

LAPORAN TUGAS BESAR 1 IF2211

STRATEGI ALGORITMA

Pemanfaatan Algoritma Greedy dalam pembuatan bot permainan Diamonds



Disusun oleh:
Kelompok 11 - tbd

Maria Flora Renata S. (13522010)

Kristo Anugrah (13522024)

Indraswara Galih Jayanegara (13522119)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

KATA PENGANTAR

Puji syukur kami panjatkan kepada Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya, penulis dapat menyelesaikan laporan ini guna memenuhi Tugas Besar I mata kuliah IF2211 Strategi Algoritma.

Kami juga ingin mengucapkan terima kasih kepada semua pihak yang telah membantu penulis dalam penyelesaian laporan ini, terutama kepada Dr. Nur Ulfa Maulidevi serta Dr. Rinaldi Munir selaku dosen mata kuliah IF2211 Strategi Algoritma dan para asisten dari Lab IRK yang telah membimbing kami dalam penyelesaian makalah ini.

Laporan ini merupakan dokumen pelengkap untuk projek Tugas Besar I IF2211 Strategi Algoritma. Laporan ini menjelaskan landasan teori serta pendekatan *greedy* yang diimplementasikan untuk menghasilkan solusi dari permainan *Diamonds*.

Penulis berharap laporan ini dapat memberikan manfaat dan memotivasi pembaca untuk terus menjelajahi serta mendalami dunia menarik dari Strategi Algoritma.

Bandung, 7 Maret 2024

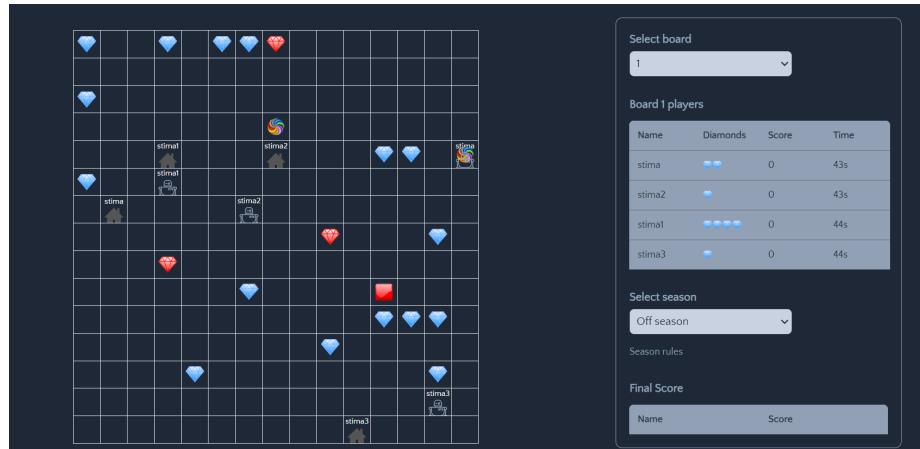
Tim Penulis

BAB I

DESKRIPSI MASALAH

1.1. Diamonds

Diamonds merupakan suatu *programming challenge* yang mempertandingkan bot yang anda buat dengan bot dari para pemain lainnya. Setiap pemain akan memiliki sebuah bot dimana tujuan dari bot ini adalah mengumpulkan *diamond* sebanyak-banyaknya. Cara mengumpulkan *diamond* tersebut tidak akan sesederhana itu, tentunya akan terdapat berbagai rintangan yang akan membuat permainan ini menjadi lebih seru dan kompleks. Untuk memenangkan pertandingan, setiap pemain harus mengimplementasikan strategi tertentu pada masing-masing bot-nya. Pada tugas pertama Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Tentunya mahasiswa harus menggunakan strategi *greedy* dalam membuat bot ini.

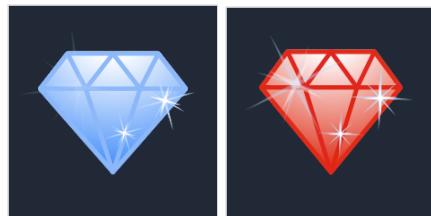


Gambar 1. Antarmuka permainan Diamonds

1.2. Komponen Permainan Diamonds

1.1. *Diamonds*

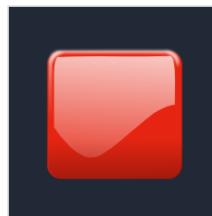
Untuk memenangkan pertandingan, kita harus mengumpulkan *diamond* ini sebanyak-banyaknya dengan melewati/melangkahinya. Terdapat 2 jenis *diamond* yaitu *diamond* biru dan *diamond* merah. *Diamond* merah bernilai 2 poin, sedangkan yang biru bernilai 1 poin. *Diamond* akan di-*regenerate* secara berkala dan rasio antara diamond merah dan biru ini akan berubah setiap *regeneration*.



Gambar 2. *Diamond* biru dan merah

1.2. *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 3. *Red button*

1.3. *Teleporters*

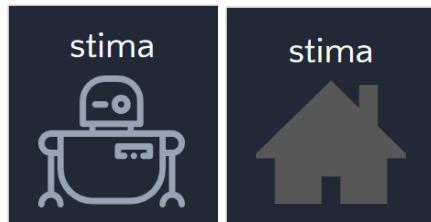
Terdapat 2 *teleporter* yang saling terhubung satu sama lain. Jika bot melewati sebuah *teleporter* maka bot akan berpindah menuju posisi *teleporter* yang lain.



