IMPLEMENTASI ALGORITMA GREEDY DALAM PEMECAHAN RUTE TERBAIK PADA PERMAINAN ETIMO *DIAMONDS* 2

TUGAS BESAR

Diajukan sebagai syarat menyelesaikan mata kuliah Strategi Algoritma (IF2211) Kelas RC di Program Studi Teknik Informatika, Fakultas Teknologi Industri, Institut Teknologi Sumatera

Dipersiapkan oleh:

Habbi Widagdo	(123140204)
Martino Kelvin	(123140165)
Mohd.Musyaffa Alief Athallah	(123140184)

Kelompok 01 – Brain Not Found

Dosen Pengampu: Winda Yulita, M.Cs.

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA
2025

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Diamonds adalah tantangan pemrograman yang menantang para peserta untuk bertanding menggunakan bot yang mereka kembangkan melawan bot peserta lain. Setiap peserta akan memiliki bot dengan tujuan utama mengumpulkan sebanyak mungkin diamond. Namun, tantangan ini tidak akan mudah karena berbagai hambatan akan menambah keseruan dan kerumitan dalam permainan. Untuk memenangkan kompetisi, peserta harus menerapkan strategi khusus pada bot mereka masing-masing. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah.

Gambar 1.1 Tampilan Frontend Diamonds

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

Program permainan *Diamonds* terdiri atas:

1. Game engine, yang secara umum berisi:
    a. Kode backend permainan, yang berisi logic permainan secara keseluruhan serta API yang disediakan untuk berkomunikasi dengan frontend dan program bot
    b. Kode frontend permainan, yang berfungsi untuk memvisualisasikan permainan
2. Game engine, yang secara umum berisi:
    a. Program untuk memanggil API yang tersedia pada backend
    b. Program bot logic (bagian ini yang akan kalian implementasikan dengan algoritma greedy untuk bot kelompok kalian)
    c. Program utama (main) dan utilitas lainnya

Komponen-komponen dari permainan Diamonds antara lain:

1. Diamonds

    Untuk memenangkan pertandingan, kita harus mengumpulkan diamond ini sebanyak-banyaknya dengan melewati/melangkahinya. Terdapat 2 jenis diamond yaitu

diamond biru dan diamond merah. Diamond merah bernilai 2 poin, sedangkan yang biru bernilai 1 poin. Diamond akan di-regenerate secara berkala dan rasio antara diamond merah dan biru ini akan berubah setiap regeneration.

Gambar 1. 2 Diamond Biru dan Merah

2. Red Button/Diamond Button

   Ketika red button ini dilewati/dilangkahi, semua diamond (termasuk red diamond) akan di-generate kembali pada board dengan posisi acak. Posisi red button ini juga akan berubah secara acak jika red button ini dilangkahi.

Gambar 1. 3 Red Button

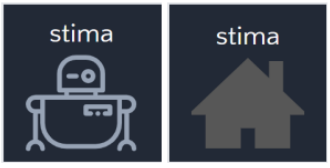
3. Teleporters

   Terdapat 2 teleporter yang saling terhubung satu sama lain. Jika bot melewati sebuah teleporter maka bot akan berpindah menuju posisi teleporter yang lain.

Gambar 1. 4 Teleporters

4. Bots dan Base

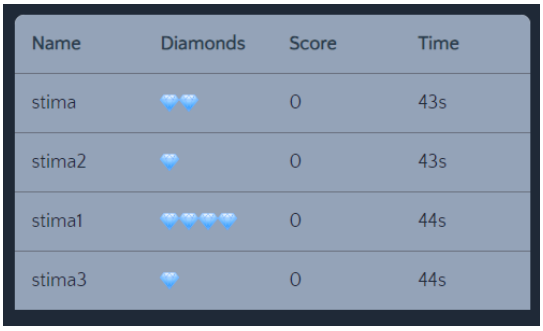
   Pada game ini kita akan menggerakkan bot untuk mendapatkan diamond sebanyak banyaknya. Semua bot memiliki sebuah Base dimana Base ini akan digunakan untuk menyimpan diamond yang sedang dibawa. Apabila diamond disimpan ke base, score bot akan bertambah senilai diamond yang dibawa dan inventory (akan dijelaskan di bawah) bot menjadi kosong.



Gambar 1. 5 Bots dan Base

5. Inventory

   Bot memiliki inventory yang berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini memiliki kapasitas maksimum sehingga sewaktu waktu bisa penuh. Agar inventory ini tidak penuh, bot bisa menyimpan isi inventory ke base agar inventory bisa kosong kembali.



Gambar 1. 6 Inventoy

Untuk mengetahui flow dari game ini, berikut ini adalah cara kerja permainan Diamonds.

1. Pertama, setiap pemain (bot) akan ditempatkan pada board secara random. Masing-masing bot akan mempunyai home base, serta memiliki score dan inventory awal bernilai nol.
2. Setiap bot diberikan waktu untuk bergerak, waktu yang diberikan semua sama untuk setiap pemain.
3. Objektif utama bot adalah mengambil diamond-diamond yang ada di peta sebanyak-banyaknya. Seperti yang sudah disebutkan di atas, diamond yang berwarna merah memiliki 2 poin dan diamond yang berwarna biru memiliki 1 poin.
4. Setiap bot juga memiliki sebuah inventory, dimana inventory berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini sewaktu-waktu bisa penuh, maka dari itu bot harus segera kembali ke home base.
5. Apabila bot menuju ke posisi home base, score bot akan bertambah senilai diamond yang tersimpan pada inventory dan inventory bot akan menjadi kosong kembali.
6. Usahakan agar bot anda tidak bertemu dengan bot lawan. Jika bot A menimpa posisi bot B, bot B akan dikirim ke home base dan semua diamond pada inventory bot B akan hilang, diambil masuk ke inventory bot A (istilahnya tackle).
7. Selain itu, terdapat beberapa fitur tambahan seperti teleporter dan red button yang dapat digunakan apabila anda menuju posisi objek tersebut.
8. Apabila waktu seluruh bot telah berakhir, maka permainan berakhir. Score masing-masing pemain akan ditampilkan pada tabel Final Score di sisi kanan layar.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma greedy sering kita kenal sebagai algoritma yang dapat memecahkan persoalan secara langkah per langkah (step by step) yang sedemikian sehingga, pada setiap langkahnya nanti:

- Menemukan pilihan terbaik yang dapat diperoleh pada saat itu tanpa perlu memperhatikan akibat kedepannya (prinsip"*take what you can get now!*")
- Berharap nantinya dapat terpilih solusi optimum lokal pada setiap langkah sehingga di akhir akan menghasilkan suatu solusi optimum global.

Terdapat beberapa eleman penting yang perlu diketahui dalam proses menjalankan algoritma greedy guna proses pemeceahan masalah secara greedy dapat dilakukan dengan lebih mudah. Elemen-elemen tersebut adalah sebagai berikut:

1. Himpunan kandidat (C), berisi sekumpulan kandidat yang akan dipilih untuk setiap langkah. Contohnya itu ada simpul atau sisi di dalam graf, koin, task, job, karakter dan benda.
2. Himpunan solusi (S), berisi sekumpulan kandidat yang telah dijadikan solusi.
3. Fungsi solusi, diperuntukkan dalam proses penentuan apakah suatu himpunan kandidat itu dapat dipilih telah memberikan solusi ataupun belum.
4. Fungsi seleksi (*selection function*), diperuntukkan dalam memilih kandidat berdasar pada strategi greedy tertentu. Diketahui bahwa strategi greedy ini bersifat heuristik yang berarti dapat berbeda untuk tiap-tiap masalah yang nanti akan dihadapi.
5. Fungsi kelayakan (*feasible*), diperuntukkan dalam memeriksa apakah kandidat yang telah dipilih dapat kita golongkan ke dalam himpunan solusi ataupun tidak.
6. Fungsi objektif, berguna dalam usaha memaksimumkan atau meminimumkan kandidat yang dipilih.

Algoritma greedy ini akan melakukan pencarian suatu himpunan bagain (S) dari himpunan kandidat (C) yang dalam hal ini, himpunan S harulah dapat memenuhi syarat yang telah ditentukan, yaitu S harus menyatakan suatu solusi dan S dioptimisasi oleh fungsi objektif. Algoritma ini juga akan menghasilkan serangkaian solusi optimum lokal dan dari serangkaian solusi lokal tersebut nanti akan didapati solusi terbaik yang dinami solusi optimum global.

Namun, solusi yang dihasilkan oleh algoritma greedy tidak selalu merupakan solusi yang terbaik, hal tersebut disebabkan oleh sangat mungkinnya solusi yang dihasilkan tersebut merupakan solusi sub-optimum ataupun pseudo-optimum. Terdapat dua alasan yang kuat kenapa hal-hal tersebut dapat terjadi, yaitu:

- Algoritma ini tidak dapat bekerja secara menyeluruh terhadap semua kemungkinan solusi yang ada (seperti halnya pada metode exhaustive search)
- Ditemukannya beberapa fungsi seleksi yang berbeda, sehingga diperlukan langkah selanjutnya untuk bisa menentukan fungsi yang tepat jika algoritma diharapkan untuk menghasilkan solusi yang optimal.

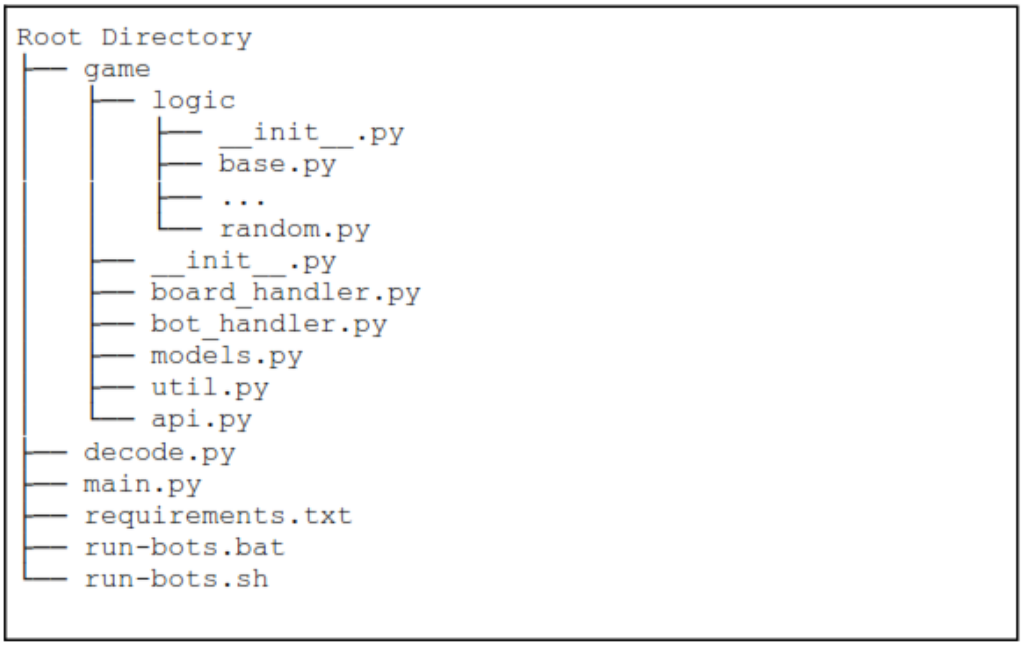
Dalam penerapannya, terdapat banyak persamalahan yang dapat diatasi dengan pendekatan greedy, yaitu contohnya sebagai berikut:

1. Persoalan memilih aktivitas
2. Persoalan penukaran uang
3. Minimalisai waktu di dalam sistem
4. Persoalan knapsack
5. Penjadwalan job dengan tenggat waktu
6. Lintasan terpendek
7. Pohon merentah minimum
8. Pecahan Mesir
9. Kode Huffman

2.2 Cara Kerja Program

2.2.1 Struktur Direktori Program

Program bot pada pada tugas besar kali ini mempunyai struktur direktoru sebagai berikut.



