandidat: seluruh musuh yang terdeteksi oleh radar dalam satu *turn*.
2. Himpunan solusi: tembakan yang berhasil mengenai musuh pertama yang ter-*scan* oleh radar.
3. Fungsi solusi: melangsungkan tembakan kepada musuh pertama yang berhasil ter-*scan* oleh radar.
4. Fungsi seleksi: memilih musuh pertama yang berhasil ter-*scan* oleh radar untuk ditembak.
5. Fungsi kelayakan: memastikan apakah posisi dari musuh yang dijadikan target baik untuk ditembak berdasarkan arah penembakan dan energi yang dimiliki *bot*.
6. Fungsi objektif: memberikan *damage* kepada *bot* musuh secepat mungkin.

b. Efisiensi dan Efektivitas

Strategi ini memberikan kecepatan serangan tinggi yang efektif karena variabel *gun heat* kecil yang lebih cepat berkurang dan memberikan *damage* kecil secara terus-menerus, membentuk strategi agresif untuk bot dalam melakukan penyerangan. Namun strategi ini memiliki kelemahan karena pergerakan bot musuh tidak dijadikan konsiderasi, sehingga akurasi penyerangan tidak dapat diandalkan akan selalu mengenai musuh dan menghabiskan energi dalam jumlah yang signifikan.

3.2.2. *Locking onto a Target*

Radar memberikan informasi mengenai posisi *bot* musuh. Maka, jika radar diarahkan untuk selalu mengarah kepada target musuh yang sama, *bot* dapat menyerang satu musuh tersebut secara berulang kali hingga hancur. Setiap serangan yang dilakukan

dapat disesuaikan untuk memberikan dampak maksimal kepada target, baik berdasarkan jarak *bot* terhadap target, kecepatan dan kekuatan peluru yang ditembakkan, dan faktor penyerangan lainnya. Oleh karena itu, dapat dibentuk prinsip heuristik dalam bentuk ‘melakukan serangan secara terus-menerus terhadap satu target hingga target hancur, lalu mencari target yang lain.’.

a. *Mapping* Elemen Strategi Greedy

1. Himpunan kandidat: seluruh musuh yang terdeteksi oleh radar dalam satu *turn*.
2. Himpunan solusi: tembakan yang berhasil mengenai target hingga target tereliminasi.
3. Fungsi solusi: melangsungkan tembakan kepada target hingga target tereliminasi.
4. Fungsi seleksi: memilih musuh yang paling optimal sebagai target berdasarkan faktor-faktor penembakan.
5. Fungsi kelayakan: memastikan apakah posisi dari musuh yang dijadikan target baik untuk ditembak berdasarkan arah penembakan dan energi yang dimiliki *bot*.
6. Fungsi objektif: mengeliminasi seluruh musuh di arena satu persatu.

b. Efisiensi dan Efektivitas

Strategi sangat bergantung terhadap keberadaan musuh yang berhasil di-*scan* oleh radar dan sensitif terhadap pergerakan musuh yang sulit untuk diprediksi, sehingga strategi sangat mudah untuk melakukan penembakan yang tidak mengenai musuh dan menghabiskan energi secara sia-sia. Namun, jika *bot* musuh memiliki pergerakan yang minimal, misalnya *stuck* atau telah kehabisan energi, maka strategi menjadi sangat efektif dalam mengeliminasi musuh dan mendapatkan poin tambahan.

3.2.3. *Minimum Distance Targeting*

Semakin besar peluru yang ditembakkan, maka semakin pelan peluru tersebut bergerak. Sehingga untuk memastikan serangan besar dapat mengenai *bot* musuh, jarak untuk melakukan penyerangan perlu seminimal mungkin. Melakukan penyerangan terhadap *bot* musuh yang paling dekat akan memberikan tingkat akurasi yang lebih tinggi

sehingga kerugian bagi *bot* dalam menembakkan peluru dengan daya lebih besar apabila meleset dapat diminimalisir. Oleh karena itu, dapat dibentuk prinsip heuristik dalam bentuk ‘memilih target dengan jarak paling kecil untuk diserang dengan daya tembak paling optimal.’.

a. *Mapping* Elemen Strategi *Greedy*

1. Himpunan kandidat: seluruh musuh yang terdeteksi oleh radar dalam satu *turn*.
2. Himpunan solusi: serangan dengan peluru besar yang telah mengenai musuh dengan jarak paling dekat.
3. Fungsi solusi: menembakkan serangan dengan peluru besar kepada musuh dengan jarak paling dekat.
4. Fungsi seleksi: memilih *bot* musuh dengan jarak paling kecil sebagai target utama.
5. Fungsi kelayakan: memastikan bahwa tingkat akurasi penembakan cukup tinggi untuk diandalkan dalam melakukan penembakan dengan peluru besar.
6. Fungsi objektif: memaksimalkan *damage* dan jumlah energi yang dikembalikan dengan menyerang target yang paling mudah dikenai.

b. Efisiensi dan Efektivitas

Pemilihan musuh dapat dilakukan dengan mudah di setiap *turn* dengan menghitung jarak *bot* dengan setiap *bot* musuh di arena saat ini. Memastikan peluru dengan kekuatan besar dapat mengenai *bot* musuh juga memberikan pengembalian energi dalam jumlah yang besar sehingga mendukung *survivability* dari *bot*.

3.2.4. *Dynamic movement*

Serangan peluru dari *bot* hanya dapat bergerak dalam garis lurus, sehingga pergerakan secara dinamis dapat digunakan untuk menghindari peluru dari tembakan musuh, seperti pergerakan zig-zag, berputar, atau bergerak lurus dengan variasi kecepatan. Pergerakan yang sulit untuk diprediksi diharapkan dapat mengurangi kemungkinan terpapar oleh peluru serangan musuh. Oleh karena itu, dapat dibentuk

prinsip heuristik dalam bentuk ‘melakukan gerakan dinamis secara kontinu untuk mengurangi peluang terkena serangan musuh untuk meningkatkan *survivability*’.

a. *Mapping* Elemen Strategi Greedy

1. Himpunan kandidat: seluruh pilihan gerakan yang dapat dilakukan oleh *bot* dalam satu *turn*.
2. Himpunan solusi: pergerakan yang paling efektif dalam menghindari tembakan musuh.
3. Fungsi solusi: melangsungkan pola gerakan yang berubah secara dinamis untuk menghindari tembakan musuh.
4. Fungsi seleksi: memilih gerakan yang memberikan peluang terkena tembakan musuh paling kecil berdasarkan posisi musuh saat ini.
5. Fungsi kelayakan: memastikan bahwa gerakan yang *bot* lakukan valid dan tidak menghentikan pergerakan *bot*, misalnya secara sengaja menabrak *body* ke dinding arena.
6. Fungsi objektif: meningkatkan *survivability bot* dengan meminimalkan kekurangan energi *bot* sebagai dampak dari terkena serangan musuh.

b. Efisiensi dan Efektivitas

Untuk memastikan pergerakan *bot* memberikan keuntungan maksimal, diperlukan kalkulasi posisi musuh di arena secara bersamaan dan terus-menerus. Hal ini memberikan efektivitas yang sangat tinggi untuk melindungi *bot* dari pola serangan berbasis prediksi lurus, meskipun memberikan pendapatan poin minimal karena *bot* mengutamakan *survivability* dibandingkan *fight power*.