Gambar 4. *Teleporter*

1.4. *Bots and Bases*

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 5. Bot dan Basenya

1.5. *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.

Name	Diamonds	Score	Time
stima	♦♦	0	43s
stima2	♦	0	43s
stima1	♦♦♦♦	0	44s
stima3	♦	0	44s

Gambar 6. Antarmuka *inventory* semua pemain

BAB II

LANDASAN TEORI

2.1. Algoritma *Greedy*

Algoritma Greedy adalah sebuah algoritma yang terkenal pada bidang *computer science* algoritma ini memiliki pendekatan untuk mencari keuntungan sebanyak-banyaknya dengan cara yang rakus. Pendekatan ini berfokus pada pengambilan keputusan pada saat ini untuk mengoptimalkan hasil yang akan didapat pada akhir nanti dengan harapan setiap langkah akan membawa kita ke solusi yang paling optimal. Salah satu tokoh yang terkenal dalam algoritma *Greedy* adalah Edsger Dijkstra yang menemukan algoritma graf yang juga dinamakan . Karakteristik dari algoritma Greedy adalah terdapat *resources* yang bisa diambil dan terdapat *constraint* seperti pada persoalan *knapsack* mengoptimalkan keuntungan dengan keterbatasan *inventory*. Pada tugas besar kali ini juga terdapat *constraint inventory* sebesar 5 dan keterbatasan waktu.

Algoritma *greedy* seringkali tidak menghasilkan solusi yang optimal. Namun, keuntungan dari algoritma ini adalah solusi yang diberikan adalah sebuah *local optimum* yang diberikan dalam waktu komputasi yang cepat, sehingga lebih efisien dibandingkan pendekatan brute force.

2.2. Cara Kerja Program

Program akan menerima informasi kondisi board saat ini dengan status bot, dan kemudian memproses data tersebut untuk mengembalikan gerakan bot selanjutnya sesuai dengan algoritma yang dipilih. Bot pada awalnya akan mempartisi board menjadi data properti, seperti array diamonds, array bots, array teleporter

BAB III

APLIKASI STRATEGI *GREEDY*

3.1. Persoalan Sebagai Elemen Algoritma *Greedy*

Pada tugas besar kali ini memang bisa dikatakan cocok dengan pendekatan algoritma Greedy karena terdapat *score* atau *point* yang mana hal tersebut merupakan hal yang lumrah dalam algoritma *Greedy* seperti pada persoalan *knapsack*. Pada permainan kali ini kita bertindak sebagai bot yang akan mengambil *diamond* sebanyak-banyaknya, tetapi *diamond* tersebut dihitung sebagai *score* apabila bot sudah mengirimkan *diamond* yang berada pada *inventory*-nya ke base dari bot tersebut. Salah satu solusi yang “mudah”, tapi tidak menjamin optimal adalah mengambil diamond terdekat dari bot tersebut, akan tetapi pada solusi ini tidak memperhitungkan aspek-aspek lainnya sehingga hanya berfokus pada diamond terdekat.

Perlu diingat bahwa dengan pendekatan *greedy* ini, hasil gerakan yang didapatkan akan menjadi suatu *local optimum*, bukan *global optimum*. Untuk mencari *global optimum*, diperlukan waktu komputasi yang sangat besar. Maka, digunakanlah algoritma *greedy* yang dapat mencari solusi yang relatif optimal dengan waktu komputasi kecil.

Elemen-elemen dari sebuah algoritma *greedy* untuk menyelesaikan permainan Diamonds dapat dibuat sebagai berikut:

1. Himpunan kandidat: semua kotak/petak di papan permainan.
2. Himpunan solusi: petak yang berisi obyek game yang dapat digunakan (*diamond*, *base*, *teleporter*, *red button*, bot musuh)
3. Fungsi solusi: memastikan himpunan solusi tidak kosong
4. Fungsi seleksi (selection function): memilih *goal* dari antara himpunan solusi sesuai dengan strategi yang digunakan
5. Fungsi kelayakan (feasible): memastikan petak yang akan dilalui untuk sampai ke *goal* tersebut valid/dapat dilalui

3.2. Eksplorasi Alternatif Solusi

1. Diamond (*greedy by value, greedy by distance*)

Pada Algoritma ini kami mencari diamond yang paling dekat yang ada di sekitar bot sehingga nantinya bot akan menuju ke diamond tersebut. Greedy pada solusi ini juga berdasarkan value sehingga mengutamakan red diamond dan jarak terdekat.

2. Enemy (*greedy by value, greedy by distance*)

Pada alternatif solusi bagian ini kami berfokus untuk menyerang musuh yang terdekat atau musuh yang memiliki diamond paling banyak saja. Jadi, kami tidak berfokus untuk mengambil diamond di sekitar, tetapi lebih memilih untuk menyerang musuh-musuh yang ada. Namun, pendekatan ini memiliki efek samping saat proses pengejaran, dimana bot akan terus mengekor bot target tanpa berhasil meng-*tackle*.

3. Base (*greedy by distance*)

Pada alternatif solusi bagian ini kami berfokus mencari diamond yang tidak terlalu jauh dari *home base* dengan tujuan bot bisa pulang dengan lebih aman dan terhindar dari bot-bot milik lawan. Jarak maksimal yang boleh ditempuh oleh bot untuk mengambil diamond paling dekat dengan home base adalah 7 kotak, selain dari itu akan ditolak.