Nantinya, proses pembuatan algoritma bot hanya akan dilakukan pada folder */game/logic*. Untuk semua *file* yang berda di luar folder *logic* hanya akan berfungsi sebagai komponen-komponen pendukung untuk dapat menjalankan bot dan untuk berkomunikasi dengan website dari *game diamond* itu sendiri.

2.2.2 Proses Pelaksanaan Aksi oleh Bot

Peramainan dalam tugas besat kali ini merupakan suatu permainan yang berbasis web, sehigga setiap aksi yang akan dilakukan mulai dari tambah bot baru sampai menjalankan aksi bot perlu adanya HTTP *request* terhadao API *endpoint* khusus yang disediakan oleh bagian *backend.* Berikut ini adalah urutan dari request yang terjadi dari awal hingga akhir permainan.

a. Program akan lakukan mengecek apakah bot telah didaftarkan atau belum dengan mengirimkan POST *request* terhadap *endpoint* */api/bots/recover* dengan body berisikan email dan password bot. Jika bot telah terdaftar, maka nantinya bagian *backaend* akan memberi *response code* 200 dengan body berisikan id dari bot tersebut dan jita tidak maka *backend* akan memberikan *response code* 404.
b. Misalkan bot belum terdaftar, maka nantinya program akan mengirim POST *request* terhadap *endpoint* */api/bots/* dengan body berisikan email, name, password, dan team. Jika berhasil, maka backend akan memberikan response code 200 dengan body yang berisi id dari bot tersebut.
c. Disaat id bot telah diketahui, bot akan dapat bergabung pada *board* dengan mengirim POST *request* terhadap endpoint */api/bots/{id}/join* dengan body yang berisi *board id* yang diinginkan (*preferredBoardId*). Jika bot berhasil untuk join, maka *backend* akan memberikan *response code* 200 dengan body yang berisi informasi dari *board*.
d. Selanjutnya program bot akan mengkalkulasikan pergerakan secara berkalan berdasar pada kondisi *board* yang diketahui dan selanjutnya akan mengirimkan POST *request* terhadap *endpoint* */api/bots/{id}/move* dengan body yang berisi direksi yang akan dilalui selanjutnya ("NORTH", "SOUTH", "EAST", atau "WEST"), Apabila berhasil, maka nantinya *backend* akan memberi *response code* 200 dengan body berisikan kondisi *board* setelah pergerakan tersebut. Langkah semacam ini diberlakukan terus menerus sampai waktu bot habis, jika waktu habis, bot akan secara otomatis akan dikeluarkan dari *board*.
e. Bagian program *frontend* secara periodik juga mengirimkan GET *request* terhadap *endpoint* */api/boards/{id}* agar mendapatkan kondisi *board* terbaru, sehingga tampilan *board* pada *frontend* agar selalu diperbaharui.

2.2.3 Implementasi Algoritma Greedy pada Bot

Sebelum kita dapat melakukan implementasi algoritma yang telah dibuat ke dalam bot, perlu pembuatan sebuah file.py baru pada direktoru */game/logic*, misalnya kita namai *MyBot.py.* Seterusnya akan dibuat kelas barus yang meng-inherit ke kelas *BaseLogic,* lalu implementasikan constuctur dan method next_move pada kelas tersebut. Berikut ini merupakan contoh dari pembuatan kelas.

```
from game.logic.base import BaseLogic
from game.models import Board, GameObject

class MyBot(BaseLogic):
    def _init_(self):
        # Initialize attributes necessary
        self.my_attribute = 0
    def next_moveself, board_bot: GameObject, board: Board):
        # Calculate next move
        delta_x = 1
        delta_y = 0

        return delta_x, delta_y
```

Terdapat fungsi *next_move* pada gambar diatas yang akan mengembalikan nilai delta_x dan delta_y, untuk nilai yang diperbolehkan hanyalah (1,0), (0,1), (-

1,0), (0,-1). Jika nilai yang dihasilkan ilegal atau di luar *range board*, sehingga move akan dibaikan oleh program lalu akan memunculkan *error invalid move*.

Untuk langkah selanjutnya adalah melakukan import kelas yang sudah dibuat di main.py dan mendaftarkannya pda dictionary CONTOROLLERS. Berikut merupakan contoh dari prosesnya.

```
from game.logic.mybot import MyBot

init()
BASE_URL = "http://localhost:3000/api"
DEFAULT_BOARD_ID = 1
CONTROLLERS = {"Random": RandomLogic, "MyBot": MyBot}
```

2.2.4 Menjalankan Program Bot

Setelah program bot berhasil dibuat, selajutnya kita dapat menjalankan program tersebut melalui aplikasi cmd atau terminal. Untuk running satu bot dengan logic yang ada pada file *game/logic/random.py* diperlukan perintah seperti di bawah ini.

```
python main.py --logic Random --email=your_email@example.com --name=your_name
--password=your_password --team etimo
```

Jika ingin menjalankan beberapa bot sekaligus dalam waktu yang bersamaan, maka kita perlu membuat file berama run-bots.bat pada windows atau run-bots.sh pada Linux dan MacOS yang nantinya file terssebut akan berisi perintah-perintah seperti pada contoh berikut.

```
@echo off
start cmd /c "python main.py --logic Random --email=your_email@example.com
--name=your_name --password=your_password --team etimo"
start cmd /c "python main.py --logic Random --email=your_email@example.com
--name=your_name --password=your_password --team etimo"
start cmd /c "python main.py --logic Random --email=your_email@example.com
--name=your_name --password=your_password --team etimo"
start cmd /c "python main.py --logic Random --email=your_email@example.com
--name=your_name --password=your_password --team etimo"
```

Catatan: untuk argumen logic pada *command/script* unutk run satu bot saja perlu adanya penyesuain untuk nama bot yang telah didaftarkan pada CONTROLLERS dan untuk *script* yang ada pada *run-bots.bat* atau *run-bots.sh* juga perlu penyesuaian dari segi email, nama ataupun password yang bersifat unik.

BAB III
APLIKASI STRATEGI GREEDY

3.1 Proses Mapping

Landasan teori dari studi pustaka menyebutkan bahwa algoritma greedy melibatkan pencarian himpunan bagian solusi (S) dari himpunan kandidat (C). Himpunan S ini harus valid (memenuhi kriteria) dan optimal menurut fungsi objektif. Untuk mengaplikasikan algoritma greedy pada game Diamonds, maka elemen-elemen game perlu dipetakan terlebih dahulu menjadi elemen-elemen khas algoritma greedy, sebagai berikut.

1. Himpunan Kandidat (C)

Himpunan kandidat untuk masalah ini berisi koordinat dari objek-objek dalam game, yaitu blue diamond, red diamond, red button, teleporter, base bot, bot musuh, dan base musuh. Aspek penting lainnya yang harus diperhatikan dalam game ini adalah jarak antara bot dan koordinat destinasinya.

2. Himpunan Solusi (S)

Himpunan solusi mencakup koordinat-koordinat yang menjanjikan perolehan keuntungan paling banyak ketika didatangi oleh bot. Sesuai dengan metode greedy yang digunakan, himpunan ini bisa saja hanya berisi satu koordinat tunggal atau sebuah lintasan yang terdiri dari beberapa koordinat berurutan.

3. Fungsi Solusi

Fungsi solusi memiliki peran penting dalam menentukan apakah kandidat yang dipilih telah secara definitif membentuk sebuah solusi. Pada umumnya, aktivasi fungsi ini terjadi hanya pada saat permainan mendekati akhir. Dalam operasinya, fungsi ini akan melakukan kalkulasi untuk menilai kecukupan sisa waktu yang tersedia untuk perjalanan menuju suatu koordinat spesifik di dalam peta, serta untuk perjalanan kembali ke base. Alasan utama mengapa fungsi ini diimplementasikan secara eksklusif pada fase akhir permainan adalah karena mekanisme pada fungsi seleksi yang bertugas menentukan kandidat solusi diasumsikan telah terlebih dahulu mengeliminasi kandidat-kandidat yang tidak memenuhi kriteria sebagai solusi.

4. Fungsi Seleksi (Selection)

Fungsi seleksi berguna untuk memilih kandidat berdasarkan strategi greedy tertentu. Fungsi seleksi akan dijelaskan lebih lanjut pada bagian alternatif solusi karena pendekatan greedy yang berbeda akan memiliki fungsi seleksi yang berbeda.

5. Fungsi Kelayakan (Feasible)

Fungsi kelayakan memiliki peran untuk memverifikasi apakah suatu kandidat yang dipilih memenuhi syarat untuk diintegrasikan ke dalam himpunan solusi. Dalam konteks permainan Diamonds, fungsi ini diaplikasikan untuk menentukan kelayakan suatu objek dalam game sebagai destinasi berikutnya bagi bot. Sebagai ilustrasi, apabila red diamond menjadi target selanjutnya namun inventaris bot sudah berisi 4 diamond (sehingga red diamond tersebut tidak dapat lagi diambil), fungsi kelayakan akan mencegah bot bergerak

menuju koordinat objek tersebut. Tindakan preventif ini bertujuan untuk menghindari potensi terjadinya bug, seperti bot yang bergerak berputar-putar tanpa henti di sekitar red diamond atau bahkan melakukan gerakan yang tidak valid (invalid move).

6. Fungsi Objektif

Fungsi objektif digunakan untuk memaksimalkan atau meminimumkan dari himpunan kandidat menjadi solusi. Dalam permasalahan ini himpunan kandidat akan melalui fungsi objektif untuk ditentukan mana kandidat yang memiliki keuntungan terbesar. Jadi, fungsi objektif akan diterapkan untuk memaksimalkan keuntungan yang didapat dari himpunan kandidat.

Keuntungan dalam fungsi objektif akan sangat berkaitan dengan jarak antara bot dengan koordinat tujuan. Semakin dekat jaraknya dan semakin tinggi poin dari objek tujuan, maka keuntungannya akan semakin besar. Karena bot hanya bisa bergerak ke atas, bawah, kanan, atau kiri, maka jarak antara bot dengan koordinat tujuan akan dihitung menggunakan Manhattan Distance. Misalkan posisi bot saat ini adalah (x_1, y_1) koordinat tujuan adalah (x_2, y_2) dan , maka Manhattan Distance akan dihitung melalui persamaan sebagai berikut.

$$Distance = |x_1 - x_2| + |y_1 - y_2|$$

Jarak yang telah didapatkan hanyalah salah satu komponen untuk menghitung keuntungan dari gerakan yang dilakukan oleh bot. Namun, secara sederhana keuntungan yang didapatkan dapat dihitung menggunakan rumus:

$$\boldsymbol{Keuntungan = \frac{Poin\ Objek}{Jarak}}$$

Namun, untuk setiap alternatif solusi akan memiliki fungsi objektif yang berbeda. Fungsi objektif untuk setiap alternatif solusi akan dijelaskan lebih lanjut pada bagian berikutnya.

3.2 Eksplorasi Alternatif Solusi Greedy

3.2.1 Algoritma Greedy Tackle

Ide dari algoritma ini muncul karena adanya fitur tackle, yakni kemampuan bot untuk bergerak ke posisi musuh dan mencuri diamond yang dibawanya. Dalam implementasi ini, bot memprioritaskan mengejar musuh yang membawa setidaknya satu diamond, dan melakukan tackle jika memungkinkan. Saat tackle berhasil, bot akan memperoleh diamond dari musuh hingga inventarisnya penuh, sementara musuh akan dipaksa kembali ke base. Strategi ini sangat menguntungkan karena memungkinkan pengambilan hingga lima diamond hanya dengan satu langkah, serta memperlambat kemajuan musuh. Oleh karena itu, pendekatan ini didasarkan pada prinsip greedy, di mana bot lebih memilih mencuri hasil kerja musuh ketimbang mengumpulkan diamond sendiri. Bot baru akan kembali ke base jika jumlah diamond yang dibawa mencapai ambang batas tertentu, dan jika tidak ada musuh yang layak ditargetkan, bot akan berpindah secara teratur dalam pola roaming untuk mencari peluang tackle berikutnya.

3.2.1.1 Himpunan Kandidat

Himpunan kandidat dalam algoritma ini adalah daftar semua bot musuh yang terdapat di papan permainan dan memiliki properti posisi serta jumlah diamond. Kandidat ini dikumpulkan dalam list self.enemies, yang diisi setiap kali fungsi _update_internal_state dijalankan. Bot-bot ini menjadi kemungkinan target yang bisa ditackle, dan dari sinilah algoritma memilih satu yang paling sesuai.

3.2.1.2 Himpunan Solusi

Himpunan solusi merupakan urutan langkah bot yang mengarah pada tujuan strategis tertentu. Dalam konteks kode ini, solusi mencakup dua strategi utama: (1) bergerak menuju base jika membawa cukup banyak diamond, atau (2) bergerak menuju musuh yang layak untuk ditackle. Jika tidak ada target yang sesuai, maka solusi fallback adalah melakukan roaming untuk mencari peluang baru.

3.2.1.3 Fungsi Solusi

Fungsi solusi ditentukan dalam next_move, yaitu penentuan arah gerak (delta_x, delta_y) berdasarkan posisi tujuan saat ini (final_goal_position). Jika tujuan adalah base, maka gerak diarahkan ke base; jika tujuan adalah musuh, maka gerak diarahkan ke posisi musuh. Jika tidak ada tujuan valid, maka bot bergerak mengikuti pola arah tertentu dalam daftar self.directions.

3.2.1.4 Fungsi Seleksi

Fungsi seleksi dilakukan oleh fungsi _find_target_enemy(), yang memilih musuh paling dekat dari himpunan kandidat berdasarkan syarat jumlah diamond minimal (MIN_DIAMONDS_FOR_PRIORITY_TARGET). Jika tidak ada musuh yang memenuhi syarat ini, maka seluruh musuh yang valid menjadi kandidat, dan dari situ dipilih musuh terdekat berdasarkan jarak Manhattan.

3.2.1.5 Fungsi Kelayakan

Fungsi kelayakan memastikan bahwa hanya musuh yang memiliki posisi dan properti valid, serta jumlah diamond yang cukup, yang dapat dipertimbangkan sebagai target. Kriteria ini diperiksa saat menyaring self.enemies, dan juga saat menentukan apakah self.targeted_enemy masih valid untuk dikejar atau perlu direset.

3.2.1.5 Fungsi Objektif

Fungsi objektif dari algoritma ini adalah memaksimalkan jumlah diamond yang dikumpulkan dengan jumlah langkah minimal, sambil menghambat laju musuh. Ini dicapai dengan mengambil tindakan tackle yang dapat mencuri banyak diamond dalam satu gerakan, serta mengirim musuh kembali ke base, yang secara tidak langsung memperlambat akumulasi diamond oleh lawan.

3.2.2 Algoritma Greedy Direct

Ide dari algoritma ini muncul dari pengamatan terhadap pola pergerakan musuh dan distribusi diamond di peta. Dalam pendekatan ini, bot difokuskan untuk mengoptimalkan efisiensi pengumpulan diamond dengan meminimalkan waktu tempuh dan memaksimalkan hasil pergerakan. Setiap langkah bot diarahkan berdasarkan evaluasi posisi diamond terdekat, kapasitas inventaris saat ini, dan kondisi sekitar seperti keberadaan musuh atau penghalang. Bot akan memprioritaskan pengambilan diamond yang paling mudah diakses dalam radius tertentu, lalu segera kembali ke base setelah kapasitas penuh atau jika situasi di medan menjadi tidak menguntungkan, seperti ancaman tackle dari musuh. Strategi ini menggunakan prinsip greedy yang berfokus pada pencapaian keuntungan lokal terbesar di setiap langkah, yakni satu unit diamond yang mudah dijangkau, untuk mencapai tujuan global berupa total diamond maksimal dalam batas waktu permainan. Jika tidak tersedia diamond yang mudah dijangkau, bot akan melakukan eksplorasi terarah (roaming) untuk memperluas cakupan pengamatan dan menemukan target baru secara efisien.

3.2.2.1 Himpunan Kandidat

Himpunan kandidat dalam logika bot ini adalah serangkaian kemungkinan target atau tujuan yang dapat dipilih oleh bot pada setiap langkah pengambilan keputusan. Kandidat-kandidat ini meliputi:

- Posisi berlian-berlian (self.diamonds): Setiap berlian di papan permainan adalah kandidat target untuk dikumpulkan.
- Posisi tombol berlian merah (self.redButton): Tombol ini adalah kandidat yang bisa ditekan untuk menghasilkan lebih banyak berlian.
- Posisi teleporter (self.teleporter): Teleporter adalah kandidat sebagai perantara untuk mencapai target lain (berlian atau markas) dengan lebih cepat. Jadi, posisi masuk teleporter adalah kandidat.
- Posisi markas sendiri (self.board_bot.properties.base): Markas menjadi kandidat tujuan utama ketika bot membawa cukup berlian atau dalam kondisi tertentu (misalnya, waktu hampir habis atau ada musuh di dekat).
- Posisi sementara untuk menghindari rintangan (self.static_temp_goals): Ketika jalur ke tujuan utama terhalang (misalnya oleh teleporter yang tidak diinginkan atau berlian merah saat membawa 4 berlian), bot menghasilkan posisi sementara sebagai kandidat tujuan jangka pendek untuk menghindari rintangan tersebut.
- Arah-arah roaming (self.directions): Jika tidak ada target spesifik yang lebih prioritas, kandidat pergerakan adalah salah satu dari empat arah mata angin untuk menjelajah.

Pada setiap siklus next_move, bot mengevaluasi berbagai kondisi untuk mempersempit himpunan kandidat ini menjadi satu tujuan yang akan dikejar.

3.2.2.2 Himpunan Solusi

Himpunan solusi dalam konteks satu pemanggilan next_move adalah keputusan pergerakan tunggal (delta_x, delta_y) yang dihasilkan. Namun, jika dilihat dari perspektif yang lebih luas, himpunan solusi yang sedang dibangun oleh bot secara iteratif adalah urutan target-target yang telah atau akan dikunjungi (disimpan dalam self.static_goals dan self.goal_position). Misalnya, jika bot memutuskan untuk mengambil berlian melalui teleporter, self.static_goals bisa berisi [posisi_teleporter_masuk, posisi_berlian_target], dan self.goal_position akan menjadi elemen pertama dari daftar ini. Setiap kali bot mencapai sebuah goal_position (yang bukan markas), ia memilih kandidat berikutnya untuk ditambahkan ke "solusi" perjalanannya. Keputusan akhir (delta_x, delta_y) adalah langkah konkret untuk merealisasikan bagian dari solusi yang sedang dibangun.

3.2.2.3 Fungsi Solusi

Fungsi solusi menentukan apakah himpunan solusi saat ini (S) sudah merupakan solusi lengkap untuk sub-masalah yang sedang dihadapi.

1. **Untuk sub-masalah mencapai target (self.goal_position)**: Solusi tercapai ketika self.board_bot.position == self.goal_position. Pada titik ini, jika goal_position adalah berlian, berlian tersebut dikumpulkan. Jika itu adalah teleporter masuk, bot diteleportasi. Jika itu markas, berlian disetor.

2. **Untuk sub-masalah pengumpulan berlian**: Secara implisit, solusi tercapai untuk satu "sesi" pengumpulan ketika bot berhasil kembali ke markas setelah mengumpulkan berlian. Kondisi seperti props.diamonds == 5 atau kondisi waktu hampir habis memicu penyelesaian "sesi" ini.

3. **Untuk keseluruhan permainan**: Fungsi solusi global (yang tidak secara eksplisit ada dalam satu next_move) adalah ketika permainan berakhir, dan tujuannya adalah memaksimalkan skor total. Logika dalam next_move membuat keputusan lokal yang diharapkan berkontribusi pada solusi global ini.

Ketika delta_x == 0 and delta_y == 0 dan ini bukan karena bot berada di markas atau menunggu di *camp spot* (konsep dari TackleLogic, tidak ada di sini), kode ini menganggap ada sesuatu yang salah atau tujuan saat ini sudah "tercapai" tanpa progres lebih lanjut, sehingga mereset goal dan memanggil next_move lagi. Ini adalah cara untuk memastikan bot tidak terjebak.

3.2.2.4 Fungsi Seleksi

Fungsi seleksi adalah inti dari pendekatan greedy, di mana bot memilih kandidat terbaik dari himpunan kandidat (C) pada setiap langkah.

Dalam kode ini, fungsi seleksi diimplementasikan melalui beberapa mekanisme:

1. **Prioritas Kembali ke Markas**: Logika should_go_to_base adalah fungsi seleksi dengan prioritas tertinggi. Jika kondisi seperti props.diamonds == 5, waktu hampir habis (props.milliseconds_left < 5000 and props.diamonds >= 2), atau ada musuh dekat saat membawa banyak berlian (enemy_is_near_and_risky), atau dekat markas dengan cukup berlian (self.calculate_near_base() and props.diamonds > 2), maka markas (self.find_best_way_to_base()) dipilih sebagai self.goal_position.
2. **Pemilihan Target Berlian/Tombol (find_nearest_diamond)**: Jika tidak kembali ke markas, fungsi ini memilih target berikutnya. Ia membandingkan:
    - Jarak langsung ke berlian terdekat (dinormalisasi dengan poin berlian).
    - Jarak ke berlian terdekat melalui teleporter (dinormalisasi dengan poin berlian).
    - Jarak ke tombol merah terdekat.
    Bot memilih opsi dengan "biaya" (jarak atau jarak/poin) terkecil. Ini adalah pilihan greedy lokal.
3. **Pemilihan Jalur ke Markas (find_best_way_to_base)**: Fungsi ini memilih antara jalur langsung ke markas atau jalur melalui teleporter, berdasarkan mana yang memiliki total jarak Manhattan lebih pendek.
4. **Pemilihan Tujuan Sementara untuk Menghindar (obstacle_on_path)**: Jika jalur ke self.goal_position yang sudah dipilih terhalang oleh teleporter yang tidak diinginkan (jika tidak bertujuan ke teleporter itu) atau berlian merah (jika membawa 4 berlian), fungsi ini akan menyeleksi posisi menghindar (Position(dest_y,dest_x-1), dll.) sebagai self.static_temp_goals, yang kemudian akan menjadi self.goal_position untuk sementara.
5. **Pemilihan Arah Roaming**: Jika tidak ada self.goal_position yang ditentukan oleh logika di atas, bot akan memilih arah berikutnya dari self.directions secara berurutan.