3.2.5. *Staying Alive*

Mengutamakan prioritas untuk bertahan hidup dapat dilakukan dengan memilih tindakan yang mengurangi risiko terkena serangan. *Bot* dapat mengutamakan pola pergerakan defensif seperti menghindar dan menjauh dari kerumunan untuk mengumpulkan poin bertahan hidup. Oleh karena itu, dapat dibentuk prinsip heuristik

dalam bentuk ‘mengambil langkah defensif untuk meminimalkan risiko terkena serangan dan bertahan hidup selama mungkin.’.

a. *Mapping* Elemen Strategi *Greedy*

1. Himpunan kandidat: seluruh langkah yang dapat mengurangi risiko terkena serangan.
2. Himpunan solusi: rangkaian langkah yang telah diambil dengan tujuan memperpanjang durasi hidup *bot*.
3. Fungsi solusi: memilih langkah yang ditujukan untuk memperpanjang durasi hidup *bot*.
4. Fungsi seleksi: memilih langkah yang memberikan kemungkinan bertahan hidup paling besar.
5. Fungsi kelayakan: memastikan bahwa langkah yang diambil tidak membawa *bot* ke dalam posisi yang lebih berisiko terkena serangan dari musuh.
6. Fungsi objektif: meningkatkan *survivability bot* dengan meminimalkan kekurangan energi *bot* sebagai dampak dari terkena serangan musuh.

b. Efisiensi dan Efektivitas

Dengan memfokuskan *bot* untuk bertahan hidup, *survival points* yang *bot* dapatkan bernilai tinggi, melalui skor bonus yang didapatkan untuk setiap *bot* yang hancur dalam sebuah *round* dan skor bonus *last survivor*. Namun, skor yang didapatkan melalui metode ini tidak dapat diandalkan untuk melebihi skor yang didapatkan dengan bermain secara *offensive*.

3.3. Pembentukan Strategi Greedy

Berdasarkan analisis efisiensi dan efektivitas yang telah dilakukan terhadap setiap heuristik sebagai dasar pembentukan strategi greedy, maka kelompok melakukan kombinasi terhadap beberapa heuristik untuk membentuk empat strategi sebagai berikut:

3.3.1. Suniper2

Sesuai dengan makna semantik nama yang diberikan, seorang penembak jitu (*sniper*) mencari tempat yang aman sebelum menembak musuh dengan tingkat keakuratan yang tinggi. Maka, *bot* ini memulai langkah dengan menentukan lokasi yang dianggap aman. Apabila *bot* musuh bergerak mendekati lokasi tertentu, maka setelah jarak mencapai batasan tertentu, *bot* ini akan bergerak menjauhinya.

Bot Suniper2 akan melancarkan penembakan dengan memprediksi posisi lawan berdasarkan kecepatan peluru yang digunakan dan kecepatan lawan pada *turn* tersebut. Hal ini dilakukan untuk mengurangi kemungkinan peluru meleset dan terbuang sia-sia. Kekuatan dari peluru yang ditembakkan juga bergantung kepada energi yang dimiliki *bot* saat itu, sehingga pada kondisi energi rendah, *bot* akan mengatur penggunaan energi dengan lebih efisien.

Dasar heuristik yang digunakan sebagai dasar dari algoritma dan alur logika *bot* Suniper2 adalah:

1. *Attacking First-scanned* → memilih *bot* musuh yang pertama ter-*scan* sebagai target;
2. *Dynamic Movement* → melakukan pergerakan zig-zag dengan waktu *pause* untuk menghindari peluru tembakan *bot* musuh;
3. *Staying alive.* → menjauhi *bot* musuh jika jarak antar *bot* terlalu dekat.

3.3.2. NearWall

Kerumunan di tengah arena sangat umum terjadi dalam arena yang diisi dengan banyak *bot*. Maka, *bot* NearWall akan menghindari kerumunan tersebut dengan berjalan mengelilingi dinding, menjauhi kerumunan yang terjadi di tengah *bot* lawan. Pengaturan dilakukan terhadap pergerakan *bot* sehingga *bot* tidak menabrakan *body* ke dinding arena, serta terhadap *gun* dan radar dari *bot* untuk selalu mengarah ke tengah arena dan menembakkan peluru setiap kali radar berhasil melakukan *scan* terhadap *bot* musuh. Jika NearWall bertabrakan dengan *bot* lain selama mengelilingi dinding, *bot* akan mengubah orientasi perputaran dari searah jarum jam menjadi berlawanan jarum jam, dan sebaliknya.

Dasar heuristik yang digunakan sebagai dasar dari algoritma dan alur logika *bot* NearWall adalah:

1. *Attacking First-scanned* → melakukan penyerangan terhadap *bot* musuh yang berhasil di-*scan* oleh radar;
2. *Staying Alive* → menjauhi kondisi-kondisi dimana *bot* rentan terkena serangan.

3.3.3. CircleImpact

Memanfaatkan fakta krusial bahwa peluru yang telah dilontarkan hanya dapat bergerak lurus, *bot* CircleImpact menggunakan gerakan melingkar untuk menghindari peluru yang tersebar di arena. *Bot* juga memanfaatkan dasar heuristik bahwa *bot* pada umumnya akan mengeluarkan banyak serangan ketika *round* dimulai, khususnya dalam arena dengan banyak *bot*. Maka, *bot* CircleImpact menjawab hal ini dengan menjadi *pacifist* untuk kondisi tertentu, yaitu 200 *turns* pertama dalam sebuah *round* jika jumlah *bot* musuh pada permainan lebih sedikit dari lima, atau jumlah *bot* musuh tinggal tersisa lima untuk permainan dengan banyak *bot* musuh. *Bot* tidak akan melakukan serangan dan hanya berputar-putar dalam lintasan yang sulit diprediksi selama mode *pacifist*.

Setelah kondisi permainan mencapai situasi yang ditentukan, *bot* CircleImpact akan berhenti menjadi *pacifist* dan mulai melakukan penyerangan terhadap *bot* musuh yang berada dalam jarak optimal untuk diserang, sembari mempertahankan pergerakan melingkar yang sulit diprediksi untuk menghindari kemungkinan terkena serangan peluru *bot* musuh.