4. Priority(*greedy by priority*)

Pada alternatif solusi ini, setiap elemen permainan seperti base, enemy dan diamond memiliki priority tersendiri. Priority ini kemudian dibandingkan satu sama lain dan akhirnya move dengan priority terbesar akan digunakan.

3.3. Analisis Efisiensi dan Efektivitas Alternatif Solusi

1. Diamond

Algoritma ini adalah yang paling sederhana dan jelas karena dia greedy pada diamond-diamond yang paling dekat dengan bot, setelah inventory-nya berisi lima diamond, maka bot akan langsung

menuju ke base. Akan tetapi, pada solusi ini tidak memperhitungkan kemungkinan-kemungkinan lainnya seperti letak dari *red button*, teleporter, musuh, dan lain-lain. Sehingga algoritma ini belum bisa dikatakan optimal, tapi sudah cukup untuk dibilang merepresentasikan pendekatan dari algoritma Greedy.

2. Enemy

Algoritma ini hanya fokus untuk menyerang musuh dan tidak memperhitungkan objek lain. Bot akan mencari dua jenis musuh, yaitu musuh yang memegang *diamond* paling banyak dan musuh yang paling dekat. Jika bot lebih cepat dari semua bot musuh, strategi ini akan bagus. Akan tetapi, bot ini tidak memiliki perbedaan kecepatan yang cukup signifikan dibandingkan dengan bot lain, sehingga strategi ini tidak optimal.

3. Base

Algoritma ini mirip dengan algoritma Diamond di poin pertama. Bot akan mencari *diamond* yang paling dekat dengan base, dengan jarak maksimal 7 petak. Kecepatan bot dalam mengumpulkan *diamond* dan poin menjadi sangat fluktuatif, karena jika tidak ada *diamond* di dekat base, bot akan diam sampai menemukan *diamond* yang sesuai lagi. Strategi ini juga sudah memenuhi kriteria sebagai algoritma *greedy*, tetapi masih tidak optimal.

4. Priority

Dalam pendekatan priority, bot dapat kembali ke base setelah mengambil sejumlah *diamond*, yang menjadikan pendekatan ini dapat menyimpan *diamond* tanpa berkelana terlalu jauh dari base. Namun, pendekatan ini memiliki waktu komputasi yang relatif lebih lama dibanding pendekatan lain, sehingga kemungkinan bot terkena *tackle* cukup tinggi.

3.4. Strategi *Greedy* yang Dipilih

Setelah mencoba strategi *greedy by diamond*, *enemy*, *base*, dan *priority* tanpa menghindar, serta *priority* dengan menghindar, strategi yang dipilih adalah *greedy by priority* tanpa menghindar. Strategi ini dipilih karena dapat mempertimbangkan lebih banyak hal dibandingkan strategi-strategi lain. Walaupun lebih lambat, *greedy by priority* juga dapat digabungkan dengan berbagai fungsi lain (seperti fungsi menghindari musuh) sehingga bot lebih adaptif terhadap situasi yang krusial seperti berdekatan dengan musuh, dekat dengan *red button*, waktu permainan yang tinggal sedikit pada papan pada setiap saat. Selain itu, menurut 20 tes yang telah kami lakukan, ditemukan bahwa strategi tersebut menghasilkan kemungkinan menang terbesar dengan rata-rata skor terbesar dengan skor 9,7.

Nomor Tes	Diamond	Enemy	Base	Priority tanpa mekanisme menghindar	Priority dengan mekanisme menghindar
1	5	5	0	9	14
2	5	0	6	14	3
3	2	4	7	16	13
4	0	4	3	7	10
5	15	6	10	5	16
6	15	4	8	10	4
7	10	1	0	14	18
8	10	0	0	9	8
9	4	3	12	13	11
10	10	3	0	3	13
11	5	7	0	7	12
12	10	5	8	7	0
13	5	1	0	10	12

14	5	3	7	7	3
15	0	12	12	14	10
16	5	8	6	9	5
17	0	9	2	10	3
18	10	0	7	8	13
19	5	5	15	8	4
20	10	1	11	14	7
Banyak menang	3	0	2	8	8
Rata-rata skor	6,55	4,05	5,7	9,7	8,95
Min. skor	0	0	0	3	0
Max. skor	15	12	15	16	18

Tabel 1. Tabel tes menggunakan 5 bot dengan algoritma berbeda

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi Algoritma *Greedy*

1. Program TBD.py

Fungsi next_move

```
wrong_moves: list gerakan yang dilarang = []
possible_moves: list posisi target yang dipertimbangkan = []
possible_moves.extend(DiamondProcessor)
possible_moves.extend(GoHomeProcessor)
possible_moves.extend(ButtonProcessor)
possible_moves.extend(selfdefenseprocessor)
possible_moves.extend(TeleporterProcessor)
for every move in possible_moves:
    if move[0] == -1:
        wrong_moves[move[1]] = True
possible_moves.sort(descending)
for every move in possible_moves:
    if move is not in wrong_moves and no_obstacle(move, current_position, target):
        return move
```

