

Laporan Tugas Besar 1

Robocode Tank Royale Bot

Disusun untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma pada Semester 2
(Genap) Tahun Akademik 2024/2025



Kelompok 21 (Rudal Sekeloa Reloaded)

Razi Rachman Widyadhana 13523004

Nayaka Ghana Subrata 13523090

Muhammad Adha Ridwan 13523098

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132
2025**

Daftar Isi

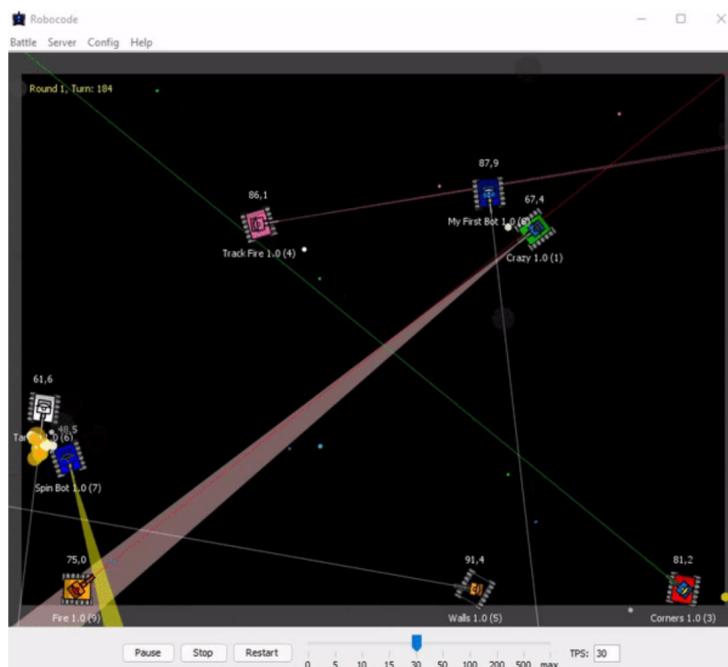
Bab I Deskripsi Masalah	1
Bab II Landasan Teori	6
2.1. Algoritma <i>Greedy</i>	6
2.2. Bot	7
2.2.1 Implementasi	7
2.2.2 <i>Usage</i>	8
2.3. <i>Game Engine</i>	9
Bab III Aplikasi Strategi <i>Greedy</i>	12
3.1. Abstraksi Permainan	12
3.2. Eksplorasi Alternatif Solusi <i>Greedy</i>	13
3.2.1 Kaze (<i>Energy Wise</i>)	13
3.2.2 Sweepredict (Scan All & Predict)	14
3.2.3 Waves	15
3.2.4 RudalSekeloa (Scan & Track + Stacking Heuristics)	16
3.3. Pemilihan Strategi <i>Greedy</i> Utama	17
Bab IV Implementasi dan Pengujian	18
4.1. Implementasi Algoritma Greedy pada Bot	18
4.2. Implementasi Solusi <i>Greedy</i> yang dipilih	25
4.3. Pengujian	33
4.4. Analisis dan Pembahasan	36
Bab V Penutup	37
5.1. Kesimpulan	37
5.2. Saran	37
5.3. Refleksi	38
Lampiran	39
6.1. Tautan	39
6.2. Tabel Spesifikasi	39
Referensi	39

Bab I

Deskripsi Masalah

Robocode adalah permainan pemrograman yang bertujuan untuk membuat kode bot dalam bentuk tank virtual untuk berkompetisi melawan bot lain di arena. Pertempuran Robocode berlangsung hingga bot-bot bertarung hanya tersisa satu seperti permainan Battle Royale, karena itulah permainan ini dinamakan Tank Royale. Nama Robocode adalah singkatan dari “Robot code”, yang berasal dari versi asli/pertama permainan ini.

Robocode Tank Royale adalah evolusi/versi berikutnya dari permainan ini, di mana bot dapat berpartisipasi melalui Internet/jaringan. Dalam permainan ini, pemain berperan sebagai programmer bot dan tidak memiliki kendali langsung atas permainan. Pemain hanya bertugas untuk membuat program yang menentukan logika atau “otak” bot. Program yang dibuat akan berisi instruksi tentang cara bot bergerak, mendeteksi bot lawan, menembakkan senjatanya, serta bagaimana bot bereaksi terhadap berbagai kejadian selama pertempuran.



Gambar 1: Robocode Tank Royale

Komponen penting dari permainan IQ Puzzler Pro terdiri dari:

1. *Rounds* dan *Turns*

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

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

Pada setiap turn, sebuah bot dapat:

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

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

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

2. Batas Waktu Giliran

Penting untuk dicatat bahwa setiap bot memiliki batas waktu untuk setiap *turn* yang disebut *turn timeout*, biasanya antara 30-50 ms (dapat diatur sebagai aturan pertempuran). Ini berarti bahwa bot tidak bisa mengambil waktu sebanyak yang mereka inginkan untuk bergerak dan menyelesaikan *turn* saat ini.

Setiap kali *turn* baru dimulai, penghitung waktu ulang diatur ulang dan mulai berjalan. Jika batas waktu tercapai dan bot tidak mengirimkan pergerakannya untuk *turn* tersebut, maka tidak ada perintah yang dikirim ke server. Akibatnya, bot akan melewatkannya. Jika bot melewatkannya, 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

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

Seperti disebutkan sebelumnya, peluru yang lebih berat akan bergerak lebih lambat. Ini berarti akan membutuhkan waktu lebih lama untuk mencapai target,

meningkatkan risiko peluru tidak mengenai sasaran. Sebaliknya, peluru yang lebih ringan bergerak lebih cepat sehingga lebih mudah mengenai target, tetapi peluru ringan tidak memberikan banyak poin energi saat mengenai bot musuh.

5. Panas Meriam (Gun Heat)

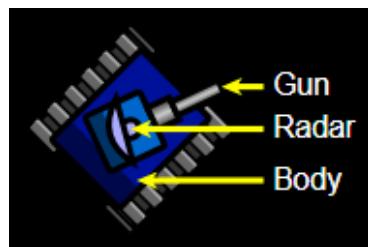
Saat menembakkan peluru, meriam akan menjadi panas. Peluru yang lebih berat menghasilkan lebih banyak panas dibandingkan peluru yang lebih ringan. Ketika meriam terlalu panas, bot tidak dapat menembak hingga suhu meriam turun ke nol. Selain itu, meriam juga sudah dalam keadaan panas di awal round dan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

6. Tabrakan

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

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

7. Bagian Tubuh Tank



Gambar 2: Bagian-bagian Tubuh Tank

Tubuh tank terdiri dari 3 bagian:

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

8. Pergerakan

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

9. Berbelok

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

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

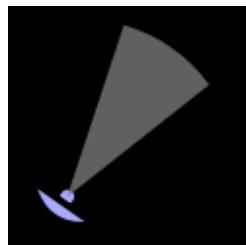
Bagian paling lambat adalah tubuh tank, yang dalam kondisi terbaik dapat berputar hingga 10 derajat per giliran. Namun, ini bergantung pada kecepatan bot saat ini. Semakin cepat bot bergerak, semakin lambat kemampuannya untuk berbelok.

Perlu diperhatikan bahwa tidak ada energi yang dikonsumsi saat bot bergerak atau berbelok.

10. Pemindaian

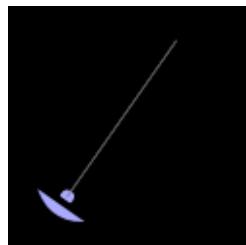
Aspek penting dalam Robocode adalah memindai bot musuh menggunakan radar. Radar dapat mendeteksi bot dalam jangkauan hingga 1200 piksel. Musuh yang berada lebih dari 1200 piksel dari bot tidak dapat terdeteksi atau dipindai oleh radar.

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



Gambar 3: Kondisi radar bergerak

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



Gambar 4: Kondisi radar tidak bergerak

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 ronde tersebut mendapatkan **50 poin**.
- **Last Survival Bonus:** Bot terakhir yang bertahan pada suatu ronde akan mendapatkan **10 poin** dikali dengan banyaknya musuh.
- **Ram Damage:** Bot mendapatkan **poin sebesar 2 kali *damage*** yang dibuat kepada bot musuh dengan cara menabrak.
- **Ram Damage Bonus:** Apabila musuh terbunuh dengan cara ditabrak, bot mendapatkan **poin sebesar 30% dari *damage*** yang dibuat kepada musuh yang terbunuh.

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

Pada Tugas Besar 1 IF2211 Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain dengan menggunakan strategi *greedy* masing-masing.

Bab II

Landasan Teori

2.1. Algoritma *Greedy*

Algoritma *greedy* adalah algoritma yang memecahkan persoalan secara langkah per langkah (*step-by-step*) sedemikian sehingga pada setiap langkahnya hanya mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan dan tanpa bisa kembali lagi ke langkah sebelumnya dengan “harapan” bahwa apabila memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global (Rinaldi, 2025).

Meskipun demikian, keputusan yang diambil mungkin hanya menghasilkan solusi *sub-optimum* atau *pseudo-optimum*. Alasannya adalah algoritma *greedy* tidak mengevaluasi secara menyeluruh semua kemungkinan solusi yang ada dan terdapat berbagai fungsi seleksi yang berbeda sehingga pemilihan fungsi yang tepat menjadi kunci dalam mencapai solusi optimal. Dengan demikian, pada beberapa kasus, algoritma *greedy* tidak selalu dapat memberikan solusi yang optimal, namun hanya solusi *sub-optimum*.

Jika kebutuhan akan solusi paling optimal tidak terlalu mendesak, algoritma *greedy* dapat menjadi pilihan untuk menghasilkan solusi hampiran. Hal ini lebih disukai daripada menggunakan algoritma yang membutuhkan waktu komputasi eksponensial untuk mencapai solusi yang eksak.

Algoritma *greedy* terdiri dari beberapa elemen, yakni:

a. **Himpunan Kandidat (C)**

Himpunan yang berisi kandidat yang akan dipilih pada setiap langkah. Misalnya graf, job, task, koin, benda, karakter, dan sebagainya.

b. **Himpunan Solusi (S)**

Himpunan yang berisi kandidat yang sudah dipilih.

c. **Fungsi Solusi**

Fungsi yang menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.

d. **Fungsi Seleksi (*Selection Function*)**

Fungsi yang memilih kandidat berdasarkan strategi tertentu. Strategi *greedy* ini bersifat heuristik.

e. **Fungsi Kelayakan (*Feasible*)**

Fungsi ini memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi dengan menguji kelayakannya.

f. **Fungsi Obyektif**

Fungsi yang menunjukkan sifat memaksimumkan atau meminimumkan.

2.2. Bot

Bot adalah program atau perangkat lunak yang bekerja secara otomatis untuk melakukan suatu tugas berulang. Bot mengikuti suatu instruksi pada kode yang telah diprogram dan menjalankannya terus menerus sampai ada perintah untuk berhenti atau suatu parameter berhenti yang lain. Setelah diaktifkan, bot dapat berinteraksi satu sama lain ataupun dengan manusia.