Dasar heuristik yang digunakan sebagai dasar dari algoritma dan alur logika *bot* CircleImpact adalah:

1. *Dynamic Movement* → melakukan pergerakan yang sulit diprediksi untuk mengurangi kemungkinan terkena peluru tembakan *bot* musuh;
2. *Staying Alive* → bersikap *pacifist* untuk sejumlah *turn* pertama;
3. *Minimum distance targeting* → melakukan penyerangan terhadap *bot* musuh yang berada dalam jarak optimal terhadap *bot*.

3.3.4. TRIPIN

Nama dari *bot* ini diambil dari pergerakan yang dilakukan oleh *bot*, yaitu *triangle spin* yang disingkat menjadi TRIPIN. *Bot* TRIPIN memanfaatkan dasar heuristik pergerakan melingkar yang sulit untuk dikenai peluru. Namun, jika terlalu lama tinggal di satu tempat, masih terdapat kemungkinan yang cukup tinggi untuk *bot* dikenai peluru yang telah ditembakkan sebelumnya. Maka, *bot* TRIPIN akan menentukan tiga lokasi untuk melakukan lintasan melingkar, dengan ketiga lokasi yang ditentukan membentuk segitiga.

Dasar heuristik yang digunakan sebagai dasar dari algoritma dan alur logika *bot* CircleImpact adalah:

1. *Attacking First-Scanned* → melakukan penyerangan terhadap *bot* musuh yang ter-*scan* pertama kali oleh radar;
2. *Dynamic Movement* → melakukan pergerakan yang sulit diprediksi untuk mengurangi kemungkinan terkena peluru tembakan *bot* musuh;

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi Algoritma *Greedy*

4.1.1. Suniper2

1. Inisialisasi variabel

- Bool runWall: menyatakan True apabila menabrak dinding.
- Int maju: Untuk membuat pola maju-diam-mundur-diam
- Bool shoot: Bernilai True apabila menemukan target untuk ditembak
- Bool startCheck: bernilai True apabila pernah menemukan bot lawan
- ScannedBotEvent en: Digunakan untuk mengolah data lawan terdekat
- ScannedBotEvent enSHOOT: digunakan untuk memprediksi lokasi masa depan musuh yang akan ditembak
- Double minX: bernilai lokasi ancaman terdekat terakhir
- Double minY: bernilai lokasi ancaman terdekat terakhir
- Double minDist: bernilai nilai jarak dari ancaman terdekat terakhir

```
SET runWall = false
SET maju = 0
SET shoot = false
SET startCheck = false
SET en = null
SET enSHOOT = null
SET minX, minY, minDist
```

2. Prosedur Run()

menjalankan keseluruhan strategi *greedy* untuk *bot* Suniper2

```
RUN METHOD:
    { Mengatur rotasi independen }
    SET rotasi independen

    { Inisialisasi posisi ancaman terdekat di tengah arena }
    SET minX = ArenaWidth / 2
    SET minY = ArenaHeight / 2
    SET minDist = DistanceTo(minX, minY)

    WHILE bot(IsRunning):
        { Memutar radar 360 derajat untuk mendeteksi ancaman }
    }
```

```

CALL SetTurnRadarRight(double.PositiveInfinity)

{ Jika pengecekan ancaman sudah pernah menemukan bot
}

IF startCheck == true:
    CALCULATE dist = DistanceTo(en.X, en.Y)
    IF dist < minDist OR EnemyCount == 1:
        SET minDist = dist
        SET minX = en.X
        SET minY = en.Y

{ Jika sedang menembak }
IF shoot == true:
    CALL TEMBAK()

{ Hitung jarak ke ancaman terdekat }
SET minDist = DistanceTo(minX, minY)

{ Jika ancaman sangat dekat (jarak < 200) }
IF minDist < 200:
    CALL SetTurnLeft(BearingTo(minX, minY))
    CALL SetBack(90)

{ Jika dekat dinding, ubah arah }
IF nearWall() == true:
    SET choice = -90
    IF Y < 150:
        IF BearingTo(X, 0) > 0:
            SET choice = 90
        ELSE IF Y > ArenaHeight - 150:
            IF BearingTo(X, ArenaHeight) > 0:
                SET choice = 90

    IF X < 150:
        IF BearingTo(0, Y) > 0:
            SET choice = 90
        ELSE IF X > ArenaWidth - 150:
            IF BearingTo(ArenaWidth, Y) > 0:
                SET choice = 90

    IF choice < 0 AND BearingTo(minX, minY) > 0:
        CALL SetTurnLeft(choice)
    ELSE IF choice > 0 AND BearingTo(minX, minY)
< 0:

```

```

CALL SetTurnLeft(choice)
CALL SetBack(60)

{ Jika tidak dikejar, lakukan gerakan maju mundur }
ELSE:
    INCREMENT maju
    IF ABS(maju) >= 75:
        CALL SetForward(0) { Diam }
    ELSE IF maju >= 50:
        CALL SetForward(70) { Maju }
    ELSE IF maju >= 25:
        CALL SetBack(0) { Diam }
    ELSE:
        CALL SetBack(70) { Mundur }
    IF maju == 100:
        SET maju = 0 {Reset counter}

    {Putar ke arah ancaman dengan sudut 90 derajat}
    CALL SetTurnLeft(BearingTo(minX, minY) - 90)

{Jika dekat dinding, stabilkan posisi}
IF runWall == true:
    CALL stabilize()

{Eksekusi semua perintah gerakan}
CALL Go()

```

3. Prosedur STABILIZE()

mengarahkan *bot* ke lokasi yang dianggap aman, yaitu berjarak minimal 200 dari dinding arena

```

STABILIZE METHOD:
    { Hitung posisi X baru agar stabil }
    SET newX2 = X
    IF X < 200:
        SET newX2 = 200
    ELSE IF X > ArenaWidth - 200:
        SET newX2 = ArenaWidth - 200

    { Hitung posisi Y baru agar stabil }
    SET newY2 = Y
    IF Y < 200:
        SET newY2 = 200
    ELSE IF Y > ArenaHeight - 200:

```



```

        SET newY2 = ArenaHeight - 200

        { Hitung sudut untuk stabilisasi }
        SET angleRun = BearingTo(newX2, newY2)
        WHILE angleRun > 180:
            SET angleRun -= 360
        WHILE angleRun < -180:
            SET angleRun += 360

        { Stabilkan posisi }
        CALL SetTurnLeft(angleRun)
        CALL SetForward(200)
        SET MaxSpeed = 10

        { Jika sudah stabil, ubah status }
        IF X >= 200 AND X <= ArenaWidth - 200 AND Y >= 200 AND Y
        <= ArenaHeight - 200:
            SET runWall = false
            SET MaxSpeed = 100

```

4. Function bool nearWall()

mengembalikan true apabila lokasi ini berjarak 150 dari dinding arena

```

NEARWALL METHOD:
    RETURN true jika bot dekat dinding