Fungsi no_obstacle

```
buttons: list button pada board = [obj for obj in board.game_objects if obj is button]
teleporters: list tele pada board = [obj for obj in board.game_objects if obj is tele]
for obstacle in buttons or teleporters:
    if obstacle.position is on the path between current_position and target:
        return False
return True
```

2. Program DiamondProcessor.py

Fungsi process

```
prio_dia = [0, 100, 90, 85, 80, 75, 51, 30, 20, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
diamonds: list diamond pada board = [obj for obj in board.game_objects if obj is diamond]
if diamonds is empty:
    return []
processed: list 2 diamond terdekat dari bot = nearestDiamond(diamonds, bot)
priorities: list processed dengan prioritas = [prio_dia[move[0]] for move in processed]
return priorities
```

Fungsi nearestDiamond

```
diamonds.sort(ascending, key=jarak diamond dari bot)
first_n_elem = diamonds[:2] # mengambil 2 diamond terdekat
return first_n_elem
```

3. Program GoHomeProcessor.py

Fungsi process

```
home_pos: Position = bot.properties.base
diamonds: list = [obj for obj in board.game_objects if obj is diamond]
inv_now: int = bot.properties.diamonds # banyak diamond di inventory
if diamonds is empty:
    return [(inv_now * 100, home_pos)]
secLeft: int = bot.properties.milliseconds_left div 1000
if secLeft - (jarak home ke bot) <= 2:
    return [(501, home_pos)]
likelihood: int = 0
multiplier: int = 0
if inv_now == 1:
    likelihood = 10
    multiplier = 18
elif inv_now == 2:
    likelihood = 40
    multiplier = 20
elif inv_now == 3:
    likelihood = 75
elif inv_now == 4:
    likelihood = 90
elif inv_now == 5:
    likelihood = 100
dist_home: int = jarak home ke bot
dist_dia : int = jarak diamonds terdekat ke bot
priority_home: int = calc_prio(dist_dia, dist_home, multiplier, likelihood)
if priority_home <= 0:
    return []
return [(priority_home, home_pos)]
```

Fungsi calc_prio

```
if likelihood = 100:
    return 500
return likelihood + multiplier * (dist_dia - dist_home)
```

4. Program ButtonProcessor.py

Fungsi process

```
closestDiamonds: list = [obj for obj in board.game_objects if distance obj to bot <= 4]
num_of_dia: int = length of closestDiamonds
button: GameObject = button pada board # dijamin hanya 1
buttonNow: int = length of button
if num_button == 0:
    return []
multiplier: int = 0
```

```

likelihood: int = 0
invNow: int = bot.properties.diamonds
if invNow == 0:
    likelihood = 100
elif invNow == 1:
    likelihood = 70
    multiplier = 20
elif invNow == 2:
    likelihood = 30
    multiplier = 30
elif invNow >= 3:
    likelihood = 10
dist_dia: int = 5
dist_dia: int = jarak button ke bot
if num_of_dia != 0:
    dist_dia: int = jarak diamond terdekat ke bot
return [eval_button(dist_but, dist_dia, likelihood, multiplier)]

```

Fungsi eval_button

```
return likelihood + multiplier * (dist_dia - dist_but)
```

5. Program selfdefenseprocess.py

```

Fungsi process
if banyak bots == 0:
    return []
dist_one: list = list bot musuh yang hanya berjarak 1 dari bot
if dist_one is not empty:
    maxWithDia: GameObject = bot dalam list dist_one yang memiliki diamond
    terbanyak
    return [(-2, maxWithDia.position)]
return []

```

6. Program TeleporterProcessor.py

```

Fungsi process
teleporter: list = [obj for obj in board.game_objects if obj is teleporter]
diamonds: list = [obj for obj in board.game_objects if obj is diamonds]
if teleporter.length == 0:
    return []
tele1, tele2: GameObject = teleporter[0], teleporter[1] # teleporter dijamin 2
minDia1: GameObject = diamond dengan jarak terdekat dari tele1
minDia2: GameObject = diamond dengan jarak terdekat dari tele2
ans: list = []
max1: int = prio_dia[minDia1]
max2 : int = prio_dia[minDia2]
if max1 > 0 and max1 >= max2:

```

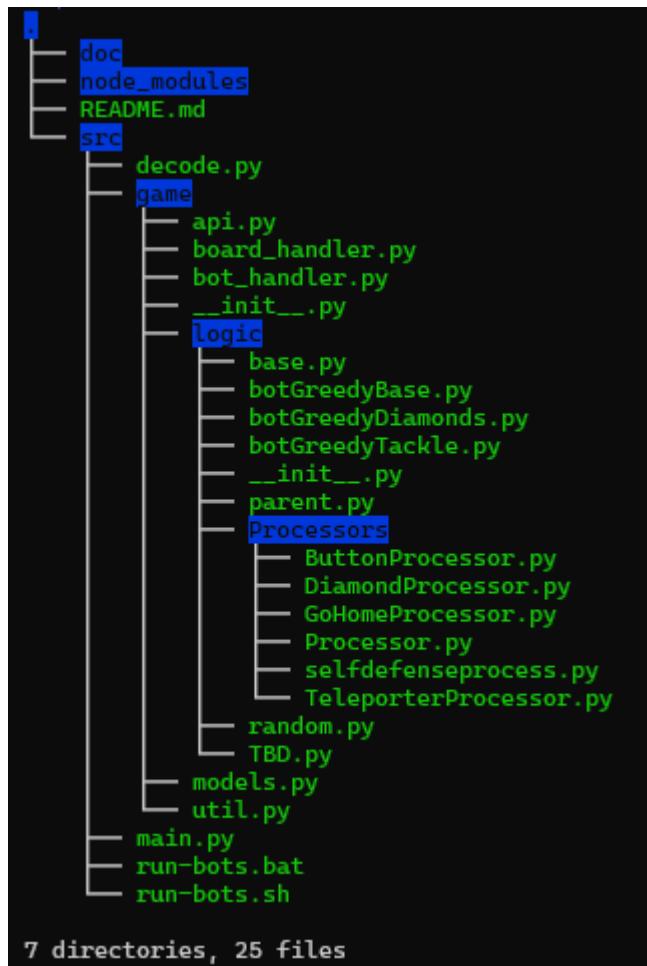
```

ans.append((max1, tele1.position))
if max2 > 0 and max2 > max1:
    ans.append((max2, tele2.position))
prio_h_through_tele1: list = GoHomeProcessor(tele1)
prio_h_through_tele2: list = GoHomeProcessor(tele2)
if length of prio_h_through_tele1 ≠ 0:
    ans.append((prio_h_through_tele1[0][0], tele1.position))
if length of prio_h_through_tele2 ≠ 0:
    ans.append((prio_h_through_tele2[0][0], tele2.position))
return ans

```

4.2. Penjelasan Struktur Data

Struktur dari program yang kami buat sebagai berikut:



Gambar 7. Struktur data program

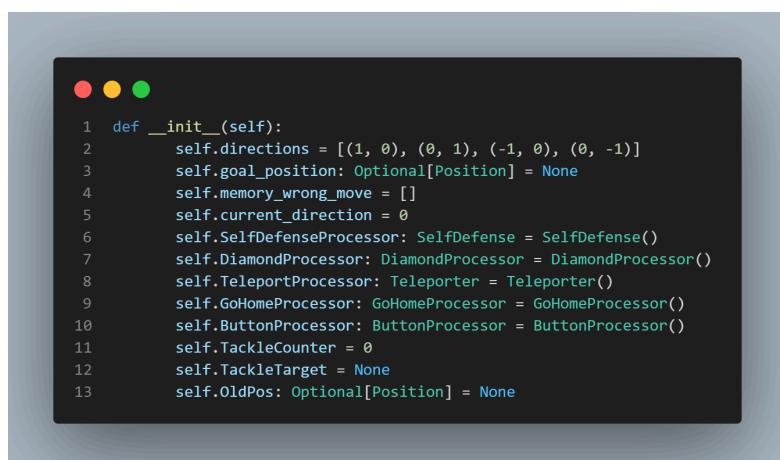
Bot diimplementasikan di TBD.py. Folder Processors berisi file-file implementasi fungsi-fungsi yang diperlukan oleh bot.

Nama file	Keterangan file	Nama fungsi	Deskripsi fungsi
ButtonProcessor.py	Berisi fungsi yang terkait objek <i>red button</i>	get_bot_pos	Mendapatkan posisi bot
		eval_button	Mengevaluasi prioritas memencet tombol dari jarak ke diamond dan ke tombol
		process	Menghitung jarak bot ke <i>diamond</i> terdekat dan ke tombol dan prioritas awal memencet tombol terkait jumlah <i>diamond</i> sekarang. Fungsi akan mengembalikan prioritas setelah pemanggilan fungsi eval_button dan lokasi tombol
DiamondProcessor.py	Berisi fungsi yang terkait objek <i>diamond</i>	nearestDiamond	Mencari beberapa <i>diamond</i> yang paling dekat dengan bot
		process	Memanggil fungsi nearestDiamond dan men-assign prioritas ke setiap <i>diamond</i> hasil
GoHomeProcessor.py	Berisi fungsi untuk menentukan saat kembali ke base	calc_prio	Menghitung prioritas berdasarkan jarak ke <i>diamond</i> , jarak ke base, dan kondisi <i>inventory</i>
		process	Mengecek jarak ke <i>diamond</i> terdekat, base, dan kondisi <i>inventory</i> , lalu memanggil fungsi

			calc_prio dan mengeluarkan prioritas dan pilihan tujuan
Processor.py	<i>Parent class</i> untuk semua processor	-	-
selfdefenseprocess.py	Berisi fungsi untuk mempertahankan diri dari bot musuh	process	Mengecek semua musuh yang berjarak 1 petak dari bot, lalu mengeluarkan prioritas dan posisi musuh dengan <i>diamond</i> paling banyak.
TeleporterProcessor.py	Berisi fungsi yang terkait objek <i>teleporter</i>	process	Membandingkan jarak <i>teleporter</i> ke bot dan ke <i>diamond</i> terdekatnya dengan jarak bot ke <i>diamond</i> terdekat sekarang, lalu menghitung apakah sebaiknya melalui <i>teleporter</i> atau tidak.