2.2.1 Implementasi

Pada tugas ini, bot mengikuti sebuah aturan dan template dari *starter-kit* berikut yang diberikan oleh asisten. Suatu bot didefinisikan dalam sebuah *folder* dengan struktur,

```
TemplateBot/  
|- TemplateBot.cmd  
|- TemplateBot.cs  
|- TemplateBot.csproj  
|- TemplateBot.json  
|- TemplateBot.sh
```

Terdapat 5 *file* yang ada dalam *folder* berikut dengan penjelasan:

(a) .cmd

Digunakan untuk mengkompilasi program pada sistem operasi Windows melalui *command*

```
.\TemplateBot.cmd
```

(b) .cs

File C# yang digunakan untuk mengimplementasi algoritma *Greedy* pada bot.

(c) .csproj

Digunakan untuk melakukan *binding* hasil *build* dari bot ke *Game Engine*.

(d) .json

Berisi metadata dari bot yang diimplementasikan, seperti `name`, `version`, `author`, `description`, dan lainnya.

(e) .sh

Digunakan untuk mengkompilasi program pada sistem operasi berbasis UNIX, seperti Linux dan macOS melalui *command*

```
./TemplateBot.sh
```

Untuk proses *development*, algoritma bot, strategi *greedy* dan API *usage* diimplementasikan pada *file .cs* pada *folder* bot yang bersangkutan. Selain itu, kita juga harus mengimplementasikan *.csproj* dan *.json* untuk melakukan integrasi bot dengan *game engine*. Untuk template dasar dari bot dapat diakses pada *pranala berikut*

Pada tahap inilah perlu membaca dengan baik dokumentasi API yang ada sehingga nantinya tidak membuat fungsi-fungsi yang sebenarnya telah disediakan dan siap pakai.

Untuk memudahkan para pembaca dalam memulai untuk mengimplementasikan bot, terdapat beberapa API utama yang sering digunakan selama penggerjaan tugas besar ini, antara lain

- Class `IBaseBot` pada `Robocode.TankRoyale.BotApi`
API inti dari permainan Robocode berisi atribut dan metode inti apa saja yang dapat diperoleh dari *Game Engine*
- Class `IBot` pada `Robocode.TankRoyale.BotApi`
Perluasan API inti dengan metode yang memudahkan untuk bergerak, berbelok, ataupun menembakkan senjata.
- Class `ScannedBotEvent` pada `Robocode.TankRoyale.BotApi.Events`
API untuk mendapatkan informasi dari bot lawan yang terpindai.

Untuk API lainnya, silakan lakukan eksplorasi pada pranala yang telah dibagikan.

2.2.2 *Usage*

Setelah berhasil implementasi algoritma *greedy* pada bot, bagaimana caranya untuk dapat digunakan?

Untuk melakukan kompilasi pada bot yang ada pada tugas besar ini, dibutuhkan beberapa *dependencies*, *dependencies* utama yang dipakai pada bot tugas besar ini adalah dotnet versi 8.0, atau dapat diunduh pada dotnet.microsoft.com, dan git yang bisa diunduh pada <https://git-scm.com/>

Setelah itu, lakukan:

```
git clone  
https://github.com/zirachw/Tubes1_Rudal-Sekeloa-Reloaded.git
```

Jika ingin melakukan kompilasi pada bot alternatif, dan:

```
cd Tubes1_Rudal-Sekeloa-Reloaded/src/alternative-bots/[botname]
```

Jika ingin melakukan kompilasi pada bot utama.

```
cd Tubes1_Rudal-Sekeloa-Reloaded/src/main-bots/[botname]
```

Setelah itu, kompilasi bot sesuai OS, untuk Windows

```
[botname].cmd
```

Setelah itu, kompilasi bot sesuai OS, untuk berbasis UNIX

```
./[botname].sh
```

2.3. Game Engine

Game engine merupakan fondasi dari game Robocode Royale yang menjadi tempat dua atau lebih *bot* untuk bertarung. Ketika *game engine* menyala, maka server permainan akan dihidupkan dan *frontend* berupa menu GUI ditampilkan dan para *bot* bisa dimasukkan. *Game Engine* dari Robocode Royale dapat diunduh dari pranala berikut: [Robocode Tank Royale](#)

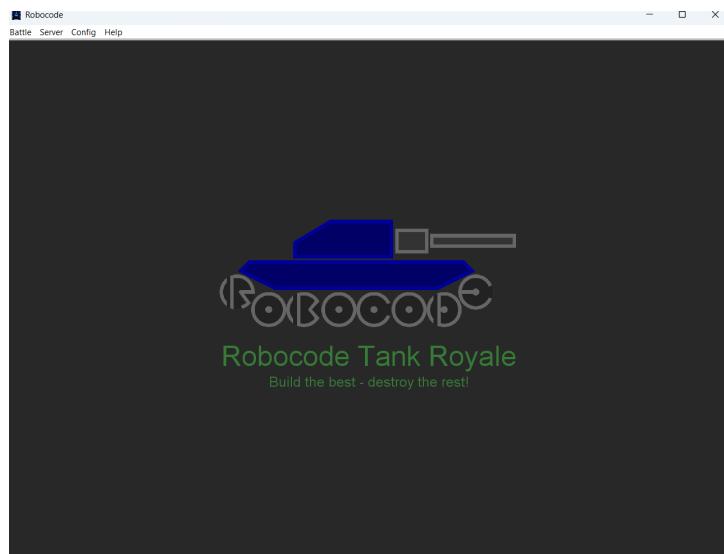
Dasar dari bagaimana Robocode Tank Royale berjalan adalah *Game Engine*. ketika *game engine* ini diaktifkan, maka *game engine* akan membuat suatu arena untuk bot tank melakukan duel dengan format 1 vs 1 vs ..., *game engine* juga akan menerima config bot dari user dengan menerima root folder dari bootnya dan akan membaca instruksi dari bot dengan format .cs (bahasa C#).

Selanjutnya, setiap bot akan diberikan energi yang sama, dengan *initial position* yang acak. Setiap waktu, *game engine* akan meminta gerakan (*turn*) yang akan dilakukan oleh bot sesuai dengan behaviour dari API yang dipakai. Jika waktu komputasi dalam satu *turn* terlalu lama, maka aksi dari bot akan di-skip sehingga dibutuhkan suatu algoritma yang sangat cepat namun menghasilkan poin semaksimum mungkin (sesuai dengan konsep algoritma *greedy* yang telah dijelaskan pada bagian 2.1)

Untuk menggunakan *Game Engine*, unduh terlebih dahulu *game engine* dari pranala sebelumnya. Pastikan berada pada *directory* yang sama dengan *game engine* yang telah diunduh, *game engine* dapat dijalankan dengan menggunakan *command*

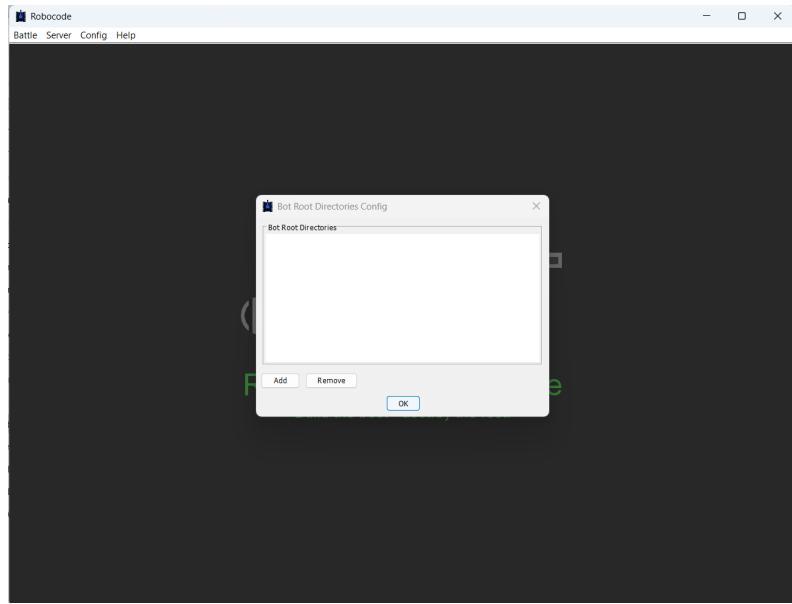
```
java -jar robocode-tankroyale-gui-x.y.z.jar
```

Kemudian akan muncul tampilan GUI utama:



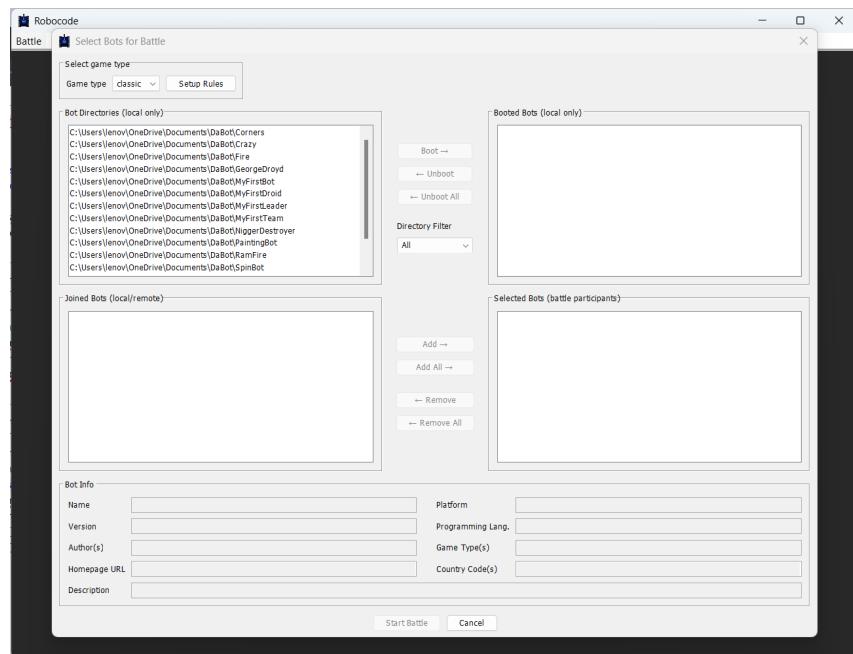
Gambar 5: Main UI Robocode Tank Royale

Untuk menambah *bot* bisa dilakukan dengan menambah *boot directories* terlebih dahulu pada opsi config atau bisa menggunakan shortcut CTRL+D.