```

5. Prosedur tembak

mengatur logika untuk menembak musuh dengan memprediksi pergerakan musuh

```

TEMBAK METHOD:
    IF GunHeat == 0:
        CALCULATE dist = DistanceTo(enSHOOT.X, enSHOOT.Y)
        CALCULATE firePow = calcPower(dist)
        CALCULATE gunAngle = calcAngle(firePow)
        CALL SetTurnGunLeft(gunAngle)

        { Jika sudut tembak sudah mendekati target, tembak }
        IF ABS(GunTurnRemaining) < 2.5:
            CALL SetFire(firePow)
            SET shoot = false
        ELSE:
            SET shoot = false

```

6. Function double calcAngle(double firepower)

Menghitung besar rotasi yang perlu dilakukan gun dari bot

```

CALCANGLE METHOD:
    CALCULATE waktu = DistanceTo(enSHOOT.X, enSHOOT.Y) /
    CalcBulletSpeed(firepower)
    CALCULATE newX = enSHOOT.X + enSHOOT.Speed * waktu *
    COS(ToRadians(enSHOOT.Direction))
    CALCULATE newY = enSHOOT.Y + enSHOOT.Speed * waktu *
    SIN(ToRadians(enSHOOT.Direction))
    CALCULATE gunAngle = GunBearingTo(newX, newY)
    NORMALIZE gunAngle ke rentang -180 hingga 180
    RETURN gunAngle

```

7. Function calcpower

menghitung kekuatan peluru berdasarkan energi bot sekarang dan jarak terhadap bot yang akan ditembak

```

CALCPOWER METHOD:
    { Hitung kekuatan tembakan berdasarkan jarak dan energi }
    IF dist < 50:
        IF Energy > 30:
            RETURN 20
        ELSE IF Energy > 20:
            RETURN 5
        ELSE IF Energy > 10:
            RETURN 2
        ELSE:
            RETURN 1
    ELSE IF dist < 150:
        RETURN MIN(3, Energy / 7)
    ELSE IF dist < 250:
        RETURN MIN(3, Energy / 17)
    ELSE IF dist < 400:
        RETURN MIN(2, Energy / 40)
    ELSE:
        RETURN MIN(1, Energy / 45)

```

8. Prosedur onScannedBot(ScannedBotEvent e)

memberikan informasi mengenai status dan lokasi *bot* yang akan di-*scan* oleh radar

```

CALCPOWER METHOD:
    { Hitung kekuatan tembakan berdasarkan jarak dan energi }
    IF dist < 50:
        IF Energy > 30:
            RETURN 20

```

```

    ELSE IF Energy > 20:
        RETURN 5
    ELSE IF Energy > 10:
        RETURN 2
    ELSE:
        RETURN 1
ELSE IF dist < 150:
    RETURN MIN(3, Energy / 7)
ELSE IF dist < 250:
    RETURN MIN(3, Energy / 17)
ELSE IF dist < 400:
    RETURN MIN(2, Energy / 40)
ELSE:
    RETURN MIN(1, Energy / 45)

```

9. Prosedur onhitbot

menjadikan bot yang menabrak sebagai ancaman terdekat

```

ONHITBOT EVENT:
    SET minDist = DistanceTo(e.X, e.Y)
    SET minX = e.X
    SET minY = e.Y

```

10. Prosedur onhitwall

mengatur nilai runWall menjadi true jika *bot* menabrak dinding

```

ONHITWALL EVENT:
    SET runWall = true

```

4.1.2. NEARWALL

1. Inisialisasi variabel

- Bool clockwise: menyatakan true saat gerakan berupa clockwise.
- Int nearWallDistance: menyatakan jarak minimal dari dinding
- Double bear: digunakan untuk menyimpan angka yang dibutuhkan untuk memutar body

```

SET turnGun = false
SET clockwise = true
SET nearWallDistance = 50

```

2. Prosedur Run()

menjalankan keseluruhan strategi *greedy* untuk *bot* NEARWALL

```

RUN METHOD:

```

```

    { Memutar turret ke kiri 90 derajat }
    CALL TurnGunLeft(90)

    WHILE bot (isRunning):
        { Jika flag turnGun aktif, putar turret 180 derajat }
        IF turnGun == true:
            SET turnGun = false
            CALL SetTurnGunLeft(180)

        { Logika untuk menentukan arah dan pergerakan bot
        berdasarkan posisi di arena }
        { Bot berada di bagian bawah arena }
        IF Y < ArenaHeight / 2:

            { Bot berada di kiri bawah arena }
            IF X < ArenaWidth / 2:

                { Rotasi searah jarum jam }
                IF clockwise == true:

                    { Bergerak ke kiri atas }
                    CALCULATE bear = BearingTo(ArenaWidth -
nearWallDistance, nearWallDistance)
                    CALL SetTurnLeft(bear)
                    CALL SetForward(DistanceTo(ArenaWidth -
nearWallDistance, nearWallDistance))

                { Rotasi berlawanan arah jarum jam }
                ELSE:

                    { Bergerak ke kanan bawah }
                    CALCULATE bear =
BearingTo(nearWallDistance, ArenaHeight - nearWallDistance)
                    CALL SetTurnLeft(bear)
                    CALL
SetForward(DistanceTo(nearWallDistance, ArenaHeight -
nearWallDistance))

            { Bot berada di kanan bawah arena }
            ELSE:

                { Rotasi searah jarum jam }
                IF clockwise == true:

```

```

        { Bergerak ke kanan atas }
        CALCULATE bear = BearingTo(ArenaWidth -
nearWallDistance, ArenaHeight - nearWallDistance)
        CALL SetTurnLeft(bear)
        CALL SetForward(DistanceTo(ArenaWidth -
nearWallDistance, ArenaHeight - nearWallDistance))
        { Rotasi berlawanan arah jarum jam }
        ELSE:

        { Bergerak ke kiri bawah }
        CALCULATE bear =
BearingTo(nearWallDistance, nearWallDistance)
        CALL SetTurnLeft(bear)
        CALL
SetForward(DistanceTo(nearWallDistance, nearWallDistance))

        { Bot berada di bagian atas arena }
        ELSE:

        { Bot berada di kiri atas arena }
        IF X < ArenaWidth / 2:

        { Rotasi searah jarum jam }
        IF clockwise == true:

        { Bergerak ke kiri bawah }
        CALCULATE bear =
BearingTo(nearWallDistance, nearWallDistance)
        CALL SetTurnLeft(bear)
        CALL
SetForward(DistanceTo(nearWallDistance, nearWallDistance))

        { Rotasi berlawanan arah jarum jam }
        ELSE:

        { Bergerak ke kanan atas }
        CALCULATE bear = BearingTo(ArenaWidth -
nearWallDistance, ArenaHeight - nearWallDistance)
        CALL SetTurnLeft(bear)
        CALL SetForward(DistanceTo(ArenaWidth -
nearWallDistance, ArenaHeight - nearWallDistance))

        { Bot berada di kanan atas arena }
        ELSE:

```

```

        { Rotasi searah jarum jam }
        IF clockwise == true:

            { Bergerak ke kiri atas }
            CALCULATE bear =
            BearingTo(nearWallDistance, ArenaHeight - nearWallDistance)
            CALL SetTurnLeft(bear)
            CALL
            SetForward(DistanceTo(nearWallDistance, ArenaHeight -
            nearWallDistance))

            { Rotasi berlawanan arah jarum jam }
            ELSE:

                { Bergerak ke kanan bawah }
                CALCULATE bear = BearingTo(ArenaWidth -
            nearWallDistance, nearWallDistance)
                CALL SetTurnLeft(bear)
                CALL SetForward(DistanceTo(ArenaWidth -
            nearWallDistance, nearWallDistance))

            { Eksekusi semua perintah gerakan }
            CALL Go()

```

3. Prosedur OnScannedBot(ScannedBotEvent e)
menembakkan peluru dengan kekuatan 5 apabila menemukan musuh

```

ONSCANNEDBOT EVENT:
    { Menembak bot yang terdeteksi dengan kekuatan 5 }
    CALL Fire(5)

```

4. Prosedur OnHitBot(HitBotEvent e)
memutar arah perputaran *boti* dengan mengganti nilai variabel clockwise

```

ONHITBOT EVENT:
    { Mengubah arah rotasi }
    SET clockwise = not clockwise

    { Mengubah flag turnGun }
    SET turnGun = not turnGun

```

4.1.3. CircleImpact

1. Inisialisasi variabel

- Bool followingEnemy: Bernilai true apabila *bot* sedang mengikuti musuh
- Bool startingCircle: Bernilai true apabila *bot* akan mulai berpatroli dalam lintasan lingkaran
- Bool goingClockwise: Bernilai true apabila *bot* berpatroli searah jarum jam
- Int turnToShoot: Bernilai banyak *turns* bagi *bot* untuk menyerang satu target sebelum memperhitungkan target terbaik berikutnya
- Int firstHundredTurns: Bernilai banyak *turns* bagi *bot* sebelum keluar dari mode *pacifist*
- Int bouncingBackTurn: Bernilai banyak *turns* bagi *bot* untuk kembali setelah menabrak objek, baik *bot* lain maupun dinding
- Double centerX: menyimpan koordinat X dari pusat arena
- Double centerY: menyimpan koordinat Y dari pusat arena

```
SET followingEnemy = false
SET startingCircle = true
SET goingClockwise = true
SET turnToShoot = 20
SET firstHundredTurns
SET bouncingBackTurn = 0
SET centerX = ArenaWidth / 2
SET centerY = ArenaHeight / 2
```

2. Prosedur Run()

menjalankan keseluruhan strategi *greedy* untuk *bot* CircleImpact

```
RUN METHOD:
{ Menentukan mode pacifist }
{ jika terdapat kurang dari lima musuh }
IF EnemyCount < 5:
    SET firstHundredTurns = 200 { Mode pacifist selesai
    setelah 200 turn }

    { jika terdapat lebih dari lima musuh }
ELSE:
    SET firstHundredTurns = -1 { Mode pacifist selesai
    setelah musuh tinggal 5 }

{ Inisialisasi radius untuk patroli melingkar }
SET radius = 200

WHILE bot masih berjalan (IsRunning):

    { Hitung jarak bot ke pusat arena }
```

```

        CALCULATE distance = Math.Sqrt((X - centerX)^2 + (Y -
centerY)^2)

        { Jika bot sedang retreat setelah tabrakan }
        IF bouncingBackTurn > 0:
            DECREMENT bouncingBackTurn

        ELSE:

            { Jika bot tidak sedang mengikuti musuh }
            IF not followingEnemy:

                { Hitung sudut ke pusat arena }
                CALCULATE angleToCenter = BearingTo(centerX,
centerY)

                { Jika bot berada di luar radius tertentu,
kembali ke dalam radius }
                IF distance > radius:
                    IF angleToCenter not = 0:
                        CALL SetTurnLeft(angleToCenter)
                    CALL SetForward(20)
                    SET startingCircle = true

                { Jika bot berada dalam radius, lakukan
patroli melingkar }
                ELSE:

                    IF goingClockwise:

                        { belok untuk mulai melingkar }
                        IF startingCircle:
                            CALL SetTurnLeft(90)
                            SET startingCircle = false

                        { mulai melingkar }
                        ELSE IF TurnRemaining == 0:
                            CALL SetTurnRight(45)
                            CALL SetForward(distance)
                        ELSE:

                            { belok untuk mulai melingkar }
                            IF startingCircle:
                                CALL SetTurnRight(90)

```



```

        SET startingCircle = false
        CALL SetForward(distance)

        { mulai melingkar }
        ELSE IF TurnRemaining == 0:
            CALL SetTurnLeft(45)
            CALL SetForward(distance)

        { Jika bot sedang mengikuti musuh }
        ELSE:

            { Jika bot tidak di mode pacifist }
            IF Rounds < 5 AND firstHundredTurns <= 0:
                CALL SetTurnLeft(60)
                CALL SetForward(10)

            { Jika turnToShoot habis, berhenti
mengikuti musuh }
            IF turnToShoot < 0:
                CALL ClearEvents()
                SET followingEnemy = false
                SET turnToShoot = 20

            { Lakukan scanning radar setelah mode pacifist
selesai }
            IF Rounds < 5 AND firstHundredTurns <= 0:
                CALL SetTurnRadarLeft(60)

        { Eksekusi semua perintah gerakan }
        CALL Go()

```

3. Prosedur OnScannedBot(ScannedBotEvent e)

melakukan penembakan serangan kepada musuh berdasarkan kondisi-kondisi yang harus dipenuhi

```

ONSCANNEDBOT EVENT:
    { Hitung jarak ke musuh yang terdeteksi }
    CALCULATE distance = DistanceTo(e.X, e.Y)

    { Jika bot tidak sedang mengikuti musuh }
    IF not followingEnemy:

        { Abaikan musuh jika jaraknya terlalu jauh }
        IF distance <= 300:

```

```

        { Mulai mengikuti musuh jika mode pacifist telah
selesai }
        IF Rounds < 5 AND firstHundredTurns <= 0:
            CALL ClearEvents()
            SET followingEnemy = true

        { Jika bot sedang mengikuti musuh }
        IF followingEnemy:
            CALCULATE bearingFromGun = GunBearingTo(e.X, e.Y)
            CALL SetTurnGunLeft(bearingFromGun)

            { Tembak musuh dengan kekuatan berdasarkan energi bot
        }

        IF Energy >= 30:
            CALL SetFire(2.5)
        ELSE:
            CALL SetFire(1.5)

```

4. Prosedur OnTick(TickEvent tickEvent)

melakukan perhitungan terhadap jumlah *turn* yang digunakan sebagai dasar pengambilan tindakan

```

ONTICK EVENT:
    { Hitung jumlah turn yang telah dilewati }
    IF bouncingBackTurn > 0:
        DECREMENT bouncingBackTurn
    ELSE:
        IF firstHundredTurns > 0:
            DECREMENT firstHundredTurns
        IF followingEnemy:
            DECREMENT turnToShoot

```

5. Prosedur OnHitBot(HitBotEvent e)

melakukan *retreat* dengan berputar balik jika *bot* mengenai *bot* musuh

```

ONHITBOT EVENT:
    { Retreat selama 10 turn }
    SET bouncingBackTurn = 10

    { Ubah arah rotasi }
    IF goingClockwise:
        SET goingClockwise = false
    ELSE:

```

```

    SET goingClockwise = true

    { Putar 180 derajat dan bergerak mundur }
    CALL TurnLeft(180)
    CALL Forward(100)

```

6. Prosedur OnHitWall(HitWallEvent e)

melakukan *retreat* dengan pergi dengan arah ke pusat arena jika *bot* mengenai dinding

```

ONHITWALL EVENT:
    { Retreat selama 10 turn }
    SET bouncingBackTurn = 10

    { Putar ke arah pusat arena dan bergerak mundur }
    CALCULATE bearing = BearingTo(centerX, centerY)
    CALL TurnLeft(bearing)
    CALL Forward(100)

```

4.1.4. TRIPIN

1. Inialisasi variabel

- Int county: berfungsi untuk mengatur pembagian posisi dalam pembentukan pola. Dibantu dengan menggunakan modulo dan pembagian integer.
- Bool RESETTING: bernilai true jika *bot* diperlukan untuk melakukan pemosisian diri ulang ke tengah arena

```

SET county = 0
SET RESETTING = true

```

2. Prosedur Run()

menjalankan keseluruhan strategi *greedy* untuk *bot* TRIPIN

```

RUN METHOD:

    WHILE bot (IsRunning):

        { Jika bot sedang mereset posisinya }
        IF RESETTING == true:

            { Putar bot ke arah tengah arena }
            CALL TurnLeft(BearingTo(ArenaWidth / 2,

```

```

ArenaHeight / 2))
    { Bergerak maju ke tengah arena }
    CALL Forward(DistanceTo(ArenaWidth / 2,
ArenaHeight / 2))
    { Reset selesai }
    SET RESETTING = false

    { Jika tidak sedang mereset }
ELSE:

    { Menambah nilai counter untuk pola gerakan }
    INCREMENT county

    { Pola gerakan berdasarkan nilai counter }
    IF (county / 6) % 16 <= 4:

        { Bergerak lurus ke depan }
        CALL SetTurnLeft(0)
        CALL SetForward(20)

    { Bergerak maju sedikit dan berputar ke kiri }
ELSE:
    CALL SetForward(5)
    CALL SetTurnLeft(30)

    { Eksekusi semua perintah gerakan }
    CALL Go()

```

3. Prosedur OnScannedBot(ScannedBotEvent e)

menembakkan peluru dengan kekuatan 5 apabila menemukan musuh, dan menembakkan dengan kekuatan tinggi apabila musuh sangat dekat

```

ONSCANNEDBOT EVENT:

    { Menembak bot yang terdeteksi dengan kekuatan 5 }
    CALL SetFire(5)

    { Jika jarak sangat dekat, gunakan kekuatan tembakan yang
lebih tinggi }

```

```

CALCULATE dis = DistanceTo(e.X, e.Y)
IF dis < 25.0 AND Energy > 60:
    CALL SetFire(50) { Tembakan dengan kekuatan 50 }
ELSE IF dis < 25.0 AND Energy > 40:
    CALL SetFire(30) { Tembakan dengan kekuatan 30 }
ELSE IF dis < 25.0 AND Energy > 20:
    CALL SetFire(10) { Tembakan dengan kekuatan 10 }

```

4. Prosedur OnHitWall(HitWallEvent e)

membuat RESETING bernilai true agar *bot* dapat kembali ke tengah arena

```

ONHITWALL EVENT:
{ Bersihkan semua event yang sedang diproses }
CALL ClearEvents()
{ Aktifkan mode reset untuk kembali ke tengah arena }
SET RESETING = true

```

4.2. Analisis dan Pengujian

Analisis dan pengujian dilakukan dengan cara menjalankan permainan Robocode untuk melihat hasil akhir yang didapatkan oleh setiap *bot*. Permainan dilakukan dengan memasukkan lima *bot* ke dalam arena, terdiri atas salah satu *bot* aplikasi strategi *greedy* yang sedang diuji dan empat *template bot* lainnya. Empat *template bot* tersebut adalah Spin Bot, Crazy, Track Fire, dan Ram Fire, dengan hasil analisis untuk setiap *bot* adalah sebagai berikut.

4.2.1. Suniper2

Results for 25 rounds											
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	sucipto 1.0	6951	3700	560	2333	176	178	4	13	5	3
2	Spin Bot 1.0	5977	2650	160	2560	172	413	22	5	10	3
3	Walls 1.0	4896	2350	200	2130	111	104	0	5	4	6
4	Crazy 1.0	3361	2450	40	756	15	95	4	1	5	10
5	Ram Fire 1.0	2579	1350	40	766	15	371	36	1	1	3

Results for 25 rounds											
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Walls 1.0	5871	3150	240	2230	140	110	0	8	7	5
2	Spin Bot 1.0	5366	2550	40	2272	120	371	13	3	8	7
3	sucipto 1.0	5132	2350	240	1966	115	419	42	6	4	4
4	Crazy 1.0	4758	3400	440	736	27	155	0	7	5	5
5	Ram Fire 1.0	2228	1050	40	720	9	407	2	1	1	4

Results for 25 rounds											
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Spin Bot 1.0	6601	3000	240	2704	195	415	46	6	10	5
2	sucipto 1.0	6259	3150	480	2244	176	193	15	12	2	3
3	Walls 1.0	4807	2550	160	1830	109	144	14	4	7	5
4	Crazy 1.0	3907	2900	120	764	14	109	0	3	4	9
5	Ram Fire 1.0	1945	900	0	630	34	370	10	0	2	3

Gambar 10. Nilai akhir pengujian *bot* Suniper2

Dari tiga permainan yang diujikan, *bot* Suniper2 berhasil meraih kemenangan untuk salah satu permainan tersebut. Kedua permainan lainnya juga memberikan hasil akhir yang dengan perbedaan relatif kecil terhadap pemenang permainan. Upaya *bot*

dalam menjaga jarak dari *bot* musuh memberikan keunggulan dalam mengurangi kemungkinan *bot* untuk hancur pada awal *turn*. Selain itu, strategi yang digunakan dalam melakukan penembakan juga memberikan akurasi yang lebih tinggi. Diikuti dengan dasar heuristik *dynamic movement* dan *staying alive*, *bot* Suniper2 memiliki daur hidup yang relatif lebih lama dibandingkan *bot* lainnya, ditandai dengan skor *survival* yang cukup tinggi untuk setiap permainan.

4.2.2. NearWall

Results for 25 rounds											
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Spin Bot 1.0	7323	3400	360	2800	272	460	31	10	6	8
2	Walls 1.0	4937	2600	200	1860	117	154	6	7	4	4
3	Sucipto 1.0	4589	2300	160	1904	70	149	5	4	7	2
4	Crazy 1.0	3783	2700	240	668	28	143	4	4	2	9
5	Ram Fire 1.0	2664	1300	40	731	0	480	114	0	6	2

Results for 25 rounds											
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Spin Bot 1.0	6690	2800	280	2768	248	559	34	11	5	3
2	Walls 1.0	5299	3100	240	1740	83	136	0	5	9	6
3	Sucipto 1.0	5188	2950	320	1664	68	180	6	7	4	6
4	Crazy 1.0	3403	2400	120	704	2	175	1	1	3	8
5	Ram Fire 1.0	2748	1250	40	874	14	521	49	1	4	2

Results for 25 rounds											
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Spin Bot 1.0	6312	2600	240	2672	271	500	29	10	5	1
2	Walls 1.0	5111	2800	160	1880	88	170	12	5	7	4
3	Sucipto 1.0	4866	2550	240	1760	63	238	15	6	3	6
4	Crazy 1.0	4654	3400	360	712	16	163	2	4	8	8
5	Ram Fire 1.0	2664	1150	0	828	15	564	107	0	2	6

Gambar 11. Nilai akhir pengujian *bot* NearWall

Bot NearWall memberikan hasil yang baik, khususnya ketika dihadapkan dengan *bot* Ram Fire. Langkah *bot* NearWall untuk mendekatkan diri ke dinding mengurangi kemungkinan bagi *bot* Ram Fire untuk menghantamkannya ke *bot* NearWall. Hal ini merupakan hasil yang diinginkan dari dasar heuristik *staying alive*, yaitu menghindari kemungkinan terjadinya konflik sebesar mungkin.

4.2.3. CircleImpact

Results for 25 rounds											
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Spin Bot 1.0	7716	3450	440	3008	270	523	24	13	3	5
2	Walls 1.0	5108	2750	200	1910	114	124	10	5	9	2
3	Crazy 1.0	4267	3200	280	648	20	119	0	5	8	4
4	sucipto 1.0	3483	2150	80	1095	53	104	0	2	4	6
5	Ram Fire 1.0	2818	950	0	1099	10	704	54	0	1	8

Results for 25 rounds											
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Walls 1.0	7106	4000	520	2280	165	125	15	12	10	0
2	Spin Bot 1.0	6410	2800	280	2512	226	582	10	8	7	2
3	Crazy 1.0	3866	2800	80	760	20	197	8	2	1	15
4	Ram Fire 1.0	3107	1100	0	1152	17	684	153	0	5	6
5	sucipto 1.0	3027	1800	120	947	53	107	0	3	2	2

Results for 25 rounds											
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Spin Bot 1.0	7233	3300	320	2896	191	467	58	11	8	5
2	Walls 1.0	5751	3100	200	2220	114	116	0	5	9	7
3	Crazy 1.0	4523	3250	320	768	34	146	4	6	6	7
4	sucipto 1.0	3145	1950	80	981	35	98	1	1	2	6
5	Ram Fire 1.0	2531	900	80	930	23	559	38	2	0	0

Gambar 12. Nilai akhir pengujian *bot* CircleImpact

Performa dari *bot* CircleImpact relatif lebih rendah dibandingkan dengan *template bot* lainnya. Hal ini kemungkinan besar disebabkan oleh dasar heuristik kebanyakan dari *template bot*, seperti Spin Bot dan Crazy, yang juga menggunakan dasar heuristik *dynamic movement* sehingga peluru yang dilemparkan oleh CircleImpact banyak yang meleset dan menghabiskan energi dengan cuma-cuma.

4.2.4. TRIPIN

Results for 25 rounds												
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds	
1	Spin Bot 1.0	5982	2900	280	2192	155	428	26	9	5	2	
2	Walls 1.0	5350	3050	320	1730	130	120	0	9	3	6	
3	Sucipto 1.0	5138	2150	80	2160	126	595	26	3	9	5	
4	Crazy 1.0	4438	3200	280	800	14	143	0	4	6	8	
5	Ram Fire 1.0	2898	1200	40	978	19	587	73	0	2	4	

Results for 25 rounds												
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds	
1	Spin Bot 1.0	6966	3350	360	2624	194	416	21	12	5	3	
2	Walls 1.0	5041	2900	280	1650	95	115	0	6	5	4	
3	Crazy 1.0	4573	3350	320	688	31	184	0	6	7	3	
4	Sucipto 1.0	3917	1750	40	1584	29	508	6	1	4	7	
5	Ram Fire 1.0	3162	1150	0	1100	40	694	178	0	4	8	

Results for 25 rounds												
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds	
1	Spin Bot 1.0	6851	3250	280	2640	211	402	68	8	9	5	
2	Walls 1.0	5666	3100	360	1970	109	126	0	10	4	3	
3	Sucipto 1.0	4601	2000	80	1920	71	494	34	2	8	6	
4	Crazy 1.0	4449	3200	280	740	29	199	0	5	4	9	
5	Ram Fire 1.0	2277	950	0	792	0	485	50	0	0	2	

Gambar 13. Nilai akhir pengujian *bot* TRIPIN

Hal yang serupa dengan hasil akhir dari *bot* CircleImpact juga ditunjukkan dalam hasil akhir *bot* TRIPIN, dimana dasar heuristik *dynamic movement* yang dimiliki oleh *template bot* lainnya turut memengaruhi performa *bot* TRIPIN dalam taraf yang cukup signifikan. Dapat disimpulkan bahwa *bot* dengan dasar heuristik *dynamic movement* akan lebih mendominasi permainan dengan mayoritas *bot* berfokus pada tindakan *offensive* dibandingkan *defensive*

4.2.5. Free for All

Results for 100 rounds												
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds	
1	suniper2 1.0	9849	4900	780	3283	383	478	24	27	6	7	
2	NearWall 1.0	7455	3750	240	3056	224	174	10	7	21	13	
3	sucipto 1.0	6346	3700	330	1962	190	156	8	10	13	15	
4	TRIPIN 1.0	4417	1500	30	1984	45	810	47	3	7	12	

Results for 100 rounds												
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds	
1	suniper2 1.0	10381	5650	900	3118	324	352	37	27	13	4	
2	NearWall 1.0	9290	4500	270	4080	334	106	0	13	24	8	
3	TRIPIN 1.0	4731	1800	120	2048	125	612	26	4	6	18	
4	sucipto 1.0	3600	2100	90	1199	65	134	10	3	4	17	

Results for 100 rounds												
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet Dmg.	Bullet Bonus	Ram Dmg.	Ram Bonus	1sts	2nds	3rds	
1	NearWall 1.0	9823	4750	580	3968	402	122	0	21	13	11	
2	suniper2 1.0	7014	3200	390	2597	185	604	38	12	11	11	
3	TRIPIN 1.0	6712	2950	260	2480	170	793	58	9	15	9	
4	sucipto 1.0	5131	3400	310	1177	76	158	9	7	10	18	

Gambar 14. Nilai akhir pengujian seluruh *bot* sucipto

Pada pengujian *bot* sucipto secara keseluruhan, dilakukan 3 kali pertandingan yang terdiri atas 100 *rounds* dan banyak *turn* maksimum adalah 50000. Pada gambar di atas, *bot* CircleImpact menggunakan alias sucipto.

Hasil akhir memperlihatkan *bot* Suniper2 memiliki rata-rata skor akhir paling tinggi. Sesuai dengan hasil analisis sebelumnya, daur hidup *bot* yang dapat diandalkan karena dasar heuristik *staying alive* dan *dynamic movement* sangat berdampak dalam memberikan skor tertinggi.

Namun, hasil akhir *bot* TRIPIN dan *bot* CircleImpact kesulitan untuk mendapatkan nilai yang tinggi. Hal ini terutama diakibatkan oleh dasar heuristik *bot* yang kurang mengutamakan poin dari bertindak secara *offensive*. Penembakan yang *bot* lakukan tidak mengutamakan akurasi banyak peluru dan energi yang terbuang sia-sia. Memang sejatinya, *offence is the best defence*.

Dari hasil analisis di atas, informasi hasil analisis tingkat keoptimalan dari setiap strategi *greedy* yang digunakan dapat ditabulasikan ke dalam tabel berikut.

Tabel 1. Hasil analisis pengujian setiap algoritma *greedy*

No.	Nama Bot	Optimal		Alasan kurang optimal
		Tingkat Hidup	Akurasi	
1.	Suniper2	✓	✓	<i>Bot</i> ini bekerja sesuai dengan strategi yang telah dibuat.
2.	TRIPIN	✗	✗	<i>Bot</i> ini memiliki kelemahan saat terdapat banyak <i>bot</i> , sehingga bisa terkena peluru dari berbagai arah. Hal ini karena <i>bot</i> ini cenderung memposisikan dirinya mendekati arena bagian tengah.
3.	NearWall	✓	✓	<i>Bot</i> ini bekerja sesuai dengan strategi yang telah dibuat.
4.	CircleImpact	✗	✗	<i>Bot</i> ini memiliki kelemahan saat melawan musuh yang memiliki gerakan dinamis dan sulit ditebak. Selain itu, pergerakan dari <i>bot</i> ini juga cenderung di tengah arena sehingga rentan terkena peluru dari <i>bot</i> musuh

BAB 5

KESIMPULAN DAN SARAN

1. Kesimpulan

Melalui tugas besar 1 mata kuliah IF2211 Strategi Algoritma, kelompok mendapatkan kesempatan untuk memahami lebih dalam mengenai algoritma *greedy*. Kelompok melakukan eksplorasi terhadap dasar-dasar heuristik yang dapat digunakan dalam permainan Robocode untuk membentuk strategi *greedy* dan diimplementasikan ke dalam *bot*, dengan harapan algoritma dan alur logika berbasis strategi *greedy* tersebut dapat membawa *bot* menjadi pemenang dalam suatu permainan Robocode.

Penyelesaian tugas besar diawali dengan pengenalan terhadap *environment* sistem yang digunakan dalam Robocode, kemudian merumuskan strategi *greedy* dan melakukan pengujian dengan melangsungkan permainan secara iteratif untuk menerima *feedback* mengenai kelebihan dan kekurangan strategi. Segala bentuk rangkaian pekerjaan tugas besar, meliputi teori yang digunakan, hasil pembentukan *source code*, serta aplikasi dari implementasi strategi *greedy* dalam *bot* telah kelompok dokumentasikan secara terstruktur melalui laporan ini. Walaupun hasil akhir yang ditunjukkan oleh setiap strategi masih jauh dari kata sempurna, dengan berbangga hati kelompok telah berhasil membentuk empat buah program berdasarkan strategi *greedy* yang diimplementasikan ke dalam *bot* di permainan Robocode sebagai hasil akhir dari tugas besar 1 mata kuliah IF2211 Strategi Algoritma.

2. Saran

Selama melalui tahapan pengerjaan, kelompok mendapati beberapa permasalahan yang menjadi hambatan dalam menyelesaikan tugas. Menanggapi permasalahan tersebut, terdapat beberapa saran yang telah menjadi evaluasi bagi setiap anggota kelompok dalam menyelesaikan tugas-tugas berikutnya, di antaranya adalah:

- Membentuk timeline berisi rancangan kasar alur dan target penyelesaian tugas besar secara bertahap setelah spek tugas diturunkan. Membentuk timeline akan membantu untuk melacak progres pengerjaan tugas sehingga setiap anggota kelompok memiliki

tingkat kesadaran yang sama dan tidak kewalahan ketika deadline pengumpulan tugas sudah dekat;

- Melakukan pembagian tugas secara lebih efisien dan tertata, sehingga anggota kelompok dapat mengerjakan bagiannya masing-masing dengan baik tanpa menunggu koordinasi terlebih dahulu;

LAMPIRAN

1. Tautan *repository*

<https://www.youtube.com/watch?v=9utL7m8S6YQ>

2. Tautan video

https://github.com/mimiCrai/Tubes1_Sucipto.git

3. Checklist

Tabel 2. Checklist

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

DAFTAR PUSTAKA

Informatika.stei.itb.ac.id. (2025). Algoritma Greedy (Bagian 1). Diakses pada 17 Maret 2025, dari

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/04-Algoritma-Greedy-\(2025\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/04-Algoritma-Greedy-(2025)-Bag1.pdf)

Informatika.stei.itb.ac.id. (2025). Algoritma Greedy (Bagian 2). Diakses pada 17 Maret 2025, dari

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/05-Algoritma-Greedy-\(2025\)-Bag2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/05-Algoritma-Greedy-(2025)-Bag2.pdf)

Informatika.stei.itb.ac.id. (2025). Algoritma Greedy (Bagian 3). Diakses pada 17 Maret 2025, dari

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/06-Algoritma-Greedy-\(2025\)-Bag3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/06-Algoritma-Greedy-(2025)-Bag3.pdf)