Tabel 2. Tabel file-file serta fungsinya pada struktur data

Berikut adalah struktur atribut utama dari bot yang kami gunakan:



```

1 def __init__(self):
2     self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
3     self.goal_position: Optional[Position] = None
4     self.memory_wrong_move = []
5     self.current_direction = 0
6     self.SelfDefenseProcessor: SelfDefense = SelfDefense()
7     self.DiamondProcessor: DiamondProcessor = DiamondProcessor()
8     self.TeleportProcessor: Teleporter = Teleporter()
9     self.GoHomeProcessor: GoHomeProcessor = GoHomeProcessor()
10    self.ButtonProcessor: ButtonProcessor = ButtonProcessor()
11    self.TackleCounter = 0
12    self.TackleTarget = None
13    self.OldPos: Optional[Position] = None

```

Gambar 8. Atribut utama di program TBD.py

Penjelasan setiap atribut sebagai berikut:

1. Directions: arah gerak yang bisa dilakukan oleh bot
2. Goal_position: posisi akhir yang sedang dituju oleh bot
3. Memory_wrong_move: gerakan yang dilarang, dapat disebabkan karena jarak musuh yang terlalu dekat atau gerakan *out of bounds*.
4. current_direction:
5. SelfDefenseProcessor, DiamondProcessor, TeleportProcessor, GoHomeProcessor, ButtonProcessor: digunakan untuk menyambungkan setiap processor ke bot utama.
6. TackleCounter: menghitung jumlah percobaan *tackle* yang dilakukan secara berturut-turut
7. TackleTarget: menyimpan bot yang sedang/baru saja menjadi *target* percobaan *tackle*.
8. OldPos: menyimpan posisi bot di pergerakan sebelumnya

4.3. Analisis dari Desain Solusi Algoritma

Algoritma ini memiliki kompleksitas waktu $O(n\log n)$, dengan n merupakan banyak target yang di-*consider* pada suatu waktu. Kompleksitas ini muncul karena proses *sorting* yang dilakukan setiap kali program utama main.py memanggil fungsi next_move. Fungsi *sort* yang dipakai merupakan fungsi bawaan dari Python yang mengimplementasikan algoritma *hybrid* antara *merge sort* dan *insertion sort*, yang memiliki kompleksitas rata-rata $O(n\log n)$.

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Kesimpulan dari tugas besar kali ini adalah sulit untuk mencari algoritma greedy yang optimal yang ada hanya solusi yang mendekati ke-optimalan tersebut. Hal ini dikarenakan banyak hal yang harus diperhitungkan dari pengambil keputusan dari bot diantaranya adalah posisi diamond, value diamond, posisi bot musuh, posisi teleporter, posisi base, inventory yang terbatas, dan waktu yang terbatas.

Hasil yang optimal secara sempurna sulit dicapai apalagi dengan adanya batasan bot harus bergerak dalam satu detik yang membuat bot harus cepat untuk mengambil keputusan hal ini mengakibatkan pemrosesan yang dilakukan oleh bot tidak bisa terlalu kompleks. Jadi, bot yang dibuat hanya memperhitungkan kemungkinan-kemungkinan yang paling krusial yang akan terjadi di dalam permainan.

5.2. Saran

Menurut kami hal yang perlu dilakukan untuk orang-orang yang ingin mencoba membuat apa yang telah kami kembangkan adalah dengan eksplorasi lebih banyak kombinasi-kombinasi greedy lainnya. Bisa saja pembaca menemukan algoritma greedy yang lebih sangkil dari yang tim penulis telah buat. Selain itu, pembaca juga bisa menggunakan bahasa pemrograman lain yang lebih cepat daripada python sehingga mungkin dengan algoritma yang sama, tetapi menghasilkan hasil yang lebih cepat.

LAMPIRAN

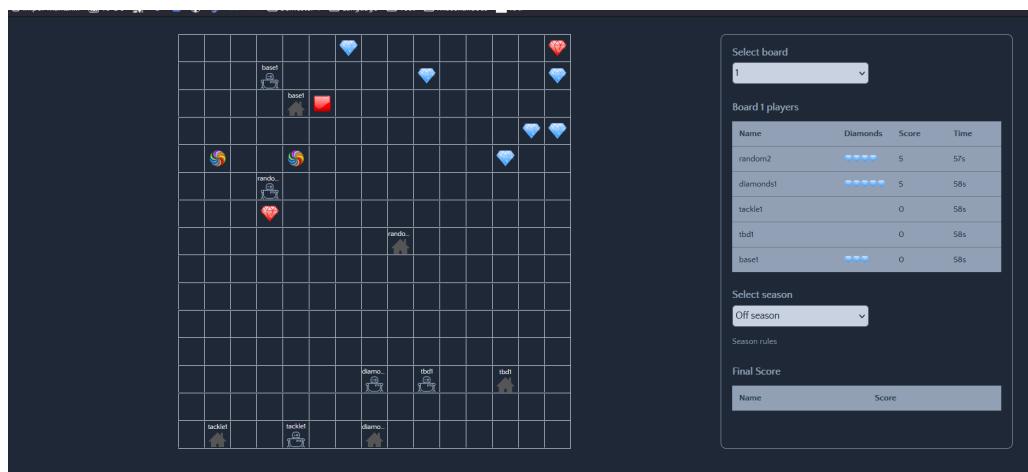
Link Github:

https://github.com/florars/Tubes1_tbd

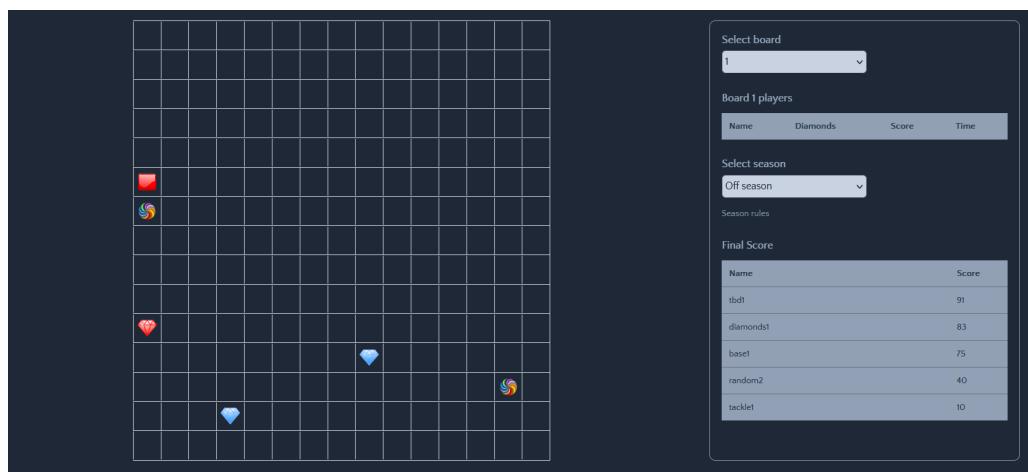
Link Video Bonus:

- Youtube: <https://youtu.be/NYRvjEj-XAI>
- Drive :

Tubes 1 STIMA: Bot Permainan Diamonds Kelompok TBD.mp4



Gambar 9. Tampilan *frontend* kompetisi dengan 5 bot



Gambar 10. Score bot saat *time sleep* diatur 0.1 dan total permainan 60 detik

```
1 def process(self, board_bot: GameObject, board: Board) -> list[tuple[int, Position]]:
2     funcGO = lambda x, y: abs(x.x - y.x) + abs(x.y - y.y)
3     closestDiamonds = list(filter(lambda g: g.type == "DiamondGameObject" and funcGO(g.position, board_bot.position) <= self.find_rad, board.game_objects))
4     num_of_dia = len(closestDiamonds)
5     buttonNow: list[GameObject] = list(filter(lambda g: g.type == "DiamondButtonGameObject" and funcGO(g.position, board_bot.position) <= self.find_rad, board.game_objects))
6     num_button: int = len(buttonNow)
7     if num_button == 0:
8         return []
9     multiplier = 0
10    likelihood = 0
11    invNow = board_bot.properties.diamonds
12    if invNow == 0:
13        likelihood = 100
14    elif invNow == 1:
15        likelihood = 70
16        multiplier = 20
17    elif invNow == 2:
18        likelihood = 30
19        multiplier = 30
20    elif invNow >= 3:
21        likelihood = 10
22    dist_dia = 5
23    dist_but = funcGO(buttonNow[0].position, board_bot.position)
24    if num_of_dia != 0:
25        nearest_dia = min(closestDiamonds, key=lambda g: funcGO(g.position, board_bot.position))
26        dist_dia = funcGO(nearest_dia.position, board_bot.position)
27    return [(self.eval_button(dist_but, dist_dia, likelihood), buttonNow[0].position)]
```

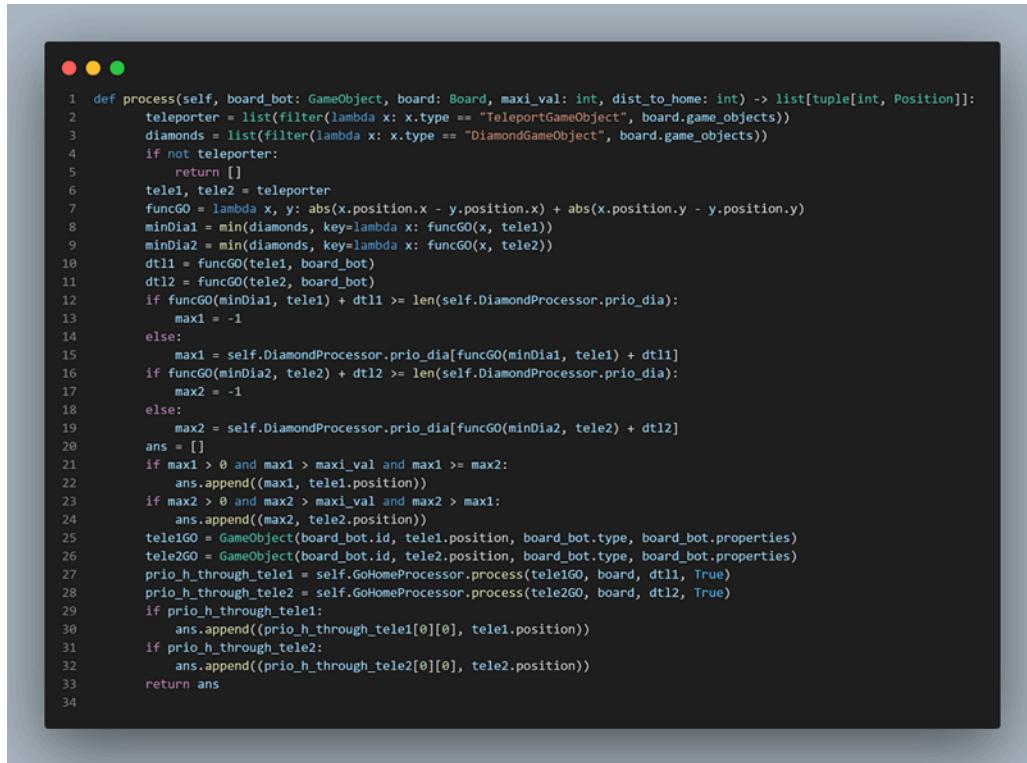
Gambar 11. Kode pada fungsi process di file ButtonProcessor.py

```
1 def process(self, board_bot: GameObject, board: Board) -> Optional[list[tuple[int, Position]]]:
2     diamonds = list(filter(lambda x: x.type == "DiamondGameObject", board.game_objects))
3     if not diamonds:
4         return []
5     if board_bot.properties is not None:
6         if board_bot.properties.diamonds == 4:
7             diamonds = list(filter(lambda x: x.properties.points == 1, diamonds))
8     processed: list[GameObject] = self.nearestDiamond(diamonds, board_bot)
9     return [(self.prio_dia[abs(game_object.position.x - board_bot.position.x) + abs(game_object.position.y - board_bot.position.y)], game_object.position) for game_object in processed]
10    if abs(game_object.position.x - board_bot.position.x) + abs(game_object.position.y - board_bot.position.y) < len(self.prio_dia)]
11
12
```