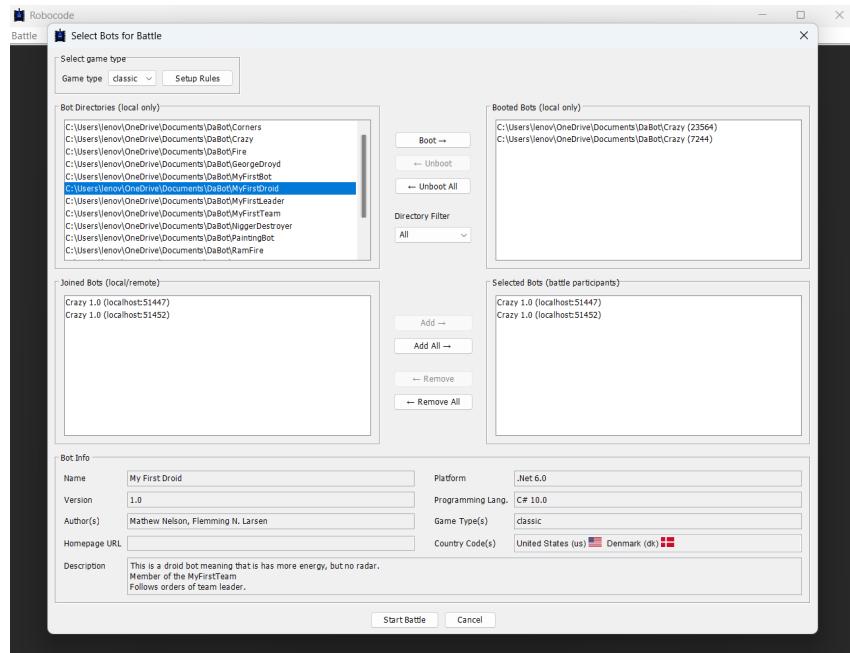
Gambar 6: Boot Directories

Untuk memulai permainan dapat dilakukan di opsi Start Battle yang ada dalam opsi Battle atau menggunakan shortcut CTRL+B.



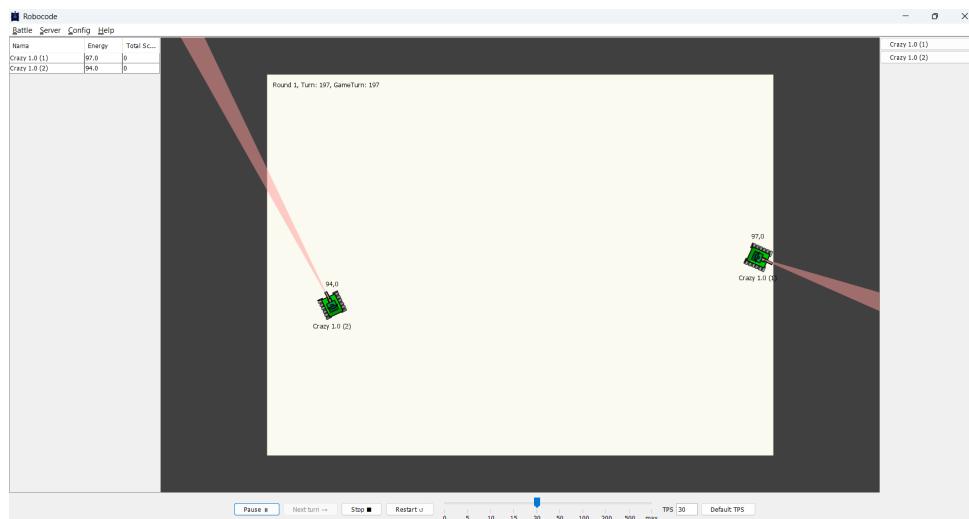
Gambar 7: Battle Config

Untuk menambahkan *bot* untuk bermain bisa dengan memilih bot pada *bot directories* kemudian *boot*, untuk memilih bot yang akan bertanding pilih dari *joined bots* dengan opsi Add. Terakhir, game bisa dimulai dengan Start Battle.



Gambar 8: Add Bot

Dan permainan sudah dimulai! ;).



Gambar 9: Battle Start

Jika bot tidak dapat di-boot, silakan merujuk dokumen pada pranala berikut

Bab III

Aplikasi Strategi *Greedy*

3.1. Abstraksi Permainan

Seperti yang telah didefinisikan sebelumnya, algoritma *greedy* hanya mengambil pilihan yang terbaik yang dapat diperoleh pada suatu saat, tanpa dapat kembali dengan harapan pilihan yang diambil menghasilkan solusi optimal.

Setelah melakukan *brainstorming* pada persoalan Robocode Tank Royale ini, dilakukan peninjauan terhadap komponen-komponen persoalan tersebut agar mempermudah proses abstraksi algoritma *greedy* pada bot permainan ini.

Ingat kembali bahwa tiap *rounds* terdiri atas sekian *turn* dan terdapat berbagai aksi pada bot yang dapat digunakan untuk tiap *turn*, antara lain bergerak, memindai, menembak, dan menabrak. Dengan meninjau tiap aksi, dilakukan *mapping* terhadap persoalan tersebut menjadi elemen-elemen algoritma *greedy*. *Mapping* yang dihasilkan dari persoalan tersebut dapat dilihat pada penjelasan di bawah ini.

a. Himpunan Kandidat (C)

Himpunan kandidat dalam permainan ini adalah semua aksi yang dapat dipertimbangkan oleh bot pada suatu *turn*, termasuk:

- Bergerak ke arah tertentu,
- Menembak dengan *firepower* tertentu,
- Memutar *gun*, *turret*, dan radar (memindai) ke arah tertentu,
- Menabrak (*ramming*) bot lawan.

b. Himpunan Solusi (S)

Himpunan solusi dalam permainan ini adalah jalur-jalur dari himpunan kandidat yang dapat menghasilkan skor tertinggi dengan *turn* paling sedikit.

- Aksi-aksi yang menghasilkan skor secara offensif (menembak dan menabrak),
- Aksi-aksi yang mengurangi kerusakan terhadap bot sendiri (penghematan energi dan *gun cooling rate*),
- Aksi-aksi yang menjaga posisi aman untuk bertahan hidup (pola gerakan/*pathing*)

c. Fungsi Solusi

Fungsi yang mengevaluasi apakah suatu strategi/aksi memberikan kontribusi signifikan terhadap tujuan utama, mengecek efektivitas hasil dari aksi/strategi yang dilakukan.

- Apakah bot berada dalam posisi yang lebih menguntungkan,
- Apakah bot berhasil mencetak skor dari serangan,
- Apakah bot berhasil bertahan dari serangan lawan,
- Apakah hasil *turn* saat ini mendekatkan bot pada skor tertinggi.

d. Fungsi Seleksi (*Selection Function*)

Fungsi yang memilih aksi terbaik dan paling menjanjikan dari himpunan kandidat berdasarkan nilai heuristik tertentu pada setiap *turn*. Heuristik ini bisa berbeda-beda tergantung strategi bot, misalnya:

- Estimasi potensi skor dari suatu aksi,
- Estimasi risiko jika aksi tersebut diambil,
- Jarak terhadap musuh atau tembok,
- Energi tersisa bot dan musuh.

e. Fungsi Kelayakan (*Feasible*)

Fungsi yang memfilter aksi-aksi yang tidak mungkin atau tidak sah untuk dijalankan.

- Tidak melebihi kecepatan atau rotasi maksimum,
- Tidak menembak saat suhu meriam terlalu panas,
- Tidak menembak saat energi yang ada kurang dari *firepower*.
- Tidak bergerak menuju *wall*

f. Fungsi Obyektif

Fungsi yang mendefinisikan tujuan akhir yang ingin dicapai bot, yaitu memaksimalkan skor total pada akhir pertandingan dari akumulasi tiap *round*.

3.2. Eksplorasi Alternatif Solusi *Greedy*

Dalam *brainstorming* penyelesaian persoalan Robocode Tank Royale ini, telah diidentifikasi beberapa alternatif solusi *greedy* yang menjanjikan untuk meningkatkan performa bot dalam permainan ini. Berikut beberapa alternatif solusi yang didapatkan :

3.2.1 Kaze (*Energy Wise*)

Kaze adalah bot yang diimplementasikan menggunakan pendekatan heuristik "*greedy by energy, distance, and bullet damage bonus*" dengan *path* yang dibentuk melalui belok kanan setiap menabrak sesuatu, seperti *Wall* dan bot lawan. Pendekatan heuristiknya membuat bot akan menembak peluru secara agresif jika memiliki ***energy advantage*** dan akan melakukan *kill steal* dengan peluru dengan energi seminimum mungkin jika ada musuh yang memiliki energi rendah (agar menaikkan potensi *hit*).

$$\text{firepower} = \begin{cases} \text{Min}(3, \text{Energy}), & \text{Energy} > \text{enemy.Energy} + 30, \\ 1, & \text{enemy.Energy} < 15. \end{cases}$$

Selain itu, penembakan peluru juga berdasarkan dengan *distance*. Setiap jenis peluru memiliki nilai *velocity* dan *firepower* yang berbeda, maka sebisa mungkin agar membuat bot selalu *hit* saat menembak. Jika musuh sangat jauh, maka tank akan menembak peluru dengan tipe 1 (peluru kecil namun sangat cepat). Sebaliknya, jika musuh cukup dekat, maka tank akan

menembak peluru dengan tipe 2 (peluru medium dengan kecepatan biasa), dan jika musuh sangat dekat, maka tank akan menembak peluru dengan tipe 3 (peluru besar namun lambat).

$$\text{firepower} = \begin{cases} 3, & \text{Distance} < 100, \\ 2, & \text{Distance} < 200, \\ 1, & \text{otherwise.} \end{cases}$$

a. Analisis Efisiensi Program

Strategi ini menawarkan solusi yang cukup *naive*, yakni mencari poin dengan cara menembak musuh dengan mempertimbangkan *advantage* yang dimiliki, seperti energi dan jarak sehingga bot ini akan sangat efisien untuk mendapatkan skor dengan cepat jika melawan bot yang memiliki pattern yang tidak random karena pemindai pada bot ini sengaja dibuat lurus secara keseluruhan untuk mendukung pendekatan heuristiknya.

b. Analisis Efektivitas Program

Untuk efektivitas, bot ini cukup efektif, bot bisa mendapatkan skor instan jika ada musuh yang *patternnya* cukup acak. Hanya saja, kekurangan dari pendekatan heuristik ini adalah bot tidak melakukan pemindaian pada arena secara keseluruhan yang berpotensi menyebabkan ada musuh yang di dekatnya namun tidak terdeteksi, tetapi hal ini coba untuk *di-handle* dengan manuver dan *radarnya* jika ada bot terkena damage (*wall*, *on hit*, dan *ram*), namun ada kondisi khusus jika dia terkena damage karena *ramming*, yakni dia akan melakukan rotasi untuk mencari bot musuh dan akan melakukan serangan balasan. Selain itu, bot ini juga bergerak dalam pattern *pseudo-random square* dan gerakan ke dalam dan ke luar arena untuk melakukan pemindaian.

3.2.2 Sweepredict (Scan All & Predict)

Sweepredict menggunakan pendekatan heuristik *greedy by bullet hit*, yakni pendekatan heuristik yang membuat bot mencoba untuk melakukan prediksi kepada pergerakan musuh berdasarkan posisi pemindaian terakhir. Radar akan selalu diputar secara 360° untuk mendapatkan posisi musuh, kemudian bot akan memprediksi posisi musuh berdasarkan arah dan kecepatan musuh dan menembak sesuai dengan hasil prediksi posisi tersebut.

$$\begin{aligned} \text{predictedX} &= e.X + \cos(e.Direction \cdot \pi/180) \cdot e.Speed \cdot (\text{distance}/\text{bulletSpeed}) \\ \text{predictedY} &= e.Y + \sin(e.Direction \cdot \pi/180) \cdot e.Speed \cdot (\text{distance}/\text{bulletSpeed}) \end{aligned}$$

Selain itu, pada awal permainan bot akan memposisikan dirinya menuju posisi sudut arena terdekat untuk meng-*align* posisinya terhadap *padded area* yang terdefinisi agar tidak pernah menabrak *Wall* untuk tidak membuang energi akibat tertabrak *Wall*.

Strategi ini menawarkan solusi yang *naive*, yakni mencari poin dengan berusaha mengenai bot musuh sebanyak mungkin dengan melakukan prediksi posisi musuh berdasarkan kecepatan dan arah gerak bot musuh.

a. Analisis Efisiensi Program

Secara efisiensi, bot ini cukup efisien dalam mendapatkan skor, keuntungan dari pendekatan ini adalah dengan menggunakan prediksi posisi musuh untuk menembak, peluru akan digunakan secara lebih efisien sehingga penggunaan energi tidak akan sia-sia, selain itu akurasi tembakan juga akan lebih akurat dan *chance* peluru bot untuk mengenai musuh akan lebih tinggi. Dan juga, bot ini akan terus melakukan pemindaian, sehingga bot ini sangat cepat untuk menemukan musuh.

b. Analisis Efektivitas Program

Pendekatan ini cukup efektif untuk mendapatkan skor karena mempertimbangkan prediksi untuk melemparkan pelurunya, tetapi *pathing* dari bot ini cukup sederhana sehingga lebih mudah ditarget oleh bot lain.

3.2.3 Waves

Waves bot adalah bot yang diimplementasikan menggunakan pendekatan heuristik *greedy by survival* yang membuat bot mencoba bertahan hidup selama mungkin. Bot bertahan hidup dengan melakukan sebuah *pattern* berupa gelombang menggunakan sifat dari fungsi sinus.

```
waveAngle = sin(wavePosition / WavePeriod) · WaveAmplitude  
SetTurnRight(waveAngle)  
SetForward(stepSize)
```

Pattern dimulai dengan bot menuju sisi dinding terdekat kemudian memulai pola gelombang menuju ke sudut dan kemudian akan mengulanginya. Selain itu, bot akan menembak musuh dengan *confidence level* sesuai faktor *distance* dan *speed* dari musuh.

$$\text{aimConfidence} = 1 \cdot [1 - (\text{distance} - 1000)] \cdot [1 - \text{Min}(8, \text{e.Speed})/16.0]]$$

Bot juga dapat mendeteksi ketika sedang di-*ram* oleh robot lainnya sehingga dapat melakukan *counter* dengan menembak dengan peluru terbesar.

Strategi ini menawarkan solusi yang *naive*, yakni berusaha mencari poin dengan bertahan hidup selama mungkin dengan melakukan *path planning* berbentuk gelombang dan bergerak pada *defined area* sebab *path* dibuat sedemikian agar memiliki *padding* pada dindingnya agar bot se bisa mungkin tidak akan menabrak dinding yang akan berakibat mengurangnya *energy*.

a. Analisis Efisiensi Program

Secara efisiensi, bot ini tidak mendapatkan skor secara instan dan cepat, tetapi, keunggulan dari strategi ini adalah bot bergerak secara sinusoidal sehingga lebih sulit untuk ditarget. Selain itu, penggunaan *firepower* yang menyesuaikan dengan jarak musuh membuat *energy* semakin hemat dan dapat memperbesar poin yang bisa didapatkan.

b. Analisis Efektivitas Program

Untuk efektivitas, bot ini cukup efektif bisa bertahan hingga akhir karena mendapatkan bonus *survival point*, karena kelemahan dari bot ini adalah kurangnya kemampuan untuk menyerang sehingga bot ini akan lebih cenderung untuk bertahan dibandingkan menyerang, ini membuat bot menjadi sasaran empuk bagi musuh yang dapat melakukan *tracking*.

3.2.4 RudalSekelo (Scan & Track + Stacking Heuristics)

Scanner bot adalah bot yang diimplementasikan menggunakan pendekatan heuristic *greedy by bullet point*, yakni pendekatan heuristic yang membuat bot akan memiliki radar yang me-*lock* musuh dalam radar yang terus mengikuti pergerakan bot musuh dan menembak mengikuti pergerakan musuh.

```
angleToEnemy = arctan2(enemyY - Y, enemyX - X) · 180/π  
radarTurn = NormalizeRelativeAngle(angleToEnemy - RadarDirection)  
SetTurnRadarLeft(radarTurn * 2)
```

Bot juga akan melakukan *greedy by enemy energy*, yakni memilih bot musuh dengan *energy* yang terendah untuk di-*lock*, terakhir bot juga akan memilih *firepower* sesuai dengan jarak terhadap musuh yang sedang di-*lock*.

Selain itu, bot akan memiliki pergerakan yang memaksimalkan kemungkinan peluru mengenai musuh dengan melakukan *strafe* kearah musuh dengan menuju posisi yang *perpendicular* terhadap arah pergerakan musuh.

Strategi ini menawarkan solusi yang *naive*, yakni mencari poin dengan mengikuti dan menembak musuh sebanyak-banyaknya.

a. Analisis Efisiensi Program

Bot ini merupakan bot yang paling efisien dibandingkan dengan 3 bot lain dalam mendapatkan skor. Keuntungan dari strategi ini adalah bot akan selalu menembak musuh pada posisi terbarunya setiap saat dikarenakan radar bot akan selalu mengunci tiap pergerakan dari musuh dan melakukan pemilihan *firepower* peluru berdasarkan *distance* untuk memaksimalkan poin, selain itu bot ini juga memiliki *predictive radar* sehingga bisa memprediksi ke mana bot musuh akan bergerak.

b. Analisis Efektivitas Program

Untuk efektivitas, bot ini efektif untuk mendapatkan skor secara cepat dan instan, hanya saja ketika melakukan *lock* pada suatu target, bot akan menjadi "buta" terhadap lingkungannya termasuk bot musuh lain, sehingga bot ini dapat menjadi sasaran yang empuk bagi bot musuh lainnya.

3.3. Pemilihan Strategi *Greedy* Utama

Strategi *greedy* utama yang dipilih adalah bot **RudalSekeloa** dikarenakan potensi *score* secara agresif yang ditawarkan oleh fitur *scan and tracking* dapat me-*outscale* potensi yang ditawarkan oleh bot lainnya. Dengan kemampuan ini, bot dapat memaksimalkan perolehan poin melalui pendekatan yang lebih proaktif dibandingkan alternatif lainnya.

Selain itu, dengan menginkorporasikan strategi yang ditawarkan oleh bot lainnya, kita dapat melakukan *heuristic stacking*, dengan ini kita dapat menutupi kelemahan-kelemahan bot utama yang berdasarkan **RudalSekeloa**. Dengan menggabungkan berbagai strategi ini dapat menciptakan pendekatan yang lebih komprehensif, sehingga memungkinkan bot untuk beradaptasi dalam berbagai situasi sambil tetap mempertahankan keunggulan kompetitifnya.

Bab IV

Implementasi dan Pengujian

4.1. Implementasi Algoritma Greedy pada Bot

(a) Kaze

```
function Kaze(C : himpunan_kandidat) -> himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma
greedy }

Deklarasi:
    nearWall : boolean
    currentDistance, nearWallDistance, innerDistance, margin : real
    segmentCount : integer
    bulletPower : real
    distance : real
    energy, enemyEnergy : real

Algoritma:
    while IsRunning do
        if nearWall then
            currentDistance <- nearWallDistance
        else
            currentDistance <- innerDistance
        endif

        Forward(currentDistance)
        TurnRight(90)

        segmentCount <- segmentCount + 1
        if segmentCount >= 4 then
            if nearWall then
                TurnRight(90)
                Forward(margin)
                nearWall <- false
            else
                TurnLeft(90)
                Forward(margin)
                nearWall <- true
            endif
            segmentCount <- 0
        endif

        if ScannedBot then
            distance <- DistanceTo(enemy.X, enemy.Y)
            bulletPower <- 1

            if distance < 100 then
                bulletPower <- 3
            else if distance < 200 then
                bulletPower <- 2
            else
                bulletPower <- 1
            endif
        endif
    endwhile
```

```

        if energy > enemyEnergy + 30 then
            bulletPower <- Min(3, energy)
        else if enemyEnergy < 15 then
            bulletPower <- 1
        endif

        SetFire(bulletPower)
    endif

    if onHitBot then
        bearing <- BearingTo(enemy.X, enemy.Y)
        if bearing > -90 and bearing < 90 then
            Back(100)
        else
            Forward(100)
        endif
    endif

    if onHitByBullet then
        TurnRight(45)
        Forward(50)
    endif

    Rescan()
endWhile

```

(b) Sweepredict

```

function Sweepredict(C : himpunan_kandidat) -> himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma
greedy }

```

Deklarasi:

```

nearWall : boolean
currentDistance, nearWallDistance, innerDistance: real
margin : real
segmentCount : integer
bulletPower : real
distance, maxShootingDistance : real
angleToEnemy, gunTurn, firePower : real
bulletSpeed, timeToHit : real
enemyHeadingRadians, predictedX, predictedY : real
angleToEnemyPredicted, gunTurnPredicted : real
dir : integer
right, left, top, bottom : real

```

Algoritma:

```
GoToNearestCorner()
```

```

while IsRunning do
    SetTurnRadarRight(infinity)
    dir <- Direction

    if dir = 0 or dir = 360 then
        Forward(right - X)
        TurnRight(90)
    else if dir = 90 then
        Forward(top - Y)
        TurnRight(90)

```

```

        else if dir = 180 then
            Forward(X - left)
            TurnRight(90)
        else if dir = 270 then
            Forward(Y - bottom)
            TurnRight(90)
        else
            TurnToCardinalDirection(0)
        endif

        if onScannedBot then
            distance <- DistanceTo(enemy.X, enemy.Y)

            if distance > maxShootingDistance then
                angleToEnemy <- arctan2(enemy.Y - Y, enemy.X - X)
                * 180 / pi
                gunTurn <- NormalizeRelativeAngle(angleToEnemy -
                GunDirection)
                SetTurnGunLeft(gunTurn)
                continue
            endif

            firePower <- 1.0
            if distance <= 50 then
                firePower <- 3.0      {Close range}
            else if distance <= 150 then
                firePower <- 2.5      {Medium-close}
            else if distance <= 300 then
                firePower <- 2.0      {Medium range}
            else if distance <= 450 then
                firePower <- 1.5      {Medium-far}
            endif

            bulletSpeed <- 20 - 3 * firePower
            timeToHit <- distance / bulletSpeed

            enemyHeadingRadians <- enemy.Direction * pi / 180
            predictedX <- enemy.X + cos(enemyHeadingRadians)
            * enemy.Speed * timeToHit
            predictedY <- enemy.Y + sin(enemyHeadingRadians)
            * enemy.Speed * timeToHit

            angleToEnemyPredicted <- arctan2(predictedY - Y,
            predictedX - X) * 180 / pi
            gunTurnPredicted <- NormalizeRelativeAngle(
            angleToEnemyPredicted - GunDirection)
            SetTurnGunLeft(gunTurnPredicted)

            if abs(gunTurnPredicted) < 10 then
                SetFire(firePower)
            endif
        endif

        Rescan()
    endwhile

```

(c) Waves

```
function Waves(C : himpunan_kandidat) -> himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma
greedy }

Deklarasi:
    nearWall : boolean
    currentDistance, nearWallDistance, innerDistance: real
    margin : real
    segmentCount : integer
    bulletPower : real
    distance, maxShootingDistance : real
    angleToEnemy, gunTurn, firePower : real
    bulletSpeed, timeToHit : real
    enemyHeadingRadians, predictedX, predictedY : real
    angleToEnemyPredicted, gunTurnPredicted : real
    dir : integer
    right, left, top, bottom : real

Algoritma:
    GoToNearestCorner()

    while IsRunning do
        SetTurnRadarRight(infinity)
        dir <- Direction

        if dir = 0 or dir = 360 then
            Forward(right - X)
            TurnRight(90)
        else if dir = 90 then
            Forward(top - Y)
            TurnRight(90)
        else if dir = 180 then
            Forward(X - left)
            TurnRight(90)
        else if dir = 270 then
            Forward(Y - bottom)
            TurnRight(90)
        else
            TurnToCardinalDirection(0)
        endif

        if onScannedBot then
            distance <- DistanceTo(enemy.X, enemy.Y)

            if distance > maxShootingDistance then
                angleToEnemy <- arctan2(enemy.Y - Y, enemy.X - X)
                * 180 / pi
                gunTurn <- NormalizeRelativeAngle(angleToEnemy -
                GunDirection)
                SetTurnGunLeft(gunTurn)
                continue
            endif

            firePower <- 1.0
            if distance <= 50 then
                firePower <- 3.0      {Close range}
            else if distance <= 150 then
```

```

        firePower <- 2.5      {Medium-close}
    else if distance <= 300 then
        firePower <- 2.0      {Medium range}
    else if distance <= 450 then
        firePower <- 1.5      {Medium-far}
    endif

    bulletSpeed <- 20 - 3 * firePower
    timeToHit <- distance / bulletSpeed

    enemyHeadingRadians <- enemy.Direction
    * pi / 180
    predictedX <- enemy.X + cos(enemyHeadingRadians)
    * enemy.Speed * timeToHit
    predictedY <- enemy.Y + sin(enemyHeadingRadians)
    * enemy.Speed * timeToHit

    angleToEnemyPredicted <- arctan2(predictedY - Y,
    predictedX - X) * 180 / pi
    gunTurnPredicted <- NormalizeRelativeAngle(
    angleToEnemyPredicted - GunDirection)
    SetTurnGunLeft(gunTurnPredicted)

    if abs(gunTurnPredicted) < 10 then
        SetFire(firePower)
    endif
endif

Rescan()
endWhile

```

(d) RudalSekeloa

```

function RudalSekeloa() -? void
{ Bot utama dengan logika hybrid heuristik }
Deklarasi:
    Energy, firePower, maxGunAngleError, aimConfidence : real
    Mode : string
    lastScannedBot : Bot
    strafeDirection : integer
    left, right, bottom, top : real
    distance, angleToEnemy, gunTurn, radarTurn : real
    bulletSpeed, timeToHit : real
    enemyHeadingRadians, predictedX, predictedY : real
    angleToEnemyPredicted, gunTurnPredicted : real
    maxShootingDistance, adjustedFirePower : real
    isDefensiveMode : boolean

Algoritma:
    while (IsRunning) do
        isDefensiveMode <- Energy < lowEnergyThreshold

        SetTurnRadarRight(360)
        SetTurnRadarLeft(360)

        if (lastScannedBot = null) then
            SetTurnRadarRight(360)
            SetTurnRadarLeft(360)
        endif

```

```

if (isDefensiveMode) then
    if (lastScannedBot != null) then
        TrackTargetRadarOnly()
    endif

    GoToNearestMiddleOfASide()
    WaitFor(new MovementCompleteCondition(this))
    MoveWavySide()

else
    if (lastScannedBot != null) then
        TrackTarget()
    else
        SetTurnRadarRight(360)
    endif

    Go()
endif

if onScannedBot then
    if (lastScannedBot = null OR enemy.ScannedBotId
    = lastScannedBot.ScannedBotId OR enemy.Energy <
    lastScannedBot.Energy) then
        lastScannedBot <- enemy
    endif

    distance <- DistanceTo(enemy.X, enemy.Y)
    angleToEnemy <- arctan2(enemy.Y - Y, enemy.X - X)
    * 180 / pi
    gunTurn <- NormalizeRelativeAngle(angleToEnemy
    - GunDirection)

    radarTurn <- NormalizeRelativeAngle(
    angleToEnemy - RadarDirection)
    SetTurnRadarLeft(radarTurn * 2)

if (distance > maxShootingDistance) then
    SetTurnGunLeft(gunTurn)
    continue
endif

firePower <- calculateFirePower(distance)
if (Energy > enemy.Energy + 30) then
    firePower <- min(firePower, Energy / 10)
endif

if (enemy.Energy < 16) then
    if (distance < 100) then
        firePower <- 3.0
    else if (distance < 300) then
        firePower <- 2.5
    endif
endif

firePower <- min(firePower, Energy)

bulletSpeed <- 20 - 3 * firePower
timeToHit <- distance / bulletSpeed

```

```

        enemyHeadingRadians <- enemy.Direction
        * pi / 180
        predictedX <- enemy.X + cos(enemyHeadingRadians)
        * enemy.Speed * timeToHit
        predictedY <- enemy.Y + cos(enemyHeadingRadians)
        * enemy.Speed * timeToHit

        angleToEnemyPredicted <- arctan2(predictedY - Y,
        predictedX - X) * 180 / pi
        gunTurnPredicted <- NormalizeRelativeAngle(
        angleToEnemyPredicted - GunDirection)
        SetTurnGunLeft(gunTurnPredicted)

        if (distance < 50) then
            aimConfidence <- 0.95
        else
            aimConfidence <- 1.0
            aimConfidence <- aimConfidence *
            (1.0 - (distance / 1000.0))
            aimConfidence <- aimConfidence *
            (1.0 - (min(8, enemy.Speed) / 16.0))
            aimConfidence <- aimConfidence *
            (1.0 - (min(45, abs(gunTurnPredicted)) / 120.0))
        endif

        if (distance < 50) then
            maxGunAngleError <- 25
        else
            maxGunAngleError <- 15 * aimConfidence
        endif

        if (abs(gunTurnPredicted) < maxGunAngleError
        AND aimConfidence > 0.3 AND GunHeat = 0) then
            if (distance < 50) then
                adjustedFirePower <- firePower
            else
                adjustedFirePower <- firePower *
                max(0.8, aimConfidence)
            endif
            Fire(adjustedFirePower)
        endif
    endif

    if onHitByBullet then
        strafeDirection <- strafeDirection * -1
    endif

    if onHitWall then
        if (isDefensiveMode) then
            Stop()
            GoToNearestMiddleOfASide()
        else
            strafeDirection <- strafeDirection * -1
            SetTurnRight(90)
        endif
    endif
    endwhile
end function

```

4.2. Implementasi Solusi *Greedy* yang dipilih



The screenshot shows a code editor window with a dark theme. At the top, there are three circular icons: red, yellow, and green. The code itself is written in C# and defines a class named `RudalSekeloa` that inherits from `Bot`. The code includes various private constants and variables, along with comments explaining the bot's behavior and heuristics.

```
1  using System;
2  using System.Drawing;
3  using Robocode.TankRoyale.BotApi;
4  using Robocode.TankRoyale.BotApi.Events;
5
6  /*
7  -----
8  RudalSekelo - A hybrid heuristic bot with main Tracking Scanner
9  -----
10 Main Heuristics:
11 - Offensive mode: main tracking with strafing to an enemy
12 - Defensive mode: wavy movement along the walls
13
14 Combines:
15 - Kaze's energy-aware firing strategy
16 - Sweepredict's radar tracking and enemy prediction
17 - Waves's wavy defensive movement for low energy situations
18 - Waves's predictive targeting with confidence levels
19 -----
20 */
21
22 public class RudalSekelo : Bot
23 {
24     // Energy threshold for switching to defensive mode
25     private const double lowEnergyThreshold = 30.0;
26
27     // Desired tracking distance from enemy
28     private const double trackingDistance = 150;
29
30     // Tolerance range for distance adjustment
31     private const double tolerance = 20;
32
33     // Scanned bot information
34     private ScannedBotEvent lastScannedBot = null;
35
36     // Strafing direction (1 = right, -1 = left)
37     private int strafeDirection = 1;
38
39     // Size of the wave
40     private const double WaveAmplitude = 60;
41
42     // Controls frequency of waves
43     private const double WavePeriod = 40;
44
45     // Maximum moves to limit continuous waving
46     private const int maxMoves = 50;
47
48     // Padding borders
49     private const double padding = 30;
```

Gambar 10: Implementasi RudalSekelo (Main Bot) - Part 1

```
1 // Maximum allowed shooting distance
2 private const double maxShootingDistance = 500;
3
4 // Each side of padded area
5 private double left, right, bottom, top;
6
7 // Firepower
8 private double firePower;
9
10 // Bot state
11 private bool isDefensiveMode = false;
12
13 static void Main(string[] args)
14 {
15     new RudalSekeloa().Start();
16 }
17
18 RudalSekeloa() : base(BotInfo.FromFile("RudalSekeloa.json")) { }
19
20 public override void Run()
21 {
22     // Set-up colors
23     BodyColor = Color.DarkGreen;
24     TurretColor = Color.DarkBlue;
25     RadarColor = Color.Yellow;
26     BulletColor = Color.Black;
27     ScanColor = Color.Cyan;
28
29     // Configure independent turning
30     AdjustGunForBodyTurn = true;
31     AdjustRadarForBodyTurn = true;
32     AdjustRadarForGunTurn = true;
33
34     // Define boundary limits for defensive mode
35     left = padding;
36     right = ArenaWidth - padding;
37     bottom = padding;
38     top = ArenaHeight - padding;
39
40     while (IsRunning)
41     {
42         // Update defensive mode based on energy Level
43         isDefensiveMode = Energy < lowEnergyThreshold;
44
45         // Continuous scanning
46         SetTurnRadarRight(360);
47         SetTurnRadarLeft(360);
48
49         if (lastScannedBot == null)
50         {
51             // If no target, keep scanning
52             SetTurnRadarRight(360);
53             SetTurnRadarLeft(360);
54         }
55
56         // Defensive movement if energy is low
57         if (isDefensiveMode)
58         {
59             if (lastScannedBot != null) TrackTargetRadarOnly();
60             GoToNearestMiddleOfASide();
61             WaitFor(new MovementCompleteCondition(this));
62             MoveWavySide();
63         }
64
65         // Offensive tracking otherwise
66         else
67         {
68             if (lastScannedBot != null) TrackTarget();
69             else SetTurnRadarRight(360);
70             Go();
71         }
72     }
73 }
```

Gambar 11: Implementasi RudalSekeloa (Main Bot) - Part 2



```
1 // Track target with strafing and distance adjustment
2 private void TrackTarget()
3 {
4     if (lastScannedBot == null) return;
5
6     double enemyX = lastScannedBot.X;
7     double enemyY = lastScannedBot.Y;
8     double angleToEnemy = Math.Atan2(enemyY - Y, enemyX - X) * 180 / Math.PI;
9     double distanceToEnemy = Math.Sqrt((enemyX - X) * (enemyX - X) + (enemyY - Y) * (enemyY - Y));
10
11    // Aim gun at enemy
12    double gunTurn = NormalizeRelativeAngle(angleToEnemy - GunDirection);
13    SetTurnGunLeft(gunTurn);
14
15    // Adjust distance if not within the desired tracking range
16    double distanceError = distanceToEnemy - trackingDistance;
17    if (Math.Abs(distanceError) > tolerance)
18    {
19        if (distanceError > 0)
20        {
21            SetForward(Math.Min(distanceError, 100));
22        }
23        else
24        {
25            SetBack(Math.Min(-distanceError, 100));
26        }
27    }
28
29    // Do strafing movement otherwise
30    else
31    {
32        double desiredStrafeAngle = NormalizeAbsoluteAngle(angleToEnemy + 90 * strafeDirection);
33        double turnToStrafe = NormalizeRelativeAngle(desiredStrafeAngle - Direction);
34
35        SetTurnRight(turnToStrafe);
36        SetForward(70);
37    }
38 }
39
40 // Track only with radar and gun (for defensive mode)
41 private void TrackTargetRadarOnly()
42 {
43     if (lastScannedBot == null) return;
44
45     double enemyX = lastScannedBot.X;
46     double enemyY = lastScannedBot.Y;
47     double angleToEnemy = Math.Atan2(enemyY - Y, enemyX - X) * 180 / Math.PI;
48
49     // Aim gun at enemy
50     double gunTurn = NormalizeRelativeAngle(angleToEnemy - GunDirection);
51     SetTurnGunLeft(gunTurn);
52
53     // Lock radar on enemy with wider sweep (overcorrection)
54     double radarTurn = NormalizeRelativeAngle(angleToEnemy - RadarDirection);
55     SetTurnRadarLeft(radarTurn * 2);
56 }
```

Gambar 12: Implementasi RudalSekelo (Main Bot) - Part 3



```
1 // Lock radar on enemy with wider sweep (overcorrection)
2 double radarTurn = NormalizeRelativeAngle(angleToEnemy - RadarDirection);
3 SetTurnRadarLeft(radarTurn * 2);
4 }
5 }
6
7 // Method to move to the nearest middle of a side of the padded area
8 private void GoToNearestMiddleOfASide()
9 {
10     double middleX = (left + right) / 2;
11     double middleY = (bottom + top) / 2;
12
13     // Find the distance to the middle point of each side using the built-in DistanceTo method
14     double dLeftSide = DistanceTo(left, middleY);
15     double dRightSide = DistanceTo(right, middleY);
16     double dBottomSide = DistanceTo(middleX, bottom);
17     double dTopSide = DistanceTo(middleX, top);
18
19     // Determine which side middle is closest
20     double minDist = dLeftSide;
21     string nearestSide = "left";
22     if (dRightSide < minDist)
23     {
24         minDist = dRightSide;
25         nearestSide = "right";
26     }
27
28     if (dBottomSide < minDist)
29     {
30         minDist = dBottomSide;
31         nearestSide = "bottom";
32     }
33
34     if (dTopSide < minDist)
35     {
36         minDist = dTopSide;
37         nearestSide = "top";
38     }
39
40     // Check if already at the middle of a side (distance is very close to 0)
41     bool alreadyAtMiddle = minDist < 0.001;
42
43     if (nearestSide == "left")
44     {
45         // Turn towards middle of Left side and facing northeast
46         if (!alreadyAtMiddle)
47         {
48             double angle = Math.Atan2(middleY - Y, left - X) * 180 / Math.PI;
49             TurnToCardinalDirection(angle);
50             Forward(minDist);
51             WaitFor(new MovementCompleteCondition(this));
52         }
53
54         TurnToCardinalDirection(45);
55     }
56     else if (nearestSide == "right")
57     {
58         // Turn towards middle of right side and facing southwest
59         if (!alreadyAtMiddle)
60         {
61             double angle = Math.Atan2(middleY - Y, right - X) * 180 / Math.PI;
62             TurnToCardinalDirection(angle);
63             Forward(minDist);
64             WaitFor(new MovementCompleteCondition(this));
65         }
66
67         TurnToCardinalDirection(225);
68     }
69     else if (nearestSide == "bottom")
70     {
71         // Turn towards middle of bottom side and facing northwest
72         if (!alreadyAtMiddle)
73         {
74             double angle = Math.Atan2(bottom - Y, middleX - X) * 180 / Math.PI;
75             TurnToCardinalDirection(angle);
76             Forward(minDist);
77             WaitFor(new MovementCompleteCondition(this));
78         }
79
80         TurnToCardinalDirection(135);
81     }
82     else // (nearestSide == "top")
83     {
84
85         // Turn towards middle of top side and facing southeast
86         if (!alreadyAtMiddle)
87         {
88             double angle = Math.Atan2(top - Y, middleX - X) * 180 / Math.PI;
89             TurnToCardinalDirection(angle);
90             Forward(minDist);
91             WaitFor(new MovementCompleteCondition(this));
92         }
93         TurnToCardinalDirection(315);
94     }
95 }
```

Gambar 13: Implementasi RudalSekelo (Main Bot) - Part 4



```

1 // Method to move to the nearest middle of a side of the padded area
2 private void GoToNearestMiddleOfASide()
3 {
4     double middleX = (left + right) / 2;
5     double middleY = (bottom + top) / 2;
6
7     // Find the distance to the middle point of each side using the built-in DistanceTo method
8     double dLeftSide = DistanceTo(left, middleY);
9     double dRightSide = DistanceTo(right, middleY);
10    double dBottomSide = DistanceTo(middleX, bottom);
11    double dTopSide = DistanceTo(middleX, top);
12
13    // Determine which side middle is closest
14    double minDist = dLeftSide;
15    string nearestSide = "left";
16    if (dRightSide < minDist)
17    {
18        minDist = dRightSide;
19        nearestSide = "right";
20    }
21
22    if (dBottomSide < minDist)
23    {
24        minDist = dBottomSide;
25        nearestSide = "bottom";
26    }
27
28    if (dTopSide < minDist)
29    {
30        minDist = dTopSide;
31        nearestSide = "top";
32    }
33
34    // Check if already at the middle of a side (distance is very close to 0)
35    bool alreadyAtMiddle = minDist < 0.001;
36
37    if (nearestSide == "left")
38    {
39        // Turn towards middle of left side and facing northeast
40        if (!alreadyAtMiddle)
41        {
42            double angle = Math.Atan2(middleY - Y, left - X) * 180 / Math.PI;
43            TurnToCardinalDirection(angle);
44            Forward(minDist);
45            WaitFor(new MovementCompleteCondition(this));
46        }
47
48        TurnToCardinalDirection(45);
49    }
50    else if (nearestSide == "right")
51    {
52        // Turn towards middle of right side and facing southwest
53        if (!alreadyAtMiddle)
54        {
55            double angle = Math.Atan2(middleY - Y, right - X) * 180 / Math.PI;
56            TurnToCardinalDirection(angle);
57            Forward(minDist);
58            WaitFor(new MovementCompleteCondition(this));
59        }
60
61        TurnToCardinalDirection(225);
62    }
63    else if (nearestSide == "bottom")
64    {
65        // Turn towards middle of bottom side and facing northwest
66        if (!alreadyAtMiddle)
67        {
68            double angle = Math.Atan2(bottom - Y, middleX - X) * 180 / Math.PI;
69            TurnToCardinalDirection(angle);
70            Forward(minDist);
71            WaitFor(new MovementCompleteCondition(this));
72        }
73
74        TurnToCardinalDirection(135);
75    }
76    else // (nearestSide == "top")
77    {
78
79        // Turn towards middle of top side and facing southeast
80        if (!alreadyAtMiddle)
81        {
82            double angle = Math.Atan2(top - Y, middleX - X) * 180 / Math.PI;
83            TurnToCardinalDirection(angle);
84            Forward(minDist);
85            WaitFor(new MovementCompleteCondition(this));
86        }
87        TurnToCardinalDirection(315);
88    }
89 }

```



The screenshot shows a code editor window with a dark theme. At the top left, there are three colored circular icons: red, yellow, and green. The code itself is a Java file with line numbers from 1 to 72 on the left. The code implements a wavy square movement pattern, handles boundary conditions, and includes methods for turning and moving forward.

```
1 // Move along one side of the wavy square
2 private void MoveWavySide()
3 {
4     // First check if we're out of bounds and handle it efficiently
5     if (X < left || X > right || Y < bottom || Y > top)
6     {
7         HandleOutOfBounds();
8         return;
9     }
10
11    // Keep track of current wave position
12    double wavePosition = 0;
13    double stepSize = 5;
14    int moveCounter = 0;
15
16    double distanceX = X - left;
17    double distanceY = Y - bottom;
18
19    while (distanceX > -3 &&
20           distanceY > -3 &&
21           distanceX < right - left + 3 &&
22           distanceY < top - bottom + 3 &&
23           moveCounter < maxMoves)
24    {
25        // Calculate the wave angle
26        double waveAngle = Math.Sin(wavePosition / WavePeriod) * WaveAmplitude;
27
28        // Turn to follow the wave pattern
29        SetTurnRight(waveAngle);
30        SetForward(stepSize);
31        Go();
32
33        distanceX = X - left;
34        distanceY = Y - bottom;
35
36        wavePosition += stepSize;
37        moveCounter++;
38    }
39 }
40
41 // Handle out of bounds situations efficiently
42 private void HandleOutOfBounds()
43 {
44     // Stop any current movement
45     Stop();
46
47     double targetDirection = -1;
48
49     // Determine which boundary we've crossed
50     if (X < left && Y < bottom) targetDirection = 45; // Northeast
51     else if (X < left && Y > top) targetDirection = 315; // Southeast
52     else if (X > right && Y < bottom) targetDirection = 135; // Northwest
53     else if (X > right && Y > top) targetDirection = 225; // Southwest
54     else if (X < left) targetDirection = 0; // East
55     else if (X > right) targetDirection = 180; // West
56     else if (Y < bottom) targetDirection = 90; // North
57     else if (Y > top) targetDirection = 270; // South
58
59     if (targetDirection != -1)
60     {
61         double turnAmount = NormalizeRelativeAngle(targetDirection - Direction);
62
63         SetTurnLeft(turnAmount);
64         Go();
65         WaitFor(new TurnCompleteCondition(this));
66
67         double moveDistance = 40;
68         SetForward(moveDistance);
69         Go();
70         WaitFor(new MovementCompleteCondition(this));
71     }
72 }
```

Gambar 15: Implementasi RudalSekelo (Main Bot) - Part 6

```

1 // Handling scanned bots with hybrid targeting strategy
2 public override void OnScannedBot(ScannedBotEvent e)
3 {
4     // Select target greedily: track enemy with lowest energy or current target.
5     if (lastScannedBot == null || e.ScannedBotId == lastScannedBot.ScannedBotId || e.Energy < lastScannedBot.Energy)
6     {
7         lastScannedBot = e;
8     }
9
10    double distance = DistanceTo(e.X, e.Y);
11    double angleToEnemy = Math.Atan((e.Y - Y, e.X - X) * 180 / Math.PI;
12    double gunTurn = NormalizeRelativeAngle(angleToEnemy - GunDirection);
13
14    // Lock radar on target with overcorrection
15    double radarTurn = NormalizeRelativeAngle(angleToEnemy - RadarDirection);
16    SetTurnRadarLeft(radarTurn * 2);
17
18    if (distance > maxShootingDistance)
19    {
20        // Still track with the gun, but don't fire
21        SetTurnGunLeft(gunTurn);
22        return;
23    }
24
25    // Determine fire power based on factors
26    firePower = calculateFirePower(distance);
27    if (Energy > e.Energy + 30) firePower = Math.Min(firePower, Energy / 10);
28    if (Energy < 16)
29    {
30        if (distance < 100) firePower = 3.0;
31        else if (distance < 300) firePower = 2.5;
32    };
33    firePower = Math.Min(firePower, Energy);
34
35    // Calculate bullet speed and time to hit
36    double bulletSpeed = 20 - 3 * firePower;
37    double timeToHit = distance / bulletSpeed;
38
39    // Predict enemy's future position
40    double enemyHeadingRadians = e.Direction * Math.PI / 180;
41    double predictedX = e.X + Math.Cos(enemyHeadingRadians) * e.Speed * timeToHit;
42    double predictedY = e.Y + Math.Sin(enemyHeadingRadians) * e.Speed * timeToHit;
43
44    // Calculate angle to the predicted position
45    double angleToEnemyPredicted = Math.Atan2(predictedY - Y, predictedX - X) * 180 / Math.PI;
46    double gunTurnPredicted = NormalizeRelativeAngle(angleToEnemyPredicted - GunDirection);
47    SetTurnGunLeft(gunTurnPredicted);
48
49    // Calculate aiming confidence
50    double aimConfidence;
51
52    // Very high confidence for close targets
53    if (distance < 50) aimConfidence = 0.95;
54    else
55    {
56        aimConfidence = 1.0;
57        aimConfidence *= 1.0 - (distance / 1000.0); // Distance factor
58        aimConfidence *= 1.0 - (Math.Min(8, e.Speed) / 16.0); // Speed factor
59        aimConfidence *= 1.0 - (Math.Min(45, Math.Abs(gunTurnPredicted)) / 120.0); // Gun turn factor
60    }
61
62    // Determine maximum allowed gun angle error
63    double maxGunAngleError = distance < 50 ? 25 : 15 * aimConfidence;
64
65    // Fire if we're confident enough and gun is cool
66    if (Math.Abs(gunTurnPredicted) < maxGunAngleError && (aimConfidence > 0.3) && GunHeat == 0)
67    {
68        // Close combat keeps full power, longer distances may reduce power slightly
69        double adjustedFirePower = distance < 50 ? firePower : firePower * Math.Max(0.8, aimConfidence);
70        Fire(adjustedFirePower);
71    }
72 }
73
74 // Determine fire power based on factors
75 public double calculateFirePower(double distance)
76 {
77    firePower = 1.0;
78
79    if (distance <= 50) firePower = 3.0; // Close range: maximum power
80    else if (distance <= 150) firePower = 2.5; // Medium-Close: high power
81    else if (distance <= 300) firePower = 2.0; // Medium range: medium power
82    else if (distance <= 450) firePower = 1.5; // Medium-far: moderate power
83
84    return firePower;
}

```

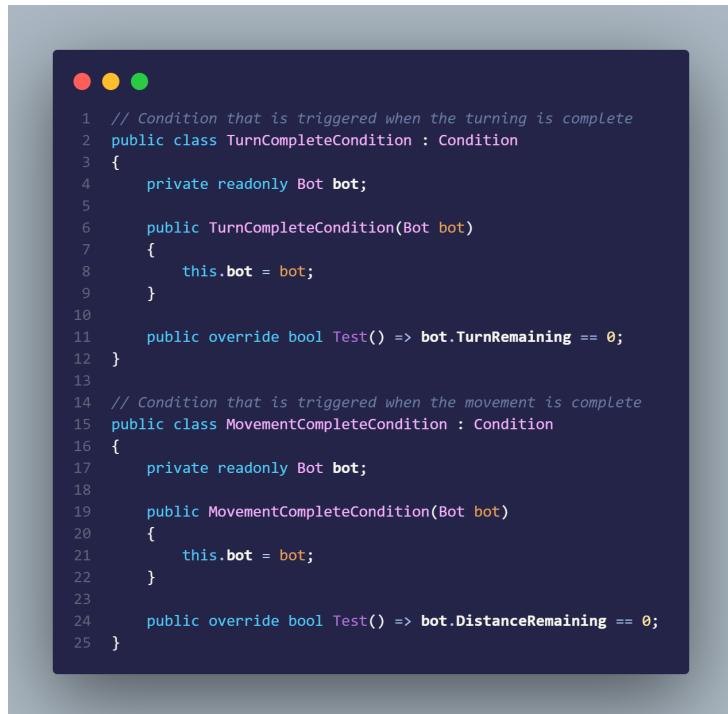
Gambar 16: Implementasi RudalSekelo (Main Bot) - Part 7



The screenshot shows a code editor window with a dark theme. The title bar is visible at the top. The code itself is a C# script for a bot, likely named 'RudalSekeloa'. It handles various events such as being hit by a bullet, hitting other bots, and hitting walls, adjusting movement and fire power accordingly.

```
1 // Dodge when hit by a bullet
2 public override void OnHitByBullet(HitByBulletEvent e)
3 {
4     // Change strafe direction to dodge
5     strafeDirection *= -1;
6 }
7
8 // Improved collision handling with other bots
9 public override void OnHitBot(HitBotEvent e)
10 {
11     double distance = DistanceTo(e.X, e.Y);
12     double bearing = BearingTo(e.X, e.Y);
13
14     // Being rammed by another bot
15     if (e.IsRammed)
16     {
17         Stop();
18
19         // Turn gun toward enemy and fire with power based on distance
20         double gunTurn = NormalizeRelativeAngle(bearing - GunDirection);
21         TurnGunLeft(gunTurn);
22
23         // Fire with power based on distance
24         firePower = calculateFirePower(distance);
25         Fire(firePower);
26     }
27
28     // Rammed into another bot
29     else
30     {
31         Stop();
32
33         double gunTurn = NormalizeRelativeAngle(bearing - GunDirection);
34         TurnGunLeft(gunTurn);
35
36         // Fire with power based on distance
37         firePower = calculateFirePower(distance);
38         Fire(firePower);
39     }
40 }
41
42 // When we hit a wall, adjust course to recover
43 public override void OnHitWall(HitWallEvent e)
44 {
45     if (isDefensiveMode)
46     {
47         Stop();
48
49         bool hitLeftWall = X <= left + 5;
50         bool hitRightWall = X >= right - 5;
51         bool hitBottomWall = Y <= bottom + 5;
52         bool hitTopWall = Y >= top - 5;
53
54         double retreatDirection = Direction;
55
56         if (hitLeftWall) retreatDirection = 0; // Move east
57         else if (hitRightWall) retreatDirection = 180; // Move west
58         else if (hitBottomWall) retreatDirection = 90; // Move north
59         else if (hitTopWall) retreatDirection = 270; // Move south
60
61         double turnAmount = NormalizeRelativeAngle(retreatDirection - Direction);
62
63         SetTurnLeft(turnAmount);
64         Go();
65         WaitFor(new TurnCompleteCondition(this));
66
67         SetForward(50);
68         Go();
69         WaitFor(new MovementCompleteCondition(this));
70
71         GoToNearestMiddleOfASide();
72     }
73     else
74     {
75         // Reverse the strafing direction upon hitting a wall to help avoid getting stuck
76         strafeDirection *= -1;
77         SetTurnRight(90);
78     }
79 }
80 }
```

Gambar 17: Implementasi RudalSekeloa (Main Bot) - Part 8



```

1 // Condition that is triggered when the turning is complete
2 public class TurnCompleteCondition : Condition
3 {
4     private readonly Bot bot;
5
6     public TurnCompleteCondition(Bot bot)
7     {
8         this.bot = bot;
9     }
10
11    public override bool Test() => bot.TurnRemaining == 0;
12 }
13
14 // Condition that is triggered when the movement is complete
15 public class MovementCompleteCondition : Condition
16 {
17     private readonly Bot bot;
18
19     public MovementCompleteCondition(Bot bot)
20     {
21         this.bot = bot;
22     }
23
24    public override bool Test() => bot.DistanceRemaining == 0;
25 }

```

Gambar 18: Implementasi RudalSekeloa (Main Bot) - Part 9

4.3. Pengujian

a. 3 Round Play

Results for 3 rounds												
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds	
1	RudalSekeloa 17.55	710	350	30	307	20	2	0	1	1	1	
2	Sweepredict 4.20	708	300	60	304	34	10	0	2	0	0	
3	Waves 9.11	500	150	0	263	15	72	0	0	2	0	
4	Kaze 9.11	234	100	0	118	0	16	0	0	0	2	

Gambar 19: 3 Round

b. 5 Round Play

Results for 5 rounds												
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds	
1	RudalSekeloa 17.55	1478	500	90	598	81	182	26	4	1	0	
2	Sweepredict 4.20	682	350	30	260	22	19	0	1	2	1	
3	Waves 9.11	552	150	0	281	0	108	13	0	0	4	
4	Kaze 9.11	368	200	0	156	0	12	0	0	2	0	

Gambar 20: 5 Round

c. Default 1st Attempt

Results for 10 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	RudalSekeloa 17.55	1755	600	90	883	143	38	0	4	1	0
2	Waves 9.11	734	300	0	355	0	79	0	0	3	2
3	Sweepredict 4.20	638	200	30	352	36	20	0	1	0	1
4	Kaze 6.9	418	250	0	156	0	12	0	0	1	2

Gambar 21: Default 1st Attempt

d. Default 2nd attempt

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...
1	RudalSekeloa 17.55	1981	700	120	959
2	Waves 9.11	902	400	0	380
3	Sweepredict 4.20	882	400	30	395
4	Kaze 6.9	376	150	0	212

Gambar 22: Default 2nd Attempt

e. Max Turn Count 3 Round

Setup Rules

Game type	classic	Arena Size
Min. Number of Participants	2	Width 800
Max. Number of Participants		Height 600
Number of Rounds	3	
Gun Cooling Rate	0.1	
Max. Inactivity Turns	450	
Ready timeout (μs)	1000000	
Turn timeout (μs)	30000	
Max Turn Count	1000	

OK Cancel Set to Default Apply

Results for 3 rounds

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	RudalSekeloa 17.55	101	0	0	76	0	25	0	1	0	0
2	Sweepredict 4.20	83	0	0	78	0	5	0	0	1	0
3	Waves 9.11	82	0	0	62	0	19	0	0	0	1
4	Kaze 9.11	40	0	0	34	0	6	0	0	0	0

Gambar 23: Max Turn Count 3 Round

f. Max Turn Count 1000

The screenshot shows the configuration interface for a game. On the left, there are several input fields for game parameters:

- Game type: classic
- Min. Number of Participants: 2
- Max. Number of Participants: (empty)
- Number of Rounds: 10
- Gun Cooling Rate: 0.1
- Max. Inactivity Turns: 450
- Ready timeout (μs): 1000000
- Turn timeout (μs): 30000
- Max Turn Count: 1000

On the right, there is a section for Arena Size with fields for Width (800) and Height (600).

Below the configuration is a results table titled "Results for 10 rounds".

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	RudalSekelo 17.55	259	50	0	130	21	58	0	1	0	0
2	Sweepredict 4.20	179	100	0	67	9	2	0	0	1	0
3	Kaze 9.11	161	100	0	60	0	1	0	0	0	1
4	Waves 9.11	74	0	0	32	0	42	0	0	0	0

Gambar 24: Max Turn Count 1000

g. Rudal Sekelo vs Kaze

The screenshot shows the configuration interface for a game. On the left, there are several input fields for game parameters:

- Game type: classic
- Min. Number of Participants: 2
- Max. Number of Participants: (empty)
- Number of Rounds: 10
- Gun Cooling Rate: 0.1
- Max. Inactivity Turns: 450
- Ready timeout (μs): 1000000
- Turn timeout (μs): 30000
- Max Turn Count: 1000

On the right, there is a section for Arena Size with fields for Width (800) and Height (600).

Below the configuration is a results table titled "Results for 10 rounds".

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	RudalSekelo 17.55	860	250	50	483	60	17	0	6	0	0
2	Kaze 9.11	182	0	0	174	0	8	0	0	6	0

Gambar 25: Rudal Sekelo vs Kaze

h. Rudal Sekelo vs Waves

The screenshot shows the configuration interface for a game. On the left, there are several input fields for game parameters:

- Game type: classic
- Min. Number of Participants: 2
- Max. Number of Participants: (empty)
- Number of Rounds: 10
- Gun Cooling Rate: 0.1
- Max. Inactivity Turns: 450
- Ready timeout (μs): 1000000
- Turn timeout (μs): 30000
- Max Turn Count: 1000

On the right, there is a section for Arena Size with fields for Width (800) and Height (600).

Below the configuration is a results table titled "Results for 10 rounds".

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	RudalSekelo 17.55	552	100	20	364	37	30	0	3	2	0
2	Waves 9.11	385	100	20	209	23	32	0	2	3	0

Gambar 26: Rudal Sekelo vs Waves

i. Sweepredict vs Kaze

The screenshot shows the configuration interface for a game. On the left, there are several input fields for game parameters:

- Game type: classic
- Min. Number of Participants: 2
- Max. Number of Participants: (empty)
- Number of Rounds: 10
- Gun Cooling Rate: 0.1
- Max. Inactivity Turns: 450
- Ready timeout (μs): 1000000
- Turn timeout (μs): 30000
- Max Turn Count: 1000

On the right, there is a section for Arena Size with fields for Width (800) and Height (600).

Below the configuration is a results table titled "Results for 10 rounds".

Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Sweepredict 4.20	567	200	40	276	48	2	0	4	0	0
2	Kaze 9.11	69	0	0	64	0	5	0	0	4	0

Gambar 27: Sweepredict vs Kaze

j. Sweepredict vs Rudal Sekelo

Results for 10 rounds											
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	RudalSekelo 17.55	717	150	30	421	54	61	0	4	3	0
2	Sweepredict 4.20	620	150	30	380	53	7	0	3	4	0

Gambar 28: Sweepredict vs Rudal Sekelo

k. Sweepredict vs Waves

Results for 10 rounds											
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Sweepredict 4.20	1073	300	60	609	83	20	0	6	0	0
2	Waves 9.11	325	0	0	231	0	94	0	0	6	0

Gambar 29: Sweepredict vs Waves

l. Waves vs Kaze

Results for 10 rounds											
Rank	Name	Total Score	Survival	Surv. Bonus	Bullet D...	Bullet Bo...	Ram Dmg.	Ram Bonus	1sts	2nds	3rds
1	Waves 9.11	229	100	20	95	13	1	0	2	1	0
2	Kaze 9.11	81	0	0	80	0	1	0	1	2	0

Gambar 30: Boot Directories

4.4. Analisis dan Pembahasan

a. Format 1 vs 1 vs 1 vs 1

Pada mode 1 vs 1 vs 1 vs 1, bot "RudalSekelo" selalu menang dengan skor yang menjanjikan, disusul dengan skor Sweepredict dan Waves serta yang terakhir adalah Kazenemy. Hal ini menunjukkan bahwa strategi *greedy* yang dipilih berhasil untuk mendapatkan nilai optimal secara efisien dan efektif. Pada saat ronde dimulai, RudalSekelo akan melakukan pemindaian dan akan mengidentifikasi apakah ada musuh yang memiliki energi yang paling kecil. Jika ada, maka RudalSekelo akan mengincar bot tersebut hingga hancur atau akan pindah target jika ada bot yang memiliki energi yang lebih kecil dibandingkan dengan target saat ini. Selain itu, RudalSekelo juga memiliki mode ofensif yang akan menembak bot dengan cara menghujaninya dengan peluru yang sesuai dengan perhitungan suatu *confidentiality parameter* dan *prediction* sehingga peluru yang dikeluarkan akan sesuai dengan prinsip efisiensi yang diinginkan. Pada mode defensif (atau ketika darah kurang dari 30 persen), RudalSekelo akan bergerak dengan lebih hati-hati, dan hanya menembak musuh jika tingkat *confidentialitynya* tinggi, jika tidak tinggi, maka RudalSekelo akan lebih memilih untuk "cari aman" agar bisa mendapatkan *survival bonus point*.

b. Format duel (1 vs 1)

Pada mode 1 vs 1, RudalSekelo mendominasi seluruh duel. Hal ini menunjukkan bahwa pendekatan yang diimplementasikan pada RudalSekelo sudah efektif dan efisien. Tetapi, RudalSekelo akan cukup kewalahan dalam melawan Sweepredict karena Sweepredict akan memprediksi Gerakan dari RudalSekelo. Tetapi secara garis besar, RudalSekelo akan menang karena RudalSekelo melakukan *hard locking* pada lawannya, sehingga RudalSekelo bisa fokus untuk berduel.

Bab V

Penutup

5.1. Kesimpulan

Dalam proyek Tugas Besar 1 IF2211 Strategi Algoritma ini, kami berhasil mengimplementasikan algoritma *greedy* pada bot kami untuk permainan Robocode Tank Royale dengan tujuan untuk memenangkan permaianan.

Kami berhasil untuk mengimplementasikan bot Utama "Rudal Sekelo" dengan pendekatan *locking radar* pada musuh dan juga menghitung tingkat *confidentiality* bot terhadap musuhnya. Selain itu, bot Utama kami juga melakukan pendekatan *greedy by distance* agar bisa mengeluarkan peluru secara efisien dan tepat sasaran. Terakhir, kami juga mengimplementasikan mode defensif jika energi bot kurang dari 30 persen untuk mengurangi risiko bot untuk hancur.

Selain itu, kami juga menyediakan 3 bot alternatif dari himpunan solusi *greedy* yang telah kami temukan. Sebagai contoh, ada *greedy by energy*, *greedy by distance*, *greedy by bullet point*, dan *greedy by survival point*. Meskipun ketiga bot tersebut belum bisa menghasilkan solusi yang paling optimum, tetapi ketiga bot tersebut bisa digunakan untuk menjawab permasalahan yang ada dari abstraksi pada game ini, dengan cara "take what you can take now!".

5.2. Saran

Pelaksanaan Tugas Besar 1 IF2211 Strategi Algoritma di Semester II (Genap) Tahun 2024/2025 merupakan pengalaman yang sangat berharga bagi kami. Dari pengalaman ini, kami ingin berbagi beberapa saran kepada pembaca yang mungkin akan menghadapi tugas serupa di masa depan:

Razi

Untuk spesifikasinya mungkin diperjelas lagi batasan perbedaan dari pendekatan heuristiknya

Nayaka

Aku suka tembak-tembakan, dar der dor!, tubesnya cukup asik, yayy

Adha

Keep cooking!

Semoga saran-saran ini membantu pembaca dalam menyiapkan diri untuk menangani tugas serupa di masa depan.

5.3. Refleksi

Ruang perbaikan dan pengembangan dalam mengerjakan tugas dapat difokuskan pada beberapa aspek yang dapat ditingkatkan. Pertama-tama, pengaturan waktu menjadi kunci utama. Kami menyadari bahwa perencanaan waktu yang lebih baik dapat meningkatkan efisiensi pengerjaan. Menerapkan strategi manajemen waktu, seperti membuat jadwal yang terstruktur dan menetapkan tenggat waktu internal untuk setiap tahap pekerjaan, dapat membantu menghindari tekanan waktu yang tidak perlu.

Selain itu, ketelitian membaca spesifikasi dari awal menjadi aspek yang perlu diperhatikan. Memahami secara menyeluruh tentang apa yang diminta dalam tugas, termasuk detail-detail kecil, dapat mengurangi risiko kesalahan dan memastikan pekerjaan berjalan sesuai dengan harapan. Menempatkan perhatian ekstra pada spesifikasi tugas dapat meminimalkan revisi dan penyesuaian yang mungkin diperlukan.

Komunikasi tim yang baik juga dapat dioptimalkan. Membuat saluran komunikasi yang jelas dan terbuka di antara anggota tim dapat menghindari kebingungan dan memastikan bahwa semua anggota memiliki pemahaman yang sama tentang tujuan dan tanggung jawab masing-masing. Diskusi reguler dan pembahasan mengenai kemajuan proyek dapat meningkatkan keterlibatan semua anggota tim.

Terakhir, evaluasi diri secara berkala dapat menjadi langkah penting. Menerima umpan balik, baik dari anggota tim maupun asisten, dan memanfaatkannya sebagai dasar untuk perbaikan lebih lanjut adalah cara efektif untuk terus berkembang. Selalu terbuka terhadap saran dan berkomitmen untuk belajar dari setiap pengalaman dapat membantu memperbaiki kinerja secara berkelanjutan.

Lampiran

6.1. Tautan

Repository program dapat diakses melalui tautan berikut:

https://github.com/zirachw/Tubes1_Rudal-Sekeloa-Reloaded

Video penjelasan program dapat diakses melalui tautan berikut:

<http://linktr.ee/Zirach>

6.2. Tabel Spesifikasi

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.	✓	

Referensi

Rinaldi Munir. 2025. “Algoritma Greedy (Bagian 1) (Versi baru 2025).” Diakses pada 3 Maret 2025. [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)

Corporate Megaxus. 2024. “Pengertian Game Engine: Sejarah, Fungsi dan Jenisnya.” Diakses pada 3 Maret 2025. <https://corporate.megaxus.com/id/game-engine>

Amazon AWS. 2024. “What is bot?” Diakses pada 3 Maret 2025. <https://aws.amazon.com/id/what-is/bot/>

Robocode Dev. 2025. “Robocode Tank Royale Docs” Diakses pada 3 Maret 2025. <https:////robocode-dev.github.io/tank-royale/>