3.2.2.5 Fungsi Kelayakan

Fungsi kelayakan digunakan untuk menentukan apakah sebuah kandidat yang dipilih oleh fungsi seleksi dapat ditambahkan ke himpunan solusi saat ini tanpa melanggar batasan masalah.

1. **Kelayakan Mengambil Berlian Merah**: Dalam find_nearest_diamond_direct dan find_nearest_diamond_te

leport, ada kondisi (diamond.properties.points == 2 and self.board_bot.properties.diamonds != 4). Ini adalah pemeriksaan kelayakan: bot hanya akan menargetkan berlian merah (nilai 2 poin) jika ia tidak sedang membawa 4 berlian, untuk menghindari kehilangan semua berlian.

2. **Kelayakan Jalur (obstacle_on_path)**: Fungsi ini secara implisit melakukan pemeriksaan kelayakan jalur. Jika jalur menuju self.goal_position melewati, misalnya, sebuah teleporter yang tidak diinginkan saat bot tidak bertujuan menggunakan teleporter tersebut untuk mencapai self.goal_position, maka jalur tersebut dianggap "tidak layak" untuk dilalui secara langsung. Solusinya adalah membuat manuver penghindaran.

3. **Kelayakan Penggunaan Teleporter**: Saat find_best_way_to_base atau find_nearest_diamond memperti mbangkan teleporter, secara implisit diasumsikan bahwa menggunakan teleporter adalah tindakan yang layak jika menghasilkan jalur yang lebih pendek atau akses ke berlian yang lebih baik. Pengecekan seperti nearest_teleport_position == None memastikan bahwa logika teleporter hanya dijalankan jika teleporter memang ada dan bisa dijangkau.

3.2.2.5 Fungsi Objektif

Fungsi objektif mendefinisikan tujuan yang ingin dioptimalkan oleh algoritma. Dalam DirectLogic, ada beberapa fungsi objektif lokal yang dikejar:

1. **Meminimalkan Jarak/Biaya ke Target**: Saat mencari berlian atau tombol (find_nearest_diamond), fungsi objektifnya adalah meminimalkan jarak (atau jarak yang dinormalisasi dengan poin berlian, distance / diamond.properties.points). Ini bertujuan untuk efisiensi dalam pengumpulan sumber daya.

2. **Meminimalkan Jarak ke Markas**: Saat kembali ke markas (find_best_way_to_base), fungsi objektifnya adalah meminimalkan jarak tempuh.

3. **Memaksimalkan Keselamatan dan Perolehan Skor**: Secara keseluruhan, meskipun tidak dinyatakan sebagai satu fungsi matematis tunggal, tujuan implisitnya adalah memaksimalkan jumlah berlian yang berhasil dibawa kembali ke markas. Keputusan untuk kembali ke markas saat membawa banyak berlian, atau saat musuh dekat, atau waktu hampir habis, adalah upaya untuk mengamankan skor yang sudah ada (meminimalkan risiko kehilangan berlian). Menghindari berlian merah saat membawa 4 berlian juga bertujuan memaksimalkan perolehan (menghindari reset jumlah berlian).

3.2.3 Algoritma Greedy BNF

Algoritma GreedyBNF ini adalah strategi bot yang berfokus pada pengumpulan diamond secara efisien sambil mengelola risiko dan sesekali bersikap agresif. Inti strateginya adalah membagi papan menjadi "blok" dan memilih blok dengan total nilai diamond tertinggi untuk dikumpulkan, memprioritaskan diamond terdekat dalam blok tersebut. Bot akan secara otomatis kembali ke markas jika diamond penuh, waktu hampir habis, terancam musuh saat membawa banyak diamond, atau jika sudah dekat markas dengan muatan signifikan. Jika bot sendiri tidak membawa banyak diamond, ia akan mencoba menyerang musuh terdekat yang membawa diamond. Navigasi mempertimbangkan teleporter untuk mempercepat perjalanan dan memiliki mekanisme untuk menghindari rintangan seperti teleporter yang tidak diinginkan atau berlian merah. Jika tidak ada target jelas, bot akan mencari diamond terdekat atau tombol merah, dan jika buntu, akan roaming.

3.2.3.1 Himpunan Kandidat

Himpunan kandidat bagi bot GreedyBNF adalah kumpulan semua kemungkinan target atau aksi yang dapat dipertimbangkan pada setiap giliran. Ini meliputi posisi setiap berlian di papan, posisi markas sendiri untuk kembali, posisi musuh lain sebagai target serangan, serta posisi masuk teleporter sebagai jalur alternatif. Selain itu, tombol merah, posisi menghindar sementara jika jalur utama terblokir, dan arah-arah standar untuk roaming juga termasuk dalam himpunan kandidat ini, yang akan diseleksi berdasarkan kondisi permainan saat itu.

3.2.3.2 Himpunan Solusi

Himpunan solusi dalam GreedyBNF adalah rencana multi-langkah yang dinamis, direpresentasikan oleh urutan target dalam self.static_goals (seperti daftar berlian dalam satu blok atau rute ke markas via teleporter), dengan self.goal_position sebagai target aktif saat ini. Keputusan pergerakan spesifik (delta_x, delta_y) pada setiap giliran merupakan langkah konkret untuk mengeksekusi bagian dari rencana ini. Ketika satu target dalam rencana tercapai, rencana diperbarui untuk target berikutnya hingga seluruh urutan selesai atau kondisi permainan memaksa perubahan rencana.

3.2.3.3 Fungsi Solusi

Fungsi solusi dalam GreedyBNF menentukan kapan sebuah target atau sub-masalah dalam rencana saat ini dianggap telah selesai. Ini terjadi ketika bot mencapai posisi markasnya (menyelesaikan misi pengantaran diamond dan mereset rencana), mencapai posisi tujuan sementara untuk menghindar (menyelesaikan manuver), keluar dari teleporter yang dituju (menyelesaikan segmen perjalanan via teleporter), atau mencapai target utama lainnya seperti berlian atau posisi musuh (memperbarui rencana ke target berikutnya atau mereset jika rencana habis).

3.2.3.4 Fungsi Seleksi

Fungsi seleksi GreedyBNF merupakan mekanisme inti yang memilih tindakan "terbaik" dari himpunan kandidat berdasarkan prioritas dan kondisi. Prioritas tertinggi adalah kembali ke markas jika kondisi mendesak (diamond penuh, waktu kritis, ancaman musuh, atau dekat markas dengan muatan). Jika tidak, bot dapat memilih menyerang musuh jika bot sendiri tidak membawa banyak diamond. Pilihan berikutnya adalah strategi blok berlian (memilih blok terkaya), diikuti pencarian berlian terdekat (langsung atau via teleporter), atau menargetkan tombol merah. Jika jalur terhalang, bot memilih manuver menghindar, dan jika tidak ada target, bot akan roaming.

3.2.3.5 Fungsi Kelayakan

Fungsi kelayakan GreedyBNF menyaring kandidat untuk memastikan hanya opsi yang valid dan tidak merugikan yang dipertimbangkan. Contohnya, bot tidak akan mengambil berlian merah jika sudah membawa 4 berlian biru (tidak layak karena akan reset). Penggunaan teleporter hanya layak jika ada minimal dua di papan. Serangan terhadap musuh hanya layak jika musuh membawa cukup diamond. Gerakan ke luar batas peta juga dianggap tidak layak. Fungsi ini memastikan keputusan yang diambil masuk akal dalam konteks aturan permainan dan kondisi bot.

3.2.3.5 Fungsi Objektif

Fungsi objektif GreedyBNF adalah serangkaian kriteria yang coba dioptimalkan bot dalam setiap keputusan lokal. Ini termasuk memaksimalkan efisiensi pengumpulan berlian (memilih blok terkaya atau berlian dengan rasio jarak/poin terbaik), meminimalkan risiko dengan mengamankan skor (kembali ke markas pada saat yang tepat), meminimalkan jarak tempuh (memilih rute terpendek ke target atau musuh), dan menjaga keberlangsungan bot (menghindari kondisi merugikan atau bahaya). Dengan mengoptimalkan fungsi-fungsi ini secara lokal, bot bertujuan mencapai skor tinggi secara keseluruhan.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Berdasarkan rancangan algoritma yang telah dibuat, telah dilakukan proses pengetesan untuk menilai seberapa baik rancangan algoritma bekerja. Proses pengetesan akan dilakukan dengan proses- proses sebagai berikut:

1. Bot dijalankan single player pada board sebanyak 5 kali dan dihitung rata-rata diamonds yang didapatkan. Game akan dijalankan selama 60 detik dengan delay tiap gerakannya adalah 1000 ms. Khusus untuk bot tackle, tes ini akan dilakukan 1 vs 1 dengan bot lain karena bot tackle tidak akan bisa berjalan jika tidak ada musuh.
2. Ketiga bot akan dijalankan bersamaan dan akan dinilai berdasarkan leaderboard yang tersedia di dalam game. Tes ini akah dilakukan sebanyak 5 kali.

Dengan proses diatas, diharapkan efektivitas dari setiap bot dapat dinilai secara objektif.Berikut adalah hasil pengetesan efisiensi yang telah dilakukan terhadap algoritma greedy direct dan algoritma greedy BNF yang dijalankan secara single player.

Tabel 3.3.1 Hasil Tes Algoritma Direct

Percobaan	**Diamonds yang diperoleh**
1	10
2	10
3	9
4	10
5	9
Rata-rata	9,6

Tabel 3.3.2 Hasil Tes Algoritma BNF

Percobaan	**Diamonds yang diperoleh**
1	11
2	7
3	12
4	4
5	4
Rata-rata	7.6

Berikut adalah hasil pengetesan yang telah dilakukan terhadap algoritma greedy tackle dengan sistem 1 vs 1 terhadap bot lain,dalam tes ini bot yang digunakan adalah bot direct.

Percobaan	Diamonds yang diperoleh
1	7
2	7
3	6
4	6
5	7
Rata-rata	6,6

Berikut adalah hasil pengetesan yang telah dilakukan terhadap algoritma greedy BNF,direct,dan tackle yang dijalankan secara bersamaan.

Percobaan	Diamond yang diperoleh		
	BNF	Direct	Tackle
1	7	0	5
2	8	4	5
3	6	6	6
4	7	5	5
5	7	3	7
Rata-rata	7	3,6	5.6

3.4 Strategi Greedy yang Dipilih

Berdasarkan hasil pengetesan yang telah dilakukan, strategi greedy yang dipilih adalah Greedy BNF. Alasan dipilihnya algoritma ini adalah sebagai berikut:

1. Dalam pengetesan single player yang telah dilakukan, algoritma ini mendapatkan score yang lebih banyak daripada algoritma tackle dan lebih sedikit daripada algoritma direct dengan jarak yang cukup jauh dengan rata-rata 7,6 , tetapi algoritma BNF mendapatkan rata-rata paling tinggi ketika 3 algoritma tersebut dimainkan bersamaan. Walaupun algoritma tackle sangat baik digunakan untuk 1 vs 1, namun sebagian besar game akan dilakukan dengan pemain lebih dari 2 bot karena kurang efektif ketika digunakan untuk bermain multiplayer karena bot hanya bisa mentarget 1 musuh dan musuh lain akan bebas untuk mengumpulkan diamond.
2. Algoritma ini telah memanfaatkan hampir semua elemen yang ada di dalam game Diamonds. Elemen- elemen tersebut termasuk red button dan teleporter. Red button akan banyak digunakan oleh bot dengan tujuan untuk merugikan lawan dan teleporter digunakan untuk mempersingkat jarak untuk mendapatkan diamonds atau jarak untuk kembali ke base.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

4.1.1 Pseudocode

```
==================================================
KELAS GreedyBNF
==================================================

  // --- Atribut Kelas (Static / Bersama) ---
  VARIABEL KELAS static_goals : DAFTAR DARI Posisi
     // Rencana urutan target utama
  VARIABEL KELAS static_goal_teleport : GameObject ATAU NIHIL
     // Teleporter yang dituju dalam rencana
  VARIABEL KELAS static_temp_goals : Posisi ATAU NIHIL
     // Target sementara untuk menghindari rintangan

  // --- Konstanta Logika Permainan ---
  KONSTANTA LOW_TIME_THRESHOLD_MS = 7000
  KONSTANTA MIN_DIAMONDS_FOR_LOW_TIME_RETURN = 2
  KONSTANTA MIN_DIAMONDS_TO_FLEE_TRESHOLD = 3
  KONSTANTA ENEMY_FLEE_DISTANCE = 2
  KONSTANTA MIN_DIAMONDS_NEAR_BASE_RETURN = 3
  KONSTANTA MAX_OWN_DIAMONDS_FOR_AGGRESSION = 2
  KONSTANTA MIN_ENEMY_DIAMONDS_TO_ATTACK = 2

  // --- Atribut Instance (Setiap Bot Punya Sendiri) ---
  VARIABEL directions : DAFTAR DARI Tuple(Integer, Integer)
     // Arah gerak dasar [(1,0), (0,1), ...]
  VARIABEL current_direction_index : Integer
     // Indeks untuk pola roaming
  VARIABEL goal_position : Posisi ATAU NIHIL
     // Target aktif saat ini dari static_goals
  VARIABEL current_diamond_target_distance : Angka
     // Jarak ke target diamond (untuk optimasi kembali ke base)

  // Atribut yang diupdate tiap giliran
  VARIABEL board_bot : GameObject
  VARIABEL board : Board
  VARIABEL diamonds : DAFTAR DARI GameObject
  VARIABEL other_bots : DAFTAR DARI GameObject
  VARIABEL teleporters : DAFTAR DARI GameObject
  VARIABEL red_button : GameObject ATAU NIHIL
--------------------------------------------------
Inisialisasi Instance
--------------------------------------------------

  FUNGSI __init__():
```

```
    SET directions KE [(1, 0), (0, 1), (-1, 0), (0, -1)]
    SET current_direction_index KE 0
    SET goal_position KE NIHIL
    SET current_diamond_target_distance KE INFINITI
    // Inisialisasi atribut board_bot, dll. akan dilakukan di _update_board_state
```

```
------------------------------------------------
Fungsi Helper: Manajemen Goal
------------------------------------------------

  FUNGSI _reset_major_goals():
    SET static_goals KE DAFTAR KOSONG
    SET static_goal_teleport KE NIHIL
    SET goal_position KE NIHIL
    SET current_diamond_target_distance KE INFINITI

  FUNGSI _reset_temp_goals():
    SET static_temp_goals KE NIHIL
```

```
------------------------------------------------
Fungsi Helper: Update State Papan
------------------------------------------------

  FUNGSI _update_board_state(board_bot_input, board_input):
    SET board_bot KE board_bot_input
    SET board KE board_input
    SET diamonds KE (daftar diamond dari board_input)
    SET other_bots KE (daftar bot lain selain board_bot_input dari board_input)
    SET teleporters KE (daftar teleporter dari board_input)
    SET red_button KE (objek tombol merah dari board_input, JIKA ADA)
```

```
------------------------------------------------
Fungsi Utama: next_move
------------------------------------------------

  FUNGSI next_move(board_bot_input, board_input) MENGEMBALIKAN Tuple(Integer,
Integer):
    PANGGIL _update_board_state(board_bot_input, board_input)

    VARIABEL props = properti board_bot
    VARIABEL current_pos = posisi board_bot
    VARIABEL my_base_pos = posisi base dari board_bot
```

```
------------------------------------------------
Fungsi Utama: next_move (lanjutan)
------------------------------------------------

// === Tahap 1: Update status goal berdasarkan posisi saat ini ===
  JIKA current_pos SAMA DENGAN my_base_pos MAKA:
    PANGGIL _reset_major_goals()
    PANGGIL _reset_temp_goals()
  LAIN JIKA static_temp_goals ADA DAN current_pos SAMA DENGAN
static_temp_goals MAKA:
    PANGGIL _reset_temp_goals()
```

```
  LAIN JIKA static_goal_teleport ADA DAN current_pos SAMA DENGAN (PANGGIL find_other_teleport_pos(static_goal_teleport)) MAKA:
    JIKA static_goals TIDAK KOSONG DAN elemen_pertama(static_goals) SAMA DENGAN posisi static_goal_teleport MAKA:
      HAPUS elemen_pertama DARI static_goals
    SET static_goal_teleport KE NIHIL
    JIKA static_goals TIDAK KOSONG MAKA:
      SET goal_position KE elemen_pertama(static_goals)
    LAIN MAKA:
      PANGGIL _reset_major_goals()
  LAIN JIKA goal_position ADA DAN current_pos SAMA DENGAN goal_position MAKA:
    JIKA static_goals TIDAK KOSONG DAN elemen_pertama(static_goals) SAMA DENGAN goal_position MAKA:
      HAPUS elemen_pertama DARI static_goals
    JIKA static_goals TIDAK KOSONG MAKA:
      SET goal_position KE elemen_pertama(static_goals)
    LAIN MAKA:
      PANGGIL _reset_major_goals()

// === Tahap 2: Tentukan goal baru jika perlu (Prioritas Kembali ke Base) ===
  VARIABEL should_return_to_base = (
    props.diamonds == 5 ATAU
    (props.milliseconds_left < LOW_TIME_THRESHOLD_MS DAN props.diamonds >= MIN_DIAMONDS_FOR_LOW_TIME_RETURN) ATAU
    (PANGGIL is_threatened(current_pos) DAN props.diamonds >= MIN_DIAMONDS_TO_FLEE_TRESHOLD) ATAU
    (PANGGIL is_opportunistic_to_return(my_base_pos, current_pos) DAN props.diamonds >= MIN_DIAMONDS_NEAR_BASE_RETURN)
  )

  JIKA should_return_to_base MAKA:
    PANGGIL set_goal_to_base(my_base_pos, current_pos)

  JIKA goal_position SAMA DENGAN my_base_pos DAN static_goals KOSONG MAKA:
    SET static_goals KE DAFTAR_BERISI [my_base_pos]

// === Tahap 3: Agresi atau mencari diamond/button jika tidak kembali ke base ===
  JIKA goal_position TIDAK ADA ATAU goal_position TIDAK SAMA DENGAN my_base_pos MAKA:
    JIKA props.diamonds <= MAX_OWN_DIAMONDS_FOR_AGGRESSION MAKA:
      VARIABEL best_enemy_target = PANGGIL find_best_enemy_target(current_pos)
      JIKA best_enemy_target ADA MAKA:
        VARIABEL enemy_dist = PANGGIL manhattan_distance(current_pos, posisi best_enemy_target)
        VARIABEL current_goal_dist = JIKA goal_position ADA DAN static_goals TIDAK KOSONG MAKA PANGGIL manhattan_distance(current_pos, goal_position) LAIN INFINITI

        JIKA enemy_dist < current_goal_dist ATAU goal_position TIDAK ADA MAKA:
```

```
        PANGGIL _reset_major_goals()
        SET goal_position KE posisi best_enemy_target
        SET static_goals KE DAFTAR_BERISI [goal_position]

    JIKA static_goals KOSONG MAKA:
     PANGGIL find_best_block_strategy()
     JIKA static_goals KOSONG MAKA:
      PANGGIL find_direct_diamond_strategy(current_pos)

     JIKA static_goals TIDAK KOSONG MAKA:
      SET goal_position KE elemen_pertama(static_goals)
      JIKA goal_position TIDAK SAMA DENGAN my_base_pos MAKA:
       SET current_diamond_target_distance KE PANGGIL manhattan_distance(current_pos, goal_position)
     LAIN JIKA red_button ADA MAKA:
      PANGGIL _reset_major_goals()
      SET goal_position KE posisi red_button
      SET static_goals KE DAFTAR_BERISI [posisi red_button]

// === Tahap 4: Logika Penghindaran Rintangan dan Perhitungan Gerakan ===
  VARIABEL effective_goal = JIKA static_temp_goals ADA MAKA static_temp_goals LAIN goal_position
  VARIABEL delta_x = 0, delta_y = 0

  JIKA effective_goal ADA MAKA:
    JIKA static_temp_goals TIDAK ADA DAN goal_position ADA MAKA:
     JIKA TIDAK (static_goal_teleport ADA DAN posisi static_goal_teleport SAMA DENGAN goal_position) MAKA:
      PANGGIL check_obstacle_on_path('teleporter', current_pos, goal_position)
     JIKA props.diamonds == 4 MAKA:
      PANGGIL check_obstacle_on_path('redDiamond', current_pos, goal_position)
     JIKA red_button ADA DAN goal_position TIDAK SAMA DENGAN posisi red_button MAKA:
      PANGGIL check_obstacle_on_path('redButton', current_pos, goal_position)

    SET effective_goal KE (JIKA static_temp_goals ADA MAKA static_temp_goals LAIN goal_position)

    JIKA effective_goal ADA MAKA:
     (delta_x, delta_y) = PANGGIL get_direction(current_pos.x, current_pos.y, effective_goal.x, effective_goal.y)

// === Tahap 5: Logika Roaming atau Jika Terjebak ===
  JIKA effective_goal TIDAK ADA ATAU (delta_x == 0 DAN delta_y == 0 DAN current_pos TIDAK SAMA DENGAN effective_goal) MAKA:
    JIKA effective_goal ADA DAN current_pos TIDAK SAMA DENGAN effective_goal MAKA:
     PANGGIL _reset_major_goals()
     PANGGIL _reset_temp_goals()
```

```
    VARIABEL moved_in_roam = SALAH
    UNTUK i DARI 0 SAMPAI panjang(directions) - 1:
      VARIABEL next_dir_idx = (current_direction_index + i) MODULO panjang(directions)
      VARIABEL (dx_try, dy_try) = directions[next_dir_idx]
      JIKA posisi_baru (current_pos.x + dx_try, current_pos.y + dy_try) VALID_DI_PAPAN
MAKA:
        SET delta_x KE dx_try
        SET delta_y KE dy_try
        SET current_direction_index KE (next_dir_idx + 1) MODULO panjang(directions)
        SET moved_in_roam KE BENAR
        BERHENTI_LOOP
    JIKA moved_in_roam SALAH MAKA:
      SET delta_x KE 0
      SET delta_y KE 0

// === Tahap 6: Safety Net Jika Masih Tidak Bergerak ===
  JIKA delta_x == 0 DAN delta_y == 0 DAN goal_position ADA DAN current_pos TIDAK
SAMA DENGAN goal_position MAKA:
    PANGGIL _reset_major_goals()
    PANGGIL _reset_temp_goals()

  KEMBALIKAN (delta_x, delta_y)
```

```
------------------------------------------------
Fungsi Helper: Kalkulasi Jarak
------------------------------------------------
FUNGSI manhattan_distance(pos1, pos2) MENGEMBALIKAN Integer:
    KEMBALIKAN abs(pos1.x - pos2.x) + abs(pos1.y - pos2.y)
```

```
------------------------------------------------
Fungsi Helper: Menentukan Tujuan ke Base
------------------------------------------------
FUNGSI set_goal_to_base(my_base_pos, current_pos):
    VARIABEL path_to_base = PANGGIL find_best_way_to_base(my_base_pos,
current_pos)
    PANGGIL _reset_major_goals()  // Reset goal lama sebelum set goal baru
    SET static_goals KE path_to_base

    JIKA static_goals KOSONG MAKA:  // Safety net
        SET static_goals KE DAFTAR_BERISI [my_base_pos]

    SET goal_position KE elemen_pertama(static_goals)

    // Jika jalur ke base via teleporter, catat teleporter yang dituju
    JIKA panjang(static_goals) > 1 DAN elemen_pertama(static_goals) TIDAK SAMA
DENGAN my_base_pos MAKA:
        UNTUK SETIAP tp_obj DALAM teleporters:
            JIKA posisi tp_obj SAMA DENGAN elemen_pertama(static_goals) MAKA:
                SET static_goal_teleport KE tp_obj
                BERHENTI_LOOP
```

```
------------------------------------------------
Fungsi Helper: Pengecekan Kondisi Bahaya/Peluang
```

```
---------------------------------------------
FUNGSI is_threatened(current_pos) MENGEMBALIKAN Boolean:
  UNTUK SETIAP enemy DALAM other_bots:
    JIKA PANGGIL manhattan_distance(current_pos, posisi enemy) <= ENEMY_FLEE_DISTANCE MAKA:
      KEMBALIKAN BENAR
  KEMBALIKAN SALAH

FUNGSI is_opportunistic_to_return(my_base_pos, current_pos) MENGEMBALIKAN Boolean:
  JIKA (properti board_bot).diamonds == 0 MAKA
    KEMBALIKAN SALAH
  VARIABEL dist_to_base = PANGGIL manhattan_distance(current_pos, my_base_pos)
  // Jika jarak ke base lebih pendek dari 75% jarak ke target diamond saat ini
  JIKA current_diamond_target_distance TIDAK SAMA DENGAN INFINITI DAN dist_to_base < current_diamond_target_distance * 0.75 DAN dist_to_base > 0 MAKA:
    KEMBALIKAN BENAR
  KEMBALIKAN SALAH
```

```
---------------------------------------------
Fungsi Helper: Mencari Target Musuh
---------------------------------------------
FUNGSI find_best_enemy_target(current_pos) MENGEMBALIKAN GameObject ATAU NIHIL:
  VARIABEL best_target = NIHIL
  VARIABEL min_score = INFINITI
  UNTUK SETIAP enemy DALAM other_bots:
    JIKA (properti enemy).diamonds >= MIN_ENEMY_DIAMONDS_TO_ATTACK MAKA:
      VARIABEL dist = PANGGIL manhattan_distance(current_pos, posisi enemy)
      JIKA dist == 0 MAKA
        LANJUT_LOOP // Jangan target diri sendiri atau musuh di posisi sama
      VARIABEL score = dist // Skor sederhana: jarak
      JIKA score < min_score MAKA:
        SET min_score KE score
        SET best_target KE enemy
  KEMBALIKAN best_target
```

```
//---------------------------------------------
// Fungsi Helper: Pengecekan Kelayakan Diamond
//---------------------------------------------
FUNGSI _can_collect_diamond(diamond_obj) MENGEMBALIKAN Boolean:
  // Tidak bisa ambil diamond merah (poin 2) jika sudah punya 4 diamond biru
  KEMBALIKAN TIDAK (
    (properti diamond_obj).points == 2
    DAN
    (properti board_bot).diamonds == 4
  )
```

```
//---------------------------------------------
// Fungsi Helper: Strategi Mencari Diamond Langsung/Via Teleporter
//---------------------------------------------
FUNGSI find_direct_diamond_strategy(current_pos):
```

```
    VARIABEL best_direct_diamond_pos = NIHIL
    VARIABEL min_direct_dist_score = INFINITI
    VARIABEL valid_diamonds = (Filter diamonds berdasarkan PANGGIL
_can_collect_diamond)

    UNTUK SETIAP diamond_obj DALAM valid_diamonds:
        VARIABEL dist = PANGGIL manhattan_distance(current_pos, posisi diamond_obj)
        JIKA dist == 0 MAKA LANJUT_LOOP
        VARIABEL score = dist / (properti diamond_obj).points // Jarak per poin
        JIKA score < min_direct_dist_score MAKA:
            SET min_direct_dist_score KE score
            SET best_direct_diamond_pos KE posisi diamond_obj

    VARIABEL best_teleport_path = DAFTAR_KOSONG
    VARIABEL min_teleport_dist_score = INFINITI
    VARIABEL chosen_teleporter_obj = NIHIL

    JIKA teleporters ADA DAN panjang(teleporters) >= 2 MAKA:
        (VARIABEL entry_tp, VARIABEL exit_tp, VARIABEL tp_obj, VARIABEL
dist_to_entry_tp) = PANGGIL find_nearest_teleporter_data(current_pos)
        JIKA entry_tp ADA DAN exit_tp ADA DAN tp_obj ADA MAKA:
            UNTUK SETIAP diamond_obj DALAM valid_diamonds:
                VARIABEL dist_exit_to_diamond = PANGGIL manhattan_distance(exit_tp,
posisi diamond_obj)
                VARIABEL total_dist_via_tp = dist_to_entry_tp + dist_exit_to_diamond
                JIKA total_dist_via_tp == 0 DAN posisi diamond_obj TIDAK SAMA DENGAN
exit_tp MAKA LANJUT_LOOP
                VARIABEL score_tp = total_dist_via_tp / (properti diamond_obj).points
                JIKA score_tp < min_teleport_dist_score MAKA:
                    SET min_teleport_dist_score KE score_tp
                    SET best_teleport_path KE DAFTAR_BERISI [entry_tp, posisi diamond_obj]
                    SET chosen_teleporter_obj KE tp_obj

    // Pilih strategi terbaik antara direct dan via teleporter
    JIKA best_direct_diamond_pos ADA DAN min_direct_dist_score <=
min_teleport_dist_score MAKA:
        SET static_goals KE DAFTAR_BERISI [best_direct_diamond_pos]
        SET static_goal_teleport KE NIHIL
    LAIN JIKA best_teleport_path TIDAK KOSONG DAN chosen_teleporter_obj ADA
MAKA:
        SET static_goals KE best_teleport_path
        SET static_goal_teleport KE chosen_teleporter_obj
    // Jika tidak ada, static_goals dan static_goal_teleport tetap (biasanya kosong atau dari
strategi lain)
```

```
//------------------------------------------------
// Fungsi Helper: Strategi Mencari Diamond Berdasarkan Blok
//------------------------------------------------
FUNGSI find_best_block_strategy():
    PANGGIL _reset_major_goals() // Reset sebelum menentukan strategi blok baru
```

```
    JIKA diamonds KOSONG MAKA:
        SET static_goals KE DAFTAR KOSONG
        KEMBALI

    VARIABEL blockH = max(1, tinggi_papan // 3)
    VARIABEL blockW = max(1, lebar_papan // 3)
    VARIABEL block_values = Array_2D(3x3) DARI Integer, diinisialisasi 0
    VARIABEL block_diamonds_lists = Array_2D(3x3) DARI Daftar_Posisi, diinisialisasi
DAFTAR KOSONG

    VARIABEL diamonds_needed = 5 - (properti board_bot).diamonds
    JIKA diamonds_needed <= 0 MAKA:
        SET static_goals KE DAFTAR KOSONG
        KEMBALI

    VARIABEL valid_diamonds = (Filter diamonds berdasarkan PANGGIL
_can_collect_diamond)
    JIKA valid_diamonds KOSONG MAKA:
        SET static_goals KE DAFTAR KOSONG
        KEMBALI

    UNTUK SETIAP diamond_obj DALAM valid_diamonds:
        JIKA posisi diamond_obj TIDAK VALID_DI_PAPAN MAKA LANJUT_LOOP
        VARIABEL block_row = min((posisi diamond_obj).y // blockH, 2)
        VARIABEL block_col = min((posisi diamond_obj).x // blockW, 2)
        TAMBAHKAN (properti diamond_obj).points KE
block_values[block_row][block_col]
        TAMBAHKAN posisi diamond_obj KE block_diamonds_lists[block_row][block_col]

    VARIABEL best_score_block = -1
    VARIABEL best_block_idx = NIHIL
    UNTUK r DARI 0 SAMPAI 2:
        UNTUK c DARI 0 SAMPAI 2:
            JIKA block_values[r][c] == 0 MAKA LANJUT_LOOP
            VARIABEL current_block_score = block_values[r][c]
            JIKA block_values[r][c] >= diamonds_needed MAKA: // Beri bonus jika blok ini
cukup untuk 5 diamond
                SET current_block_score KE current_block_score + 10
            JIKA current_block_score > best_score_block MAKA:
                SET best_score_block KE current_block_score
                SET best_block_idx KE (r, c)

    JIKA best_block_idx ADA MAKA:
        (VARIABEL r_best, VARIABEL c_best) = best_block_idx
        SET static_goals KE (URUTKAN block_diamonds_lists[r_best][c_best]
BERDASARKAN PANGGIL manhattan_distance(posisi board_bot, p_diamond))
    LAIN MAKA:
        SET static_goals KE DAFTAR KOSONG
```

```
//--------------------------------------------------
// Fungsi Helper: Navigasi Teleporter
```

```
//------------------------------------------------
FUNGSI find_nearest_teleporter_data(current_pos) MENGEMBALIKAN Tuple(Posisi, Posisi, GameObject, Integer) ATAU Tuple(NIHIL, NIHIL, NIHIL, INFINITI):
   JIKA teleporters KOSONG ATAU panjang(teleporters) < 2 MAKA:
      KEMBALIKAN (NIHIL, NIHIL, NIHIL, INFINITI)

   VARIABEL nearest_entry_pos = NIHIL
   VARIABEL min_dist_to_entry = INFINITI
   VARIABEL selected_tp_obj = NIHIL

   UNTUK SETIAP tp_obj DALAM teleporters:
      VARIABEL dist = PANGGIL manhattan_distance(current_pos, posisi tp_obj)
      JIKA dist < min_dist_to_entry MAKA:
         SET min_dist_to_entry KE dist
         SET nearest_entry_pos KE posisi tp_obj
         SET selected_tp_obj KE tp_obj

   JIKA nearest_entry_pos ADA DAN selected_tp_obj ADA MAKA:
      VARIABEL other_tp_pos = PANGGIL find_other_teleport_pos(selected_tp_obj)
      JIKA other_tp_pos ADA MAKA:
         KEMBALIKAN (nearest_entry_pos, other_tp_pos, selected_tp_obj, min_dist_to_entry)
   KEMBALIKAN (NIHIL, NIHIL, NIHIL, INFINITI)


FUNGSI find_other_teleport_pos(teleporter_obj_input) MENGEMBALIKAN Posisi ATAU NIHIL:
   UNTUK SETIAP tp DALAM teleporters:
      JIKA id tp TIDAK SAMA DENGAN id teleporter_obj_input MAKA:
         KEMBALIKAN posisi tp
   KEMBALIKAN NIHIL


FUNGSI find_best_way_to_base(my_base_pos, current_pos) MENGEMBALIKAN Daftar_Posisi:
   VARIABEL dist_direct = PANGGIL manhattan_distance(current_pos, my_base_pos)
   JIKA dist_direct == 0 MAKA
      KEMBALIKAN DAFTAR_BERISI [my_base_pos]

   VARIABEL path_via_teleporter = NIHIL
   VARIABEL dist_via_teleporter = INFINITI

   JIKA teleporters ADA DAN panjang(teleporters) >= 2 MAKA:
      (VARIABEL entry_tp, VARIABEL exit_tp, _, VARIABEL dist_to_entry_tp) = PANGGIL find_nearest_teleporter_data(current_pos)
      JIKA entry_tp ADA DAN exit_tp ADA MAKA:
         VARIABEL dist_exit_to_base = PANGGIL manhattan_distance(exit_tp, my_base_pos)
         SET dist_via_teleporter KE dist_to_entry_tp + dist_exit_to_base
```

```
        SET path_via_teleporter KE DAFTAR_BERISI [entry_tp, my_base_pos] // Urutan: ke teleporter, lalu ke base

    JIKA path_via_teleporter ADA DAN dist_via_teleporter < dist_direct MAKA:
        KEMBALIKAN path_via_teleporter
    KEMBALIKAN DAFTAR_BERISI [my_base_pos] // Jalur langsung lebih baik atau satu-satunya pilihan
```

```
//--------------------------------------------------
// Fungsi Helper: Penghindaran Rintangan
//--------------------------------------------------
FUNGSI check_obstacle_on_path(obj_type_to_avoid, current_pos, ultimate_dest_pos):
    JIKA current_pos SAMA DENGAN ultimate_dest_pos MAKA
        KEMBALI // Sudah di tujuan

    VARIABEL objects_to_check = DAFTAR_KOSONG
    JIKA obj_type_to_avoid SAMA DENGAN 'teleporter' MAKA
        SET objects_to_check KE teleporters
    LAIN JIKA obj_type_to_avoid SAMA DENGAN 'redDiamond' MAKA
        SET objects_to_check KE (Filter diamond dengan poin 2)
    LAIN JIKA obj_type_to_avoid SAMA DENGAN 'redButton' DAN red_button ADA MAKA
        SET objects_to_check KE DAFTAR_BERISI [red_button]

    JIKA objects_to_check KOSONG MAKA
        KEMBALI

    (VARIABEL dx_naive, VARIABEL dy_naive) = PANGGIL get_direction(current_pos.x, current_pos.y, ultimate_dest_pos.x, ultimate_dest_pos.y)
    JIKA dx_naive == 0 DAN dy_naive == 0 MAKA
        KEMBALI // Tidak ada arah naif (mungkin sudah di tujuan atau terblokir total)

    VARIABEL next_pos_naive = Posisi(current_pos.x + dx_naive, current_pos.y + dy_naive)
    VARIABEL is_obstacle_on_naive_path = (ADA object DALAM objects_to_check YANG posisinya SAMA DENGAN next_pos_naive)

    JIKA is_obstacle_on_naive_path SALAH MAKA
        KEMBALI // Jalur naif aman

    // Jika jalur naif terhalang, coba langkah memutar
    VARIABEL detour_options = DAFTAR_KOSONG
    JIKA dx_naive TIDAK SAMA DENGAN 0 MAKA
        // Jika bergerak horizontal, coba vertikal
        SET detour_options KE [(0, 1), (0, -1)]
    LAIN JIKA dy_naive TIDAK SAMA DENGAN 0 MAKA
        // Jika bergerak vertikal, coba horizontal
        SET detour_options KE [(1, 0), (-1, 0)]

    UNTUK SETIAP (ddx, ddy) DALAM detour_options:
        VARIABEL detour_pos = Posisi(current_pos.x + ddx, current_pos.y + ddy)
```

```
    JIKA detour_pos VALID_DI_PAPAN DAN (TIDAK ADA object DALAM objects_to_check YANG posisinya SAMA DENGAN detour_pos) MAKA:
        SET static_temp_goals KE detour_pos
        KEMBALI // Ditemukan jalur memutar

  // Jika langkah memutar menyamping gagal, coba mundur (jika ada arah naif)
  JIKA dx_naive TIDAK SAMA DENGAN 0 ATAU dy_naive TIDAK SAMA DENGAN 0 MAKA:
    VARIABEL backward_pos = Posisi(current_pos.x - dx_naive, current_pos.y - dy_naive)
    JIKA backward_pos VALID_DI_PAPAN DAN (TIDAK ADA object DALAM objects_to_check YANG posisinya SAMA DENGAN backward_pos) MAKA:
        SET static_temp_goals KE backward_pos
        KEMBALI // Mundur sebagai opsi terakhir penghindaran
```

4.1.2 Penjelasan Alur Program

Pada setiap gilirannya, bot GreedyBNF memulai dengan mengamati kondisi terkini papan permainan, memperbarui pengetahuannya tentang posisi diamond, musuh, dan elemen penting lainnya. Ia kemudian memeriksa apakah telah mencapai tujuan dari giliran sebelumnya, seperti tiba di markas, menyelesaikan manuver penghindaran, atau keluar dari teleporter. Jika ya, ia akan mereset rencana terkait dan bersiap untuk langkah berikutnya.

Selanjutnya, bot membuat keputusan strategis utama. Prioritas tertingginya adalah kembali ke markas jika ia membawa banyak diamond, waktu hampir habis, terancam oleh musuh di dekat, atau jika secara kebetulan sudah sangat dekat dengan markasnya sambil membawa muatan berharga. Jika kondisi kembali ke markas terpenuhi, ia akan merencanakan rute teraman dan tercepat pulang, mengabaikan sementara target lain.

Namun, jika kembali ke markas belum mendesak, bot mempertimbangkan opsi lain. Jika ia tidak membawa banyak diamond, ia akan mencari kesempatan untuk menyerang musuh terdekat yang diketahui membawa diamond, berharap bisa merebut hasil kerja keras lawan. Jika tidak ada musuh yang menggiurkan atau jika agresi bukan pilihan terbaik, bot akan beralih ke mode pengumpulan. Ia akan mencoba strategi "blok", yaitu mengidentifikasi area 3x3 di peta yang paling kaya akan diamond dan merencanakan untuk mengumpulkannya secara berurutan. Jika strategi blok tidak menjanjikan, ia akan mencari diamond tunggal terbaik yang bisa dicapai, baik secara langsung maupun melalui teleporter, dengan memperhitungkan jarak dan nilai diamond tersebut. Sebagai pilihan terakhir jika tidak ada diamond yang menarik, ia mungkin akan menargetkan tombol merah.

Setelah tujuan utama (atau serangkaian tujuan) ditentukan, bot memikirkan jalur praktisnya. Sebelum bergerak, ia akan memeriksa apakah jalur langsung ke tujuannya terhalang oleh rintangan yang tidak diinginkan, seperti teleporter yang bukan bagian dari rencananya, diamond merah (jika ia membawa 4 diamond biru), atau tombol merah yang bukan targetnya. Jika ada halangan, ia akan

mencoba melakukan manuver menghindar sementara dengan mengambil satu langkah ke samping atau mundur.

Akhirnya, jika setelah semua pertimbangan ini bot tidak memiliki tujuan yang jelas, atau jika ia merasa terjebak dan tidak bisa bergerak menuju tujuannya, ia akan beralih ke mode "roaming", yaitu bergerak secara acak namun terstruktur ke salah satu arah yang valid untuk menjelajahi area baru dan mencari peluang. Dengan alur ini, bot berusaha untuk terus produktif, menyeimbangkan antara pengumpulan sumber daya, manajemen risiko, dan adaptasi terhadap kondisi permainan yang dinamis.

4.2 Struktur Data yang Digunakan

Dalam merancang logikanya, bot GreedyBNF memanfaatkan beberapa struktur data kunci untuk menyimpan informasi dan rencana. Rencana pergerakan jangka panjangnya, seperti urutan diamond yang akan diambil dalam satu area atau langkah-langkah menuju markas, disimpan dalam sebuah daftar posisi yang disebut static_goals. Jika dalam rencana ini bot perlu menggunakan teleporter, objek GameObject dari teleporter yang dituju akan disimpan dalam static_goal_teleport. Untuk manuver penghindaran rintangan jangka pendek, bot menggunakan static_temp_goals, sebuah posisi opsional yang menandakan target sementara. Target aktif yang sedang dikejar saat ini juga disimpan sebagai posisi opsional dalam goal_position.

Informasi mengenai kondisi papan permainan juga dikelola dengan cermat. Semua diamond yang tersedia disimpan dalam sebuah daftar GameObject bernama diamonds, sementara musuh-musuh lain disimpan dalam daftar serupa, other_bots. Begitu pula dengan teleporters. Keberadaan tombol merah di papan dicatat dalam red_button sebagai sebuah GameObject opsional. Untuk membantu dalam pengambilan keputusan, seperti kapan harus kembali ke markas karena dekat, jarak ke target diamond saat ini disimpan dalam current_diamond_target_distance sebagai angka float, yang bisa bernilai tak terhingga jika belum ada target. Arah-arah dasar pergerakan disimpan sebagai daftar tuple integer dalam directions. Ketika bot melakukan strategi pembagian area, ia menggunakan daftar bersarang (mirip array 2D) untuk mencatat total nilai diamond dan daftar posisi diamond dalam setiap blok peta. Terakhir, bot sangat bergantung pada struktur data bawaan dari game seperti GameObject, Position, dan Board untuk mengakses detail setiap entitas di papan dan properti bot itu sendiri.

4.3 Pengujian Program

4.3.1 Skenario Pengujian

Skenario	Penjabaran
Pengumpulan Diamond Dasar & Strategi Blok	• **Kondisi:** Papan permainan dengan beberapa diamond tersebar, beberapa di antaranya membentuk "blok" yang lebih kaya, sementara yang lain terisolasi. Tidak ada musuh atau teleporter yang signifikan mengganggu. Bot memulai dengan 0 diamond dan waktu permainan cukup. • **Perilaku yang Diharapkan:** Bot pertama-tama harus mencoba mengidentifikasi dan menargetkan blok diamond yang paling menguntungkan (find_best_block_strategy). Jika tidak ada blok yang jelas atau setelah blok selesai, bot harus

	beralih mencari diamond individual terdekat yang paling efisien (rasio jarak/poin terbaik melalui find_direct_diamond_strategy). Bot harus terus mengumpulkan hingga mencapai 5 diamond, lalu otomatis menargetkan markasnya. • **Fokus Pengujian:** Keakuratan pemilihan blok, efisiensi pengumpulan dalam blok, transisi ke pencarian diamond individual, dan keputusan kembali ke markas setelah 5 diamond.
Skenario Prioritas Kembali ke Markas (Kondisi Kritis)	• **Kondisi:** Bot sedang dalam misi mengumpulkan diamond (misalnya, memiliki 2-3 diamond). Kemudian, salah satu kondisi berikut terjadi: (a) waktu permainan hampir habis (di bawah LOW_TIME_THRESHOLD_MS), ATAU (b) musuh mendekat dalam ENEMY_FLEE_DISTANCE saat bot membawa cukup diamond (MIN_DIAMONDS_TO_FLEE_TRESHOLD). • **Perilaku yang Diharapkan:** Segera setelah kondisi kritis terdeteksi, bot harus mengabaikan target diamond saat ini (atau target musuh jika sedang agresif) dan langsung mengubah tujuannya ke markas (set_goal_to_base). Ia harus memilih jalur tercepat ke markas, mempertimbangkan teleporter jika ada. • **Fokus Pengujian:** Responsivitas terhadap kondisi waktu habis dan ancaman musuh, pengabaian target lama, dan pemilihan jalur efisien ke markas.
Skenario Agresi Cerdas terhadap Musuh	• **Kondisi:** Bot membawa sedikit diamond (<= MAX_OWN_DIAMONDS_FOR_AGGRESSION). Ada satu atau lebih musuh di papan, beberapa di antaranya membawa sejumlah diamond yang signifikan (>= MIN_ENEMY_DIAMONDS_TO_ATTACK). Pilihan diamond biasa mungkin kurang menarik dibandingkan potensi rampasan dari musuh. • **Perilaku yang Diharapkan:** Bot harus mengidentifikasi musuh yang paling layak diserang (berdasarkan kedekatan dan jumlah diamond yang dibawa musuh). Ia harus mengubah targetnya untuk mengejar musuh tersebut. Jika berhasil "men-tackle" (mencapai posisi musuh), ia akan mendapatkan diamond musuh. Bot harus kembali ke mode pengumpulan atau kembali ke base setelah agresi, tergantung jumlah diamond yang dimilikinya sekarang. • **Fokus Pengujian:** Aktivasi mode agresi, pemilihan target musuh yang benar, dan transisi setelah agresi.
Skenario Navigasi Menggunakan Teleporter (Efisiensi & Tujuan Jauh)	• **Kondisi:** Target utama bot (baik itu diamond bernilai tinggi atau markasnya saat kembali) berada cukup jauh. Terdapat sepasang teleporter di papan yang jika digunakan akan signifikan mengurangi total jarak tempuh ke target tersebut. • **Perilaku yang Diharapkan:** Bot harus dapat menghitung dan membandingkan jarak langsung dengan jarak via

	teleporter (find_direct_diamond_strategy atau find_best_way_to_base). Jika teleporter lebih efisien, static_goals harus diatur untuk menuju teleporter masuk terlebih dahulu, kemudian ke target akhir setelah keluar dari teleporter. Bot harus berhasil mengeksekusi rencana multi-langkah ini. • **Fokus Pengujian:** Keputusan menggunakan teleporter, pengaturan static_goals dan static_goal_teleport yang benar, dan navigasi yang mulus melalui teleporter menuju tujuan akhir.
Skenario Penghindaran Rintangan Dinamis	• **Kondisi:** Bot sedang bergerak lurus menuju targetnya (misalnya, diamond atau markas). Tiba-tiba, di petak berikutnya yang akan ia injak, terdapat rintangan yang harus dihindari: (a) teleporter yang tidak ingin ia gunakan, (b) diamond merah saat ia membawa 4 diamond biru, atau (c) tombol merah yang bukan tujuannya. • **Perilaku yang Diharapkan:** Fungsi check_obstacle_on_path harus mendeteksi rintangan ini. Bot harus menetapkan static_temp_goals ke posisi menghindar (langkah ke samping atau mundur) dan bergerak ke sana terlebih dahulu, sebelum melanjutkan ke tujuan akhirnya. • **Fokus Pengujian:** Deteksi rintangan, pemilihan manuver penghindaran yang benar, dan kelancaran melanjutkan ke tujuan utama setelah menghindar.
Skenario Perilaku Saat Tidak Ada Target Jelas (Roaming & Stuck)	• **Kondisi:** Papan permainan minim sumber daya atau bot berada di area yang terisolasi. Atau, bot memiliki tujuan tetapi jalur menuju tujuan tersebut terblokir total sehingga get_direction mengembalikan (0,0). • **Perilaku yang Diharapkan:** Jika tidak ada target, bot harus masuk ke mode roaming, bergerak secara sistematis menggunakan self.directions untuk menjelajahi peta. Jika terjebak dan get_direction gagal, bot harus mereset tujuannya (_reset_major_goals, _reset_temp_goals) dan mencoba roaming sebagai upaya terakhir untuk keluar dari situasi buntu. • **Fokus Pengujian:** Aktivasi mode roaming, pola pergerakan roaming, dan mekanisme reset tujuan saat terjebak.
Skenario Interaksi Fitur dan Transisi Prioritas	• **Kondisi:** Sebuah skenario dinamis di mana kondisi permainan berubah, memaksa bot untuk mengganti strateginya. Contoh: 1. Bot memulai dengan mengumpulkan diamond menggunakan strategi blok. 2. Setelah mengumpulkan 1-2 diamond, ia mendeteksi musuh dekat yang membawa banyak diamond (memicu logika agresi karena bot sendiri masih sedikit diamond). 3. Setelah berhasil menyerang musuh dan mendapatkan diamond tambahan (misalnya, total menjadi 3-4 diamond), tiba-tiba waktu permainan menjadi kritis atau musuh lain yang lebih kuat mendekat.

	• **Perilaku yang Diharapkan:** Bot harus menunjukkan transisi yang logis antar strategi. Misalnya, dari pengumpulan blok -> agresi -> kembali ke markas. Prioritas tertinggi (seperti kembali ke markas karena waktu habis atau ancaman besar) harus mengesampingkan strategi dengan prioritas lebih rendah (seperti melanjutkan agresi atau pengumpulan diamond). • **Fokus Pengujian:** Kelancaran transisi antar mode (pengumpulan, agresi, kembali ke base), penegakan hierarki prioritas keputusan, dan adaptasi terhadap perubahan kondisi permainan secara keseluruhan.

4.1.2 Hasil Pengujian dan Analisis

Skenario	Hasil	Penjabaran
Pengumpulan Diamond Dasar & Strategi Blok	Valid	Bot berhasil memilih jalan yang efisien dan kembali ke markas ketika diamond yang dikumpulkan telah memenuhi.
Skenario Prioritas Kembali ke Markas (Kondisi Kritis)	Invalid	Bot tidak kembali ketika waktu akan berakhir.
Skenario Agresi Cerdas terhadap Musuh	Invalid	Bot dirancang fokus untuk mengumpulkan diamond dan kembali ke base ketika diamond yang dikumpulkan penuh.
Skenario Navigasi Menggunakan Teleporter (Efisiensi & Tujuan Jauh)	Valid	Bot dapat menggunakan fitur teleport, tapi tidak rensponsive terhadap tujuan akhir.
Skenario Penghindaran Rintangan Dinamis	Valid	Bot dapat melakukan penghindaran rintangan dinamis untuk mencapai tujuan akhir
Skenario Perilaku Saat Tidak Ada Target Jelas (Roaming & Stuck)	Valid	Bot melakukan penjelajahan ketika menemui titik buntu.
Skenario Interaksi Fitur dan Transisi Prioritas	Invalid	Bot tidak dapat melakukan perubahan strategi dan hanya fokus melakukan pencarian diamond.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan dari pengembangan algoritma greedy untuk menyelesaikan permainan diamonds adalah sebagai berikut:

1. Algoritma greedy cukup efektif dalam menyelesaikan game Diamonds, terbukti dengan rata-rata diamonds yang berhasil dikumpulkan sebanyak 7,6 dalam mode single player.Pendekatan greedy ini memungkinkan bot untuk membuat keputusan berdasarkan langkah yang paling menguntungkan pada setiap tahap permainan, hal tersebut termasuk memanfaatkan teleporter secara efisien dan menghindari obstacle.
2. Kehadiran bot musuh dalam permainan mempengaruhi performa algoritma greedy,dengan rata-rata diamonds yang berhasilkumpulkan turun menjadi 98,6.Hal ini menunjukkan bahwa algoritma yang bekerja baik dalam mode single player belum tentu bekerja dengan baik ketika dihadapkan dengan bot musuh.
3. Algoritma greedy masih memerlukan pengembangan lebih lanjut. Pengembangan ini perlu difokuskan pada penanganan edge case, seperti kesalahan yang terjadi ketika bot terkena tackle dan terjadi error. Dengan melakukan pengembangan lebih lanjut, diharapkan performa dan kestabilan algoritma dapat ditingkatkan

5.2 Saran

Berdasarkan temuan, terdapat beberapa area yang dapat dioptimalkan untuk meningkatkan performa dan stabilitas algoritma greedy pada permainan Diamonds. Saran yang diajukan meliputi:

1. Peningkatan Penanganan Edge Case: Tim pengembang disarankan untuk lebih fokus dalam mengidentifikasi dan mengelola berbagai edge case guna meminimalisir risiko error pada fungsionalitas bot.
2. Optimalisasi Pertimbangan Lawan: Algoritma bot sebaiknya ditingkatkan kemampuannya dalam menganalisis dan mengantisipasi kehadiran serta berbagai kemungkinan strategi yang diterapkan oleh musuh.
3. Ekstensifikasi Pengujian Algoritma: Perlu dilakukan perluasan cakupan dan intensitas pengujian terhadap algoritma bot dalam berbagai skenario permainan. Langkah ini krusial untuk mengungkap potensi kelemahan dan error, sehingga dapat segera dilakukan perbaikan dan ditemukan solusi yang lebih efektif serta optimal.

Dengan penerapan rekomendasi tersebut, diharapkan terjadi peningkatan yang berarti pada kinerja algoritma greedy dalam konteks permainan Diamonds.

LAMPIRAN

Video Demo

Repository Github

DAFTAR PUSTAKA