Gambar 12. Kode pada fungsi process di file DiamondProcessor.py

```
1 def process(self, board_bot: GameObject, board: Board):
2     props = board_bot.properties
3     if len(board.bots) == 1:
4         return []
5     func_dist = lambda x, y: abs(x.position.x - y.position.x) + abs(x.position.y - y.position.y)
6     dist_one = list(filter(lambda x: func_dist(x, board_bot) == 1 and (x.position.x != x.properties.base.x or x.position.y != x.properties.base.y), board.bots))
7     if dist_one:
8         maxWithDia = max(dist_one, key=lambda x: x.properties.diamonds)
9         return [(-2, maxWithDia)]
10    return []
```

Gambar 13. Kode pada fungsi process di file selfdefenseprocess.py



```
1 def process(self, board_bot: GameObject, board: Board, maxi_val: int, dist_to_home: int) -> list[tuple[int, Position]]:
2     teleporter = list(filter(lambda x: x.type == "TeleportGameObject", board.game_objects))
3     diamonds = list(filter(lambda x: x.type == "DiamondGameObject", board.game_objects))
4     if not teleporter:
5         return []
6     tele1, tele2 = teleporter
7     funcGO = lambda x, y: abs(x.position.x - y.position.x) + abs(x.position.y - y.position.y)
8     minDia1 = min(diamonds, key=lambda x: funcGO(x, tele1))
9     minDia2 = min(diamonds, key=lambda x: funcGO(x, tele2))
10    dt11 = funcGO(tele1, board_bot)
11    dt12 = funcGO(tele2, board_bot)
12    if funcGO(minDia1, tele1) + dt11 >= len(self.DiamondProcessor.prio_dia):
13        max1 = -1
14    else:
15        max1 = self.DiamondProcessor.prio_dia[funcGO(minDia1, tele1) + dt11]
16    if funcGO(minDia2, tele2) + dt12 >= len(self.DiamondProcessor.prio_dia):
17        max2 = -1
18    else:
19        max2 = self.DiamondProcessor.prio_dia[funcGO(minDia2, tele2) + dt12]
20    ans = []
21    if max1 > 0 and max1 > maxi_val and max1 >= max2:
22        ans.append((max1, tele1.position))
23    if max2 > 0 and max2 > maxi_val and max2 > max1:
24        ans.append((max2, tele2.position))
25    tele1GO = GameObject(board_bot.id, tele1.position, board_bot.type, board_bot.properties)
26    tele2GO = GameObject(board_bot.id, tele2.position, board_bot.type, board_bot.properties)
27    prio_h_through_tele1 = self.GoHomeProcessor.process(tele1GO, board, dt11, True)
28    prio_h_through_tele2 = self.GoHomeProcessor.process(tele2GO, board, dt12, True)
29    if prio_h_through_tele1:
30        ans.append((prio_h_through_tele1[0][0], tele1.position))
31    if prio_h_through_tele2:
32        ans.append((prio_h_through_tele2[0][0], tele2.position))
33    return ans
34
```

Gambar 14. Kode pada fungsi process di file TeleporterProcessor.py

DAFTAR PUSTAKA

- Levitin, A. (2011). Introduction to the Design and Analysis of Algorithms (3rd ed.). Pearson.
- Barron, A. R., Cohen, A., Dahmen, W., and DeVore, R. A. (2008). Approximation and learning by greedy algorithms. Institute of Mathematical Statistics.
- Cormen et al. (2009). Introduction to Algorithms 3rd ed. Massachusetts Institute of Technology.
- Wolfram, M. (2020, April 7). Greedy algorithms. FreeCodeCamp.
<https://www.freecodecamp.org/news/greedy-algorithms/>
- javatpoint. (n.d.). Greedy algorithms. JavaTpoint.
<https://www.javatpoint.com/greedy-algorithms>
- Rinaldi Munir (2021). Algoritma Greedy (Bagian 1).
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf)