LAPORAN TUGAS BESAR I

IF2211 STRATEGI ALGORITMA

PEMANFAATAN ALGORITMA *GREEDY* DALAM PEMBUATAN *BOT* PERMAINAN DIAMONDS

KELOMPOK AKIONG

ANGGOTA :

1. 10023500 MIFTAHUL JANNAH

2. 13522019 WILSON YUSDA

3. 13522045 FAREL WINALDA

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

DAFTAR ISI 2
DAFTAR GAMBAR 4
DAFTAR TABEL 5
BAB I DESKRIPSI MASALAH 6
BAB II LANDASAN TEORI 9
Algoritma Greedy 9
Game Engine Etimo Diamonds 10
Cara Kerja Program 11
Cara Menjalankan Game Engine Beserta Bot 11
Cara mengimplementasikan *bot* 14
BAB III APLIKASI STRATEGI GREEDY 15
Alternatif Greedy 15
Konsep Pencarian Red Button 15
Konsep Penanganan Portal 15
Variabel Tambahan 15
Alternatif Greedy by Shortest Distance 16
Alternatif Greedy by Shortest Distance Concerning Highest Diamond Value 17
Alternatif Greedy by Closest to Base 19
Alternatif Greedy by Highest Density Value 20
Alternatif Greedy by Clustering Efficiency 22
Strategi greedy yang diimplementasikan 23
BAB IV IMPLEMENTASI DAN PENGUJIAN 27
Class dan Struktur Data dalam Program 27
Implementasi Program dalam Pseudocode 29
Analisis Algoritma Greedy 39
Tes Kemampuan Pencarian Clustering 40

Tes Kemampuan Kembali ke Base 41
Tes Kemampuan Portal 42
Test Case Posisi Base diujung map dan hanya ada 1 portal 43
Test Case Posisi Base diujung ditengah map dan hanya ada 1 portal 44
Test Case Posisi Base Di ujung dengan 3 portal 45
Test Case Posisi Base Ditengah dengan 3 portal 47
Kesimpulan Analisis 48
BAB V KESIMPULAN DAN SARAN 49
KESIMPULAN 49
SARAN 49
LAMPIRAN 50
Pranala Repository: 50
Pranala Video Youtube: 50
DAFTAR PUSTAKA 51

DAFTAR GAMBAR

Gambar 1. 1 Tampilan game Diamonds.....6

Gambar 3. 1 Final Score Multiplayer 1 25
Gambar 3. 2 Final Score Multiplayer 2 26
Gambar 3. 3 Final Score Multiplayer 3 26

Gambar 4. 1 Tes Kemampuan Clustering (Before).....40
Gambar 4. 2 Tes Kemampuan Clustering (After) 40
Gambar 4. 3 Tes Kemampuan Return to Base (Before) 41
Gambar 4. 4 Tes Kemampuan Return to Base (After).....41
Gambar 4. 5 Tes Kemampuan Portal (Before).....42
Gambar 4. 6 Tes Kemampuan Portal (After) 42
Gambar 4. 7 Test Case 1 Portal Base diujung (Before) 43
Gambar 4. 8 Test Case 1 Portal Base diujung (After).....43
Gambar 4. 9 Test Case 1 Portal Base ditengah (Before).....44
Gambar 4. 10 Test Case 1 Portal Base ditengah (After) 45
Gambar 4. 11 Test Case 3 Portal Base diujung (Before) 46
Gambar 4. 12 Test Case 3 Portal Base diujung (After).....46
Gambar 4. 13 Test Case 3 Portal Base ditengah (Before).....47
Gambar 4. 14 Test Case 3 Portal Base ditengah (After) 47

DAFTAR TABEL

Tabel 3. 1 Pengukuran Diamond Rata Rata Secara Solo 24

BAB I
DESKRIPSI MASALAH

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. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah.

Gambar 1. 1 Tampilan game Diamonds

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*.

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.

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.

4. Bots and Bases

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

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.

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

Algoritma Greedy

Algoritma greedy merupakan sebuah pendekatan dalam pemrograman dinamis yang memilih solusi terbaik pada setiap langkahnya tanpa mempertimbangkan konsekuensi di masa depan. Pendekatan ini sering digunakan untuk menyelesaikan berbagai masalah optimasi, seperti masalah penjadwalan, penentuan rute, dan lain-lain.

Dalam algoritma greedy, pilihan yang tampaknya terbaik pada saat itu dibuat dengan harapan bahwa pilihan ini akan mengarah pada solusi global yang optimal. Setelah membuat pilihan, algoritma greedy tidak pernah mempertimbangkan pilihan tersebut kembali, artinya algoritma ini tidak memiliki mekanisme backtracking.

Untuk menentukan pilihan yang optimal pada setiap langkah, algoritma greedy menggunakan kriteria tertentu, yang sering kali berupa fungsi untuk memaksimalkan atau meminimalkan nilai tertentu. Masalah yang dapat diselesaikan dengan algoritma greedy sering memiliki substruktur optimal, yang berarti solusi optimal untuk masalah tersebut dapat dibangun dari solusi optimal untuk submasalahnya.

Sifat pilihan serakah menyatakan bahwa pilihan lokal yang dibuat oleh algoritma greedy dapat diintegrasikan ke dalam solusi global tanpa perlu mempertimbangkan kembali pilihan sebelumnya. Contoh masalah yang sering diselesaikan dengan algoritma greedy adalah masalah coin change, masalah fractional knapsack, dan masalah minimum spanning tree (MST). Walaupun algoritma greedy sederhana dan cepat, tidak semua masalah optimasi dapat diselesaikan secara efektif dengan pendekatan ini. Algoritma greedy dapat menghasilkan solusi yang suboptimal atau bahkan salah untuk masalah tertentu.

Terdapat elemen-elemen dalam algoritma greedy sebagai berikut.

a. Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap langkah.
b. Himpunan solusi, S : berisi kandidat yang sudah dipilih
c. Fungsi solusi : menentukan apakah himpunan yang dipilih sudah memberikan solusi
d. Fungsi seleksi (selection function): memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.
e. Fungsi kelayakan (feasible): memeriksa apakah kandidat yang dipilih layak atau tidak untuk dimasukkan ke dalam himpunan solusi.

f. Fungsi obyektif: untuk memaksimumkan atau meminimumkan

Game Engine Etimo Diamonds

Game Engine adalah sistem perangkat lunak yang dimanfaatkan untuk mengembangkan atau menciptakan sebuah *game*. *Game engine* merupakan *library* yang digunakan untuk membuat *game. Game engine etimo diamonds* merupakan *game* yang berasal dari permainan dahulu yaitu catur *Go*/Baduk. Pada permainan tersebut kita diminta untuk mengepung batu lawan sehingga kita dapat mengambil batu lawan yang dikepung. Tentu hal ini membuat kita ingin mengambil atau mengepung lawan sebanyak-banyaknya dan hal ini pula yang menjadi dasar dari pembuatan *Game engine etimo diamonds*, yaitu ingin mengambil atau mendapatkan *diamonds* sebanyak-banyaknya. *Game engine etimo diamonds* diciptakan untuk menambah tingkat kesulitan permainan dengan menambahkan beberapa *fitur* dan aturan yang membuat permainan menjadi lebih menarik.

Game engine etimo diamonds tetap mempertahankan latar catur pada permainan Baduk namun yang berubah adalah pada catur permainanya, jika pada permainan Baduk catur yang digunakan hanya berwarna hitam dan putih yang berjumlah 181 batu hitam dan 180 batu putih dan permainan ini hanya terbatas pada dua pemain saja namun *Game engine etimo diamonds* dapat dimainkan oleh beberapa orang. *Game* ini sedikit berbeda dengan catur Baduk, jika permainan Baduk akan memenangkan banyak batu lawan, *game* ini akan dimenagkan berdasarkan banyaknya *diamonds* yang didapat. Bukan hanya itu game ini juga memiliki batasan waktu dan permainan tersebut mengandalkan *bot* yang telah dirancang dengan algorima Greedy untuk memenangkan permainan. Dapat dikatakan bahwa game ini merupakan pertarungan antar *bot*.

Game engine etimo diamonds merupakan pertarungan antar *bot*, dengan batasan waktu, serta beberapa aturan baru. Aturan tersebut adalah jika berhasil mendapatkan *Diamonds* merah bernilai 2 poin, sedangkan yang biru bernilai 1 poin. Terdapat juga *red button,* ketika *red button* dilewati maka semua *diamond* akan di-*generate* kembali pada *board* dengan posisi acak. Posisi *red button* ini juga akan berubah secara acak jika *red button* ini dilangkahi. Terdapat juga 2 *teleporter* yang terhubung satu sama lain, *teleport* tersebut berbebntuk cakram warna-warni, jika *teleporter* tersebut dilalui maka *bot* akan berpindah. Hal yang perlu diperhatikan pada game ini adalah sebuah *base. Base* ini berfungsi untuk menyimpan *diamonds* yang telah kita dapatkan, apabila *diamonds* disimpan ke *base* maka *score* akan bertambah sebanyak *diamonds* yang dibawa. *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.

Cara Kerja Program

Program game Diamonds beroperasi dengan urutan langkah-langkah berikut:

1. Inisialisasi Game: Engine permainan diaktifkan dan menjalankan pertandingan di alamat yang ditentukan, seperti localhost:8082 untuk lingkungan lokal.
2. Koneksi Bot: Engine menunggu sampai semua bot pemain terhubung sebelum memulai pertandingan. Proses logging dilakukan untuk merekam kejadian penting selama permainan.
3. Pengaturan Jumlah Bot: Jumlah bot yang berpartisipasi dalam satu pertandingan ditentukan oleh konfigurasi dalam file "appsettings.json".
4. Mulai Pertandingan: Pertandingan dimulai setelah semua bot terhubung dan jumlahnya sesuai dengan pengaturan yang ditetapkan.
5. Interaksi Bot dan Engine: Bot pemain berinteraksi dengan engine, menerima event dari engine dan mengirimkan respons berupa aksi yang akan dilakukan oleh bot.
6. Pelaksanaan Game: Game berlangsung sampai kondisi berakhirnya pertandingan terpenuhi, seperti batas waktu atau kriteria kemenangan. Hasil pertandingan dicatat dalam file JSON untuk analisis lebih lanjut.
7. Note: Program ini mengelola interaksi antara engine dan bot pemain, memfasilitasi pelaksanaan pertandingan Diamonds, dan mendokumentasikan hasilnya.

Cara Menjalankan Game Engine Beserta Bot

Setelah mengetahui mengenai game engine, maka berikut adalah cara untuk menjalankan game angine berserta bot nya.

1. Cara menjalankan game angine

a. *Requirement* yang harus di-install

- Node.js
  Node.js dapat diinstal pada link berikut: (https://nodejs.org/en). Node.js berfungsi untuk menghubungkan terminal ke website.
- Docker desktop
  Docker desktop berfungsi untuk setup local database.
  Aplikasi ini dapat diunduh pada (https://www.docker.com/products/docker-desktop/)

- Yarn

  Yarn berfungsi sebagai penghubung, yarn dapat diinstal dengan

```
npm install --global yarn
```

b. Instalasi dan konfigurasi awal

1. Download source code (.zip) pada release game engine.
2. Extract file zip tersebut, lalu masuk ke folder hasil extractnya dan buka aplikasi terminal
3. Masuk ke root directory dari project (sesuaikan dengan nama rilis terbaru)

```
cd tubes1-IF2110-game-engine-1.1.0
```

4. Install dependencies menggunakan Yarn

```
yarn
```

5. Setup default environment variable dengan menjalankan script berikut

   Untuk Windows:

```
./scripts/copy-env.bat
```

   Untuk Linux / (possibly) macOS

```
chmod +x ./scripts/copy-env.sh
./scripts/copy-env.sh
```

6. Setup local database dengan membuka aplikasi docker desktop terlebih dahulu, lalu jalankan command berikut di terminal.

```
docker compose up -d database
```

   Lalu jalankan script berikut:

   Untuk Windows :

```
./scripts/setup-db-prisma.bat
```

Untuk linux / (possibly) macOS :

```
chmod +x ./scripts/setup-db-prisma.sh
./scripts/setup-db-prisma.sh
```

c. Jalankan perintah berikut untuk melakukan build frontend dari game-engine

```
npm run build
```

d. Jalankan perintah berikut untuk memulai game-engine

```
npm run start
```

Frontend dapat dikunjungi melalui http://localhost:8082/. Perlu diingat bahwa untuk menjalankan game kedua kalinya, hanya perlu menbangun frontend dan menyalakan game engine sesuai langkah no c dan d diatas.

2. Cara menjalankan *bot*

a. Instalasi dan konfigurasi awal

1) Download source code (.zip) pada release bot starter pack
2) *Extract* file zip tersebut, lalu masuk ke folder hasil extractnya dan buka terminal
3) Masuk ke *root directory* dari project (sesuaikan dengan nama rilis terbaru)

```
cd tubes1-IF2110-bot-starter-pack-1.0.1
```

4) Install dependencies menggunakan pip

```
pip install -r requirements.txt
```

Untuk menjalankan satu bot (pada contoh ini, kita menjalankan satu bot dengan logic yang terdapat pada file game/logic/random.py)

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

Untuk menjalankan beberapa bot sekaligus (pada contoh ini, kita menjalankan 4 bot dengan logic yang sama, yaitu game/logic/random.py)

→ Untuk windows :

```
./run-bots.bat
```

→ Untuk Linux / (possibly) macOS :

```
./run-bots.sh
```

NOTE:

1. Jika kalian menjalankan beberapa bot, pastikan setiap email dan nama unik
2. Email bisa apa saja asalkan mengikuti sintaks email yang benar, tidak harus email yang terdaftar (misal stima22@email.com)
3. Nama dan password bisa apa saja tanpa spasi

Cara mengimplementasikan *bot*

1) Buatlah folder baru pada direktori /game/logic (misalnya mybot.py)
2) Buatlah kelas yang meng-inherit kelas BaseLogic, lalu implementasikan constructor dan method next_move pada kelas tersebut
3) Import kelas yang telah dibuat pada main.py dan daftarkan pada dictionary CONTROLLERS
4) Jalankan program seperti step c pada bagian 2 (sesuaikan argumen logic pada command/script tersebut menjadi nama bot yang telah terdaftar pada CONTROLLERS). Anda bisa menjalankan satu bot saja atau beberapa bot menggunakan .bat atau .sh script.

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

BAB III
APLIKASI STRATEGI GREEDY

Alternatif Greedy

Dalam pembuatan bot berbasis Algoritma Greedy, kami memakai beberapa pendekatan greedy. Beberapa pendekatan ini tidak dilampirkan dalam bentuk *bot greedy version* dalam github, namun merupakan tahapan eksplorasi untuk membentuk bot final kami. Namun, untuk setiap alternatif greedy, terdapat sebuah pengaturan fitur yang telah ditetapkan yang tidak menganggu konsep greedy, namun lebih bergantung pada efisiensi jarak serta mekanik permainan. Konsep konsep tersebut diantaranya:

Konsep Pencarian Red Button

Red button berfungsi untuk mengubah posisi *diamonds* dalam *map* sehingga menurut kami menargetkan red button secara terus menerus akan bersifat sangat merugikan. Oleh karena itu, konsep pencarian red button akan berada di luar konsep greedy dalam pencarian diamond, sehingga bot akan menargetkan red button hanya jika posisi red button bersifat dekat. Alasan dimana kami hanya memperhitungkan jarak dan bukan dari alternatif greedy lainnya akan dijelaskan dibawah yakni karena memperhitungkan jika jumlah diamond dalam peta tergolong sedikit, atau bahkan dalam kondisi tertentu tidak menguntungkan, solusi yang terbaik yakni berupa menekan red button dan berharap posisi akan menjadi lebih menguntungkan.

Konsep Penanganan Portal

Portal berfungsi memindahkan sebuah bot ke posisi lain dengan *pair id* yang sama. Portal sendiri juga tidak digabungkan dengan pengaplikasian greedy dibawah dimana untuk setiap alternatif greedy kami akan mengembalikan sebuah target efektif, dan untuk target efektif tersebut, akan sudah terhitung jarak efektifnya. Setiap alternatif greedy akan mengecek apakah target greedy yang mereka kembalikan dapat dituju lebih cepat dengan portal dan jika iya, maka target akan menjadi portal tersebut. Setelah bot dipindahkan dari 1 portal ke tujuan portal, tidak peduli dengan alternatif greedy apapun, bot dipastikan untuk menargetkan diamonds hasil greedy sebelumnya, sehingga membuat konsep penanganan portal ini tidak menganggu konsep greedy pencarian diamonds.

Variabel Tambahan

Dalam pengetesan bot, terdapat beberapa variabel yang dapat menganggu pengukuran efektivitas bot. Beberapa diantaranya:

1. Base yang terlalu jauh

   Base yang terlalu jauh mengakibatkan kemungkinan *collision* lebih tinggi disebabkan karena berbagai algoritma greedy memandu bot untuk ke suatu titik sehingga akan sangat sulit untuk kembali ke base.
2. Base yang berdampingan dengan base lawan

   Base yang berdampingan juga dapat meningkatkan resiko *collision,* terutama jika keduanya memiliki target diamonds serupa sehinggauntuk kembali ke base akan ada kemungkinan *collision.*

Setelah membuat konsep fitur dan penjelasan variabel selain diamonds diatas, barulah kami mengimplementasikan berbagai konsep greedy dan melakukan berbagai eksplorasi guna mencari konsep terbaik:

Alternatif Greedy by Shortest Distance

Greedy by shortest distance merupakan pendekatan alternatif greedy dimana algoritma greedy ini memfokuskan pada mencari diamonds yang paling dekat dengan dirinya, tidak peduli nilai dari diamond tersebut. Pendekatan ini merupakan basis awal kami yang kemudian mulai dikembangkan menjadi berbagai konsep lainnya.

a. Pemetaan Elemen Greedy
   - Himpunan Kandidat: Himpunan diamond yang tersedia pada board, yang meliputi diamond merah yang bernilai 2 poin dan diamond biru yang bernilai 1 poin.
   - Himpunan Solusi: Himpunan diamond yang terpilih.
   - Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
   - Fungsi Seleksi: Memilih diamond dengan jarak terdekat dari bot
   - Fungsi kelayakan: Memeriksa jumlah diamond yang telah diperoleh tidak melebihi *max inventory* pada setiap iterasi, dan apabila jumlah diamond telah mencapai *max inventory*, bot akan diarahkan untuk kembali ke base untuk menyimpan diamond.
   - Fungsi obyektif: Jumlah poin diamond yang diperoleh maksimum.

b. Analisis Efisiensi Solusi

   Efisiensi dari logika ini terutama bergantung pada fungsi findDistance, yang menghitung jalur terpendek ke target, dengan mempertimbangkan pergerakan langsung dan teleportasi. Algoritma ini mengulang melalui semua berlian dan teleporter di papan, mengakibatkan kompleksitas waktu yang sebanding secara linier dengan jumlah berlian dan teleporter. Namun, dalam skenario terburuk di mana papan dipenuhi dengan objek-

objek ini, kompleksitas waktu bisa mendekati $O(n^2)$, di mana n adalah jumlah total objek permainan. Dalam praktiknya, kinerja dari logika ini seharusnya dapat diterima untuk papan berukuran kecil hingga sedang, tetapi mungkin menurun pada papan yang lebih besar dengan banyak berlian dan teleporter.

c. Analisis Efektivitas Solusi

Solusi ini merupakan pendekatan yang paling logis dan tepat untuk dijadikan konsep awal dalam pembuatan bot pencarian diamond. Efektivitas bot ini terlihat pada cara pengambilan diamond yang tidak mubajir gerakan dan menerapkan konsep clusteringnya sendiri. Namun, alternatif ini akan tidak efektif dengan beberapa alasan dibawah:

1. Pendekatan yang terlalu umum sehingga seringkali bot yang berdekatan menargetkan diamonds yang sama dan menyebabkan *collision* yang membahayakan bot, terutama jika sedang memiliki diamonds yang banyak.
2. Bot seringkali bersifat tidak efektif dalam mengumpulkan diamonds, terutama jika salah satu jarak diamond hanya lebih jauh sedikit saja, namun memiliki poin lebih besar.
3. Algoritma memandu bot semakin jauh dari base sehingga waktu sering terbuang sia-sia untuk kembali ke base.
4. Posisi diamonds terkadang terkumpul pada suatu titik yang jauh dimana alternatif ini pasti tidak akan menargetkan hal tersebut karena algoritma hanya melihat jarak diamond terdekat.
5. Diamond dapat bernilai -1 dari *max nventory* yang membuat bot mungkin menghindari diamond merah sehingga menghabiskan lebih banyak waktu.

Alternatif Greedy by Shortest Distance Concerning Highest Diamond Value

Alternatif ini merupakan alternatif *upgrade* dari alternatif sebelumnya dimana selain memperhatikan jarak diamond, bot juga memastikan nilai dari diamond yang ditargetkan. Hal ini membuat bot menjadi lebih cenderung memprioritaskan diamond merah karena memiliki poin yang lebih besar, namun tetap memperhatikan jarak terdekat agar tidak mencari diamond merah sampai terlalu jauh. Mulai dari versi ini pula mulai dipertimbangkan untuk fitur menghindari portal guna memaksimalkan efisiensi.

a. Pemetaan Elemen Greedy

- Himpunan Kandidat: Himpunan diamond yang tersedia pada board, yang meliputi diamond merah yang bernilai 2 poin dan diamond biru yang bernilai 1 poin.

- Himpunan Solusi: Himpunan diamond yang terpilih.
- Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
- Fungsi Seleksi: Memilih diamond dengan jarak terdekat dari bot dengan memprioritaskan diamond dengan value yang lebih besar
- Fungsi kelayakan: Memeriksa jumlah diamond yang telah diperoleh tidak melebihi *max inventory* pada setiap iterasi, dan apabila jumlah diamond telah mencapai *max inventory*, bot akan diarahkan untuk kembali ke base untuk menyimpan diamond.
- Fungsi obyektif: Jumlah poin diamond yang diperoleh maksimum.

b. Analisis Efisiensi Solusi

Kompleksitas kode ini dapat dianalisis sebagai berikut: fungsi findTeleportandRedButton memiliki kompleksitas O(n), di mana n adalah jumlah objek permainan. Fungsi findDistance memiliki kompleksitas O(m), di mana m adalah jumlah pasangan teleport. Fungsi findTarget memanggil findDistance untuk setiap berlian, sehingga memiliki kompleksitas O(d * m), di mana d adalah jumlah berlian. Fungsi findQuadran dan isTeleportInterupt memiliki kompleksitas O(1). Fungsi ignoreTeleport beriterasi melalui semua pasangan teleport sekali, sehingga memiliki kompleksitas O(m). Metode next_move memanggil findTeleportandRedButton, findTarget, dan ignoreTeleport, dengan kompleksitas didominasi oleh findTarget, yaitu O(d * m). Secara keseluruhan, kompleksitas kode ini adalah O(n + d * m), di mana n adalah jumlah objek permainan, d adalah jumlah berlian, dan m adalah jumlah pasangan teleport. Dalam kasus terburuk, jika setiap objek permainan adalah berlian atau teleport, kompleksitasnya bisa dianggap O(n^2).

c. Analisis Efektivitas Solusi

Solusi ini memiliki pendekatan yang lebih baik dalam mencapai fungsi obyektif disebabkan karena dengan menuju ke diamond yang lebih besar terlebih dahulu, akan lebih cepat memenuhi inventory dan bot akan semakin berkesempatan untuk kembali ke base. Alternatif ini juga bersifat efektif karena jika *max inventory* bernilai ganjil, maka bot pasti akan memenuhi kuota dengan diamond genap (diamond merah) terlebih dahulu baru mengarahkan ke diamond ganjil guna memenuhi kuota sehingga kemungkinan melewati diamond merah lebih sedikit.

Namun alternatif ini bersifat kurang efektif untuk berbagai alasan dibawah:

1. Pendekatan yang terbilang umum sehingga seringkali bot yang berdekatan menargetkan diamonds yang sama dan menyebabkan *collision* yang membahayakan bot, terutama jika sedang memiliki diamonds yang banyak.

2. Algoritma memandu bot semakin jauh dari base sehingga waktu sering terbuang sia-sia untuk kembali ke base.
3. Algoritma memandu bot semakin jauh dari base sehingga waktu sering terbuang sia-sia untuk kembali ke base.
4. Posisi diamonds terkadang terkumpul pada suatu titik yang jauh dimana alternatif ini pasti tidak akan menargetkan hal tersebut karena algoritma hanya melihat jarak diamond terdekat.

Alternatif Greedy by Closest to Base

Pendekatan algoritma greedy ini mempertimbangkan efektivitas saat kembali ke base. Dengan memfokuskan prioritas pada diamond dengan jarak yang paling dekat dari base membuat bot dapat mengumpulkan diamond sebanyak mungkin dan dengan jalur yang sebaik mungkin.

a. Pemetaan Elemen Greedy
   - Himpunan Kandidat: Himpunan diamond yang tersedia pada board, yang meliputi diamond merah yang bernilai 2 poin dan diamond biru yang bernilai 1 poin.
   - Himpunan Solusi: Himpunan diamond yang terpilih.
   - Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
   - Fungsi Seleksi: Memilih diamond dengan jarak terdekat dari base sebagai prioritas utama.
   - Fungsi kelayakan: Memeriksa jumlah diamond yang telah diperoleh tidak melebihi *max inventory* pada setiap iterasi, dan apabila jumlah diamond telah mencapai *max inventory*, bot akan diarahkan untuk kembali ke base untuk menyimpan diamond.
   - Fungsi obyektif: Jumlah poin diamond yang diperoleh maksimum.
b. Analisis Efisiensi Solusi

   Fungsi findRedButton mencari posisi tombol merah dengan kompleksitas O(n), di mana n adalah jumlah objek permainan. Fungsi isNearBase memeriksa apakah posisi tertentu dekat dengan basis dengan kompleksitas O(1). Fungsi findDiamondNearBase mencari berlian di sekitar basis dengan kompleksitas O(d), di mana d adalah jumlah berlian. Fungsi findClosestDiamond mencari berlian terdekat dengan kompleksitas O(d). Metode next_move di kelas V4Logic menentukan langkah selanjutnya berdasarkan posisi target yang ditemukan, dengan kompleksitas didominasi oleh findDiamondNearBase dan findClosestDiamond. Secara keseluruhan, kompleksitas kode ini adalah O(n + d), dengan asumsi bahwa jumlah berlian jauh lebih kecil daripada jumlah total objek permainan, sehingga kompleksitas aktual mungkin lebih rendah.

c. Analisis Efektivitas Solusi

Solusi ini efektif terutama jika diamond sangat banyak bertabur disekitar base. Pendekatan ini juga menhindari bot dari bertabrakan dengan bot lain karena bot hanya memprioritaskan jalur sekitar base. Keuntungan lainnya yakni jika base berdekatan dengan base lawan karena adanya kemungkinan diamond yang direset menjadi disekitar base musuh dan base pribadi dan posisi lawan masih jauh dari base sehingga lawan akan mendapat diamond lebih sedikit dan justru meningkatkan resiko di-*tackle* oleh bot.

Namun terdapat beberapa kelemahan dari pendekatan ini yakni:

1. Terlalu bergantung pasda alur game sehingga jika diamond tidak muncul disekitar base membuat bot menerapakan implementasi shortest distance concering value.
2. Jika diamond terlalu sedikit dan bot harus keluar dari zona nyaman, juga akan menerapkan konsep yang sama dari konsep distance concerning value, kecuali jika ada bot yang mereset diamond dengan red button, menjelaskan betapa bergantungkan bot dengan diamond yang ada.
3. Hanya unggul dalam map kecil dimana dengan map kecil dapat dianggap diamond dalam map berupa disekitar base.
4. Kurang cocok jika dihadapkan dengan lawan dan kurang konsisten dalam pertarungan antar bot karena sering kali fitur map tidak mendukung kondisi bot ini.

Alternatif Greedy by Highest Density Value

Alternatif ini merupakah alternatif yang seutuhnya berubah dari alternatif sebelumnya dimana pendekatan ini dicoba jika bot diarahkan murni pada posisi diamond yang berkumpulan. Pendekatan ini dilakukan guna unutk memaksimalkan efisiensi perjalanan pergi bot agar tidak menelusuri jarak yang terlalu jauh karena dipancing oleh diamond dekat namun hanya singular.

a. Pemetaan Elemen Greedy

- Himpunan Kandidat: Himpunan diamond yang tersedia pada board, yang meliputi diamond merah yang bernilai 2 poin dan diamond biru yang bernilai 1 poin.
- Himpunan Solusi: Himpunan diamond yang terpilih.
- Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
- Fungsi Seleksi: Memilih diamond dengan jarak (*threshold)* tertentu dengan diamond lainnya.

- Fungsi kelayakan: Memeriksa jumlah diamond yang telah diperoleh tidak melebihi *max inventory* pada setiap iterasi, dan apabila jumlah diamond telah mencapai *max inventory*, bot akan diarahkan untuk kembali ke base untuk menyimpan diamond.
- Fungsi obyektif: Jumlah poin diamond yang diperoleh maksimum.

b. Analisis Efisiensi Solusi

Kode ini mengimplementasikan logika untuk menentukan langkah selanjutnya dalam permainan dengan mempertimbangkan posisi teleport, tombol merah, dan berlian. Fungsi findTeleportandRedButton mencari pasangan teleport dan posisi tombol merah dengan kompleksitas O(n), di mana n adalah jumlah objek permainan. Fungsi findDistance menghitung jarak antara posisi saat ini, teleport, dan target dengan kompleksitas O(m), di mana m adalah jumlah pasangan teleport. Fungsi findCluster mencari target berlian terbaik berdasarkan jarak dan jumlah berlian di sekitarnya, dengan kompleksitas O(d * m), di mana d adalah jumlah berlian. Metode next_move di kelas V3Logic menentukan langkah selanjutnya berdasarkan posisi target yang ditemukan, dengan kompleksitas didominasi oleh findCluster. Secara keseluruhan, kompleksitas kode ini adalah O(n + d * m).

c. Analisis Efektivitas Solusi

Solusi ini bekerja dengan baik pada map luas karena sangat sedikit langkah yang terbuang sia sia pada saat pencarian diamond. Pendekatan ini juga lebih mungkin menghindari *collision* karena pendekatan yang menargetkan clustering dimana algoritma lain mungkin saja menerapkan pendekatan jarak. Selain itu, konsep clustering bertujuan untuk memenuhi diamond secepat-cepatnya dan kembali ke base dengan cepat sehingga mengurangi kemungkinan diserang oleh bot lawan.

Namun beberapa kelemahan dalam pendekatan ini yakni:

1. Posisi base yang terlalu ujung membuat pendekatan ini menjadi sangat sulit untuk kembali mengingat bahwa fokus utama dalam algoritma ini khusus hanya untuk mencari cluster terbanyak.
2. Karena merupakan hasil upgrade dari bot sebelumnya, terkadang efektivitas yang ada pada cluster diamond biru dibandingkan dengan diamond merah dan sering merujuk pada diamond merah karena dari awal code diamond merah code terbilang lebih efektif atas kemampuan mengisi inventory. Namun terkadang ada test-case yang tidak mendukung efektivitas ini.

3. Sering kali terlalu terfokus pada cluster tanpa mempertimbangkan diamond yang sebenarnya diperlukan sehingga sering melewatkan diamond singular yang mungkin ada didekat bot.
4. Perjalanan pencarian cluster yang terlalu jauh membuat bot menjadi sasaran yang empuk untuk bot lainnya.

Alternatif Greedy by Clustering Efficiency

Alternatif ini mempertimbangkan berbagai faktor yang mungkin dapat menganggu konsistensi bot. Dalam beberapa kondisi, terutama posisi diamond dan portal, bot bergerak secara tidak efektif. Pendekatan ini mempertimbangkan berbagai faktor dalam map dan diukur dalam bentuk efisiensi dan memberikan target optimal bagi bot.

$$Efficiency = \frac{total\ diamonds\ value\ in\ cluster}{distance\ to\ cluster + \ internal\ distance + distance\ to\ base}$$

Untuk mencegah bot terpaku pada cluster, juga dilakukan perbandingan efisiensi untuk diamond singular sehingga bot akan dapat membuat keputusan yang tepat dalam setiap penentuan targetnya. Cara kerja utama dari clustering yakni berupa mengelompokkan diamond sesuai threshold yang dianggap, kemudian dicek efisiensi yang ada dan dikembalikan efisiensi terbesar.

a. Pemetaan Elemen Greedy

- Himpunan Kandidat: Himpunan diamond yang tersedia pada board, yang meliputi diamond merah yang bernilai 2 poin dan diamond biru yang bernilai 1 poin.
- Himpunan Solusi: Himpunan diamond yang terpilih.
- Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
- Fungsi Seleksi: Memilih diamond dengan efisiensi terbesar.
- Fungsi kelayakan: Memeriksa jumlah diamond yang telah diperoleh tidak melebihi *max inventory* pada setiap iterasi, dan apabila jumlah diamond telah mencapai *max inventory*, bot akan diarahkan untuk kembali ke base untuk menyimpan diamond.
- Fungsi obyektif: Jumlah poin diamond yang diperoleh maksimum.

b. Analisis Efisiensi Solusi

Fungsi findTeleportandRedButton mencari pasangan teleport dan posisi tombol merah dengan kompleksitas O(n), di mana n adalah jumlah objek permainan. Fungsi findDistance menghitung jarak antara posisi saat ini dan target dengan kompleksitas O(m), di mana m adalah jumlah pasangan teleport. Fungsi findCluster mencari kluster

berlian terbaik dengan kompleksitas O(d^2), di mana d adalah jumlah berlian. Fungsi findTarget menentukan target berikutnya berdasarkan kondisi permainan dengan kompleksitas O(d * m). Fungsi ignoreTeleport memodifikasi target untuk menghindari teleport yang tidak diinginkan dengan kompleksitas O(m). Metode next_move di kelas AkiongLogic menentukan langkah selanjutnya dengan mempertimbangkan sisa waktu, skor, dan kondisi permainan lainnya. Secara keseluruhan, kompleksitas kode ini adalah O(n + d^2 + d * m), yang didominasi oleh fungsi findCluster karena pencarian kluster berlian.

c. Analisis Efektivitas Solusi

Solusi ini terbilang efektif karena telah mempertimbangkan banyak parameter. Algoritma ini akan dapat bekerja dengan baik jika jumlah diamonds dalam map selalu banyak sehingga filter clustering dapat dilakukan dan menyebabkannya lebih unggul. Solusi ini juga terbilang tidak begitu umum karena adanya pendapat dari subjektif dari kami yang menganggap pendekatan umum lainnya bersifat baik hanya dalam beberapa skenarion. Namun, terdapat beberapa kelemahan dari algoritma ini:

1. Terlalu kompleks penyusunan code sehingga untuk efisiensi sedikit lebih kurang dari beberapa pendekatan greedy.
2. Terkadang adanya bentrokan antara kedua sektor dengan value serupa mengakibatkan adanya move yang terbuang.
3. Karena merupakan hasil *upgrade* dari bot sebelumnya, terkadang efektivitas yang ada pada cluster diamond buru dibandingkan dengan diamond merah dan sering merujuk pada diamond merah karena dari awal code diamond merah code terbilang lebih efektif atas kemampuan mengisi *inventory.* Namun terkadang ada test-case yang tidak mendukung efektivitas ini.
4. Jarak base yang berdekatan membuat clustering tidak efektif karena teknik clustering yang mempertimbangkan jarak ke base kami nantinya akan bentrok dengan pandangan jarak terdekat dari bot lain dan sering kali mengundang *collision.*

Strategi greedy yang diimplementasikan

Sesuai dengan urutan penempatan penjelasan untuk tiap alternatif serta penjelasan diawal dimana dijelaskan bahwa setiap alternatif diatas merupakan versi bot greedy sehingga tentu saja konsep yang akan kami gunakan yaitu *Greedy by Clustering Efficiency*. Kami memilih bot ini karena kami merasa bahwa konsep clustering sudah cukup bagus, namun akan lebih baik

lagi jika juga mempertimbangkan efisiensi, baik dari clustering maupun juga memperhatikan apakah perlu dilakukan clustering.

Secara umum, kami tidak mengatakan bahwa pendekatan pendekatan versi sebelumnya buruk. Hal ini disebabkan karena fitur dalam permainan dapat terus diganti sehingga masing masing versi dapat memberikan kemampuan terbaik pada pendekatannya. Namun, kami memilih bot berdasarkan konsistensi yang ada.

Beberapa pendapat kami untuk tiap bot:

a. Greedy by Shortest Distance terlalu umum dan kurang cocok dalam sisi kompetitif, namun dari sisi pribadi juga terlalu mengandalkan portal, dan sangat tidak efisien dalam perjalanan.
b. Greedy by Shortest Distance Concerning Value cocok dalam kompetitif jika value red lebih dekat, namun dari sisi solo terkadang sangat terpaku pada red diamond dan menolak blue diamond.
c. Greedy by Shortest to Base sangat mengandalkan kondisi map sehingga sangat *coinflip* dalam implementasinya. Namun jika optimal dapat memberikan hasil yang bagus.
d. Greedy by Highest Density Value cukup bagus namun memiliki keadaan ketika dibawa terlalu oleh cluster diamond yang tidak memenuhi inventory.
e. Greedy by Clustering Efficiency cukup konsisten dalam peraihan diamond baik secara kompetitif maupun solo, hanya saja komplexitas yang ada cukup tinggi.

Berikut merupakan hasil uji coba pencarian diamond secara solo dalam 15 pengulangan dan dihitung rata rata jumlah diamond yang ada dalam map berukuran 14x14 dengan 2 portal dan waktu 60 detik.

Tabel 3. 1 Pengukuran Diamond Rata Rata Secara Solo

Shortest Distance	Shortest Distance Concerning Value	Closest to Base	Clustering / Highest Density Value	Clustering Efficiency
8	10,25	10,75	12	13,5

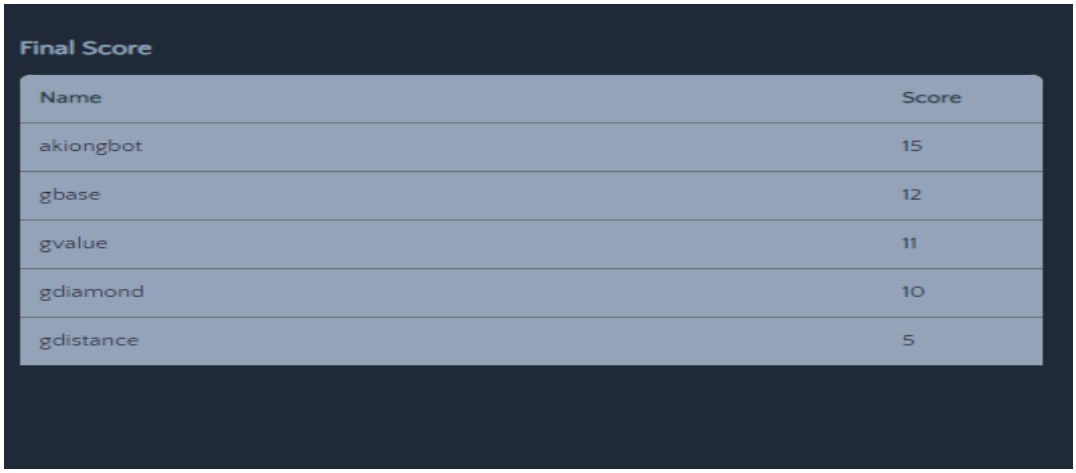
Selain itu juga dilakukan beberapa uji cobna kelima bot secara bersamaan. Dalam hal ini, jumlah diamond yang diraih oleh algoritma *Greedy by Clustering Efficiency* mungkin tidak begitu terlihat. Namun menurut kami hal ini lazim dan tidak bisa dijadikan parameter kemampuan karena fokus utama tugas besar kali ini yakni berupa penanganan algoritma greedy. Greedy dalam hal permainan ini secara umum tentu untuk mengambil diamond sebanyak banyaknya.

Jika dihadapkan dalam permainan secara *multiplayer*, faktor tambahan (seperti yang dijelaskan pada bagian "Variabel Tambahan") terlalu mempengaruhi hasil uji coba sehingga tidak cocok dijadikan syarat pemilihan jika standar kita berupa kemenangan telak.

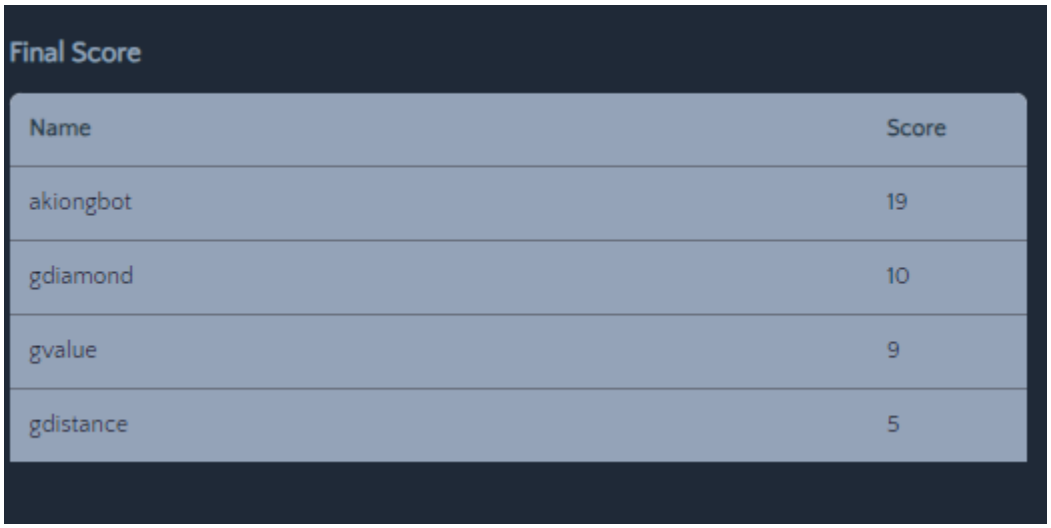
Atas hal tersebut, kami melakukan uji coba selama beberapa kali untuk mendapat konsistensi dari program kami. Dari hasil uji coba kami selama 20x, tercatat bahwa algoritma kami menang sebanyak 9 kali, dan untuk 11 kekalahan lainnya juga bukan merupakan juara terakhir.

Berikut beberapa hasil uji coba:

Gambar 3. 1 Final Score Multiplayer 1



Gambar 3. 2 Final Score Multiplayer 2



Gambar 3. 3 Final Score Multiplayer 3

Dari sini terlihat bahwa dalam *multiplayer*, bot akiongbot (Cluster by Efficiency) tidak selalu menang, namun konsisten dalam mendapat peringkat menengah dan mencegah dirinya dari berada di paling bawah. Digabungkan dengan kemampuan dalam mengambil diamond jika diadu sendiri, membuat *Clustering by Efficiency* menjadi pilihan kami.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

Class dan Struktur Data dalam Program

Beberapa file penting dalam penjalanan bot:

A. File main.py dalam folder game/ berisi algoritma utama yang mengintegrasikan semua komponen dalam bot starter pack ini. Kita dapat menambahkan logika yang telah kita implementasikan dalam folder game/logic/ dengan mengimpor pilihan tersebut dan kemudian menempatkannya dalam variabel CONTROLLERS yang ada dalam file ini.
B. Dalam file utils.py yang berada di folder game/, terdapat beberapa fungsi template sebagai berikut:
    - Fungsi clamp digunakan untuk memastikan nilai n berada dalam rentang nilai minimum dan maksimum yang ditentukan. Jika n lebih kecil dari batas bawah, maka nilai yang diambil adalah batas bawah, dan jika n lebih besar dari batas atas, maka nilai yang diambil adalah batas atas.
    - Fungsi get_direction digunakan untuk menghitung perubahan koordinat (delta_x dan delta_y) yang diperlukan untuk menggerakkan bot menuju suatu posisi tujuan (destination).
    - Fungsi position_equals digunakan untuk memeriksa apakah dua posisi yang diberikan sebagai parameter memiliki koordinat yang sama atau tidak.

Terdapat beberapa struktur data dari class yang dipakai dalam pembuatan program menggerakkan bot dengan algoritma greedy. Penjelasan mengenai source code dapat dilihat pada pranala youtube dibawah.

Class Bot
(Kelas dari Bot yang berisikan identitas bot)

`Type Bot :`
`  < name: String`
`  email: String`
`  id: String >`

Class Position
(Kelas atribut yang menyatakan posisi / koordinat suatu objek)
`Type Position :`

```
   < x: int
     y: int >
```

Class Properties

(Kelas atribut yang menyimpan data-data seperti nama, skor, inventory, poin, diamond sekarang, dan lainnya dari Game Object)

```
Type Properties :
   < points: int = None #atribut diamond, pembeda biru dan merah
     pair_id: str = None #atribut pair, mengetahui pasangan teleport
     diamonds: int = None #atribut bot, jumlah diamond di saku
     score: int = None #atribut bot, total diamond dikumpulkan
     name: str = None
     inventory_size: int = None #atribut bot, kapasitas saku
     can_tackle: bool = None  #atribut bot bisa saling memakan/tidak
     milliseconds_left: int = None #atribut bot, sisa waktu
     time_joined: str = None #atribut bot, waktu bergabung
     base: Position = None > #atribut bot, letak posisi pulang
```

Class GameObject

(Kelas atribut yang merupakan identitas dan juga menyimpan beberapa data seperti posisi, id, type, dan properties)

```
Type GameObject :
    < id: int
      position: Position
      type: str
      properties: Properties = None >
```

Class Config

(Kelas atribut yang memiliki beberapa pengaturan yang mengatur jalannya permainan seperti bisa saling memakan, waktu, rasio diamond, inventory, dan lainnya)

```
Type Config :
   < generation_ratio: float = None
     min_ratio_for_generation: float = None
     red_ratio: float = None
     seconds: int = None
     pairs: int = None
     inventory_size: int = None
     can_tackle: bool = None >
```

Class Feature

(Kelas atribut yang berisikan fitur dan berisi konfigurasi pengaturan)

```
Type Feature :
   < name: str
     config: Config = None >
```

Class Board

(Kelas atribut yang memuat segala jenis data permainan seperti Fitur, Game Object, dan daftar player, serta diamond)

```
Type Board :
    < id: int
      width: int #lebar papan
      height: int #tinggi papan
      features: List[Feature]
      minimum_delay_between_moves: int
      game_objects: Optional[List[GameObject]] #properti objek game
      bots: List[GameObject] #daftar pemain/bot
      diamonds: List[GameObject] > #daftar diamond
```

Implementasi Program dalam Pseudocode

Dalam pengimplementasian dari algoritma greedy, program dibagi menjadi beberapa function yang digunakan untuk menggerakkan bot, serta mencari jalur tersingkat dan menerapkan algoritma greedy dalam mencari diamond

Fungsi findTeleportandRedButton

(Fungsi untuk mencari posisi *teleporter* dua arah {bisa lebih dari satu} dan juga posisi *redbutton*)

```
function findTeleportandRedbutton (game_objects: List[GameObject]) ->[Position],
Position
       teleport_pair <- null
       redbutton <- null
       for game_object in game_objects then
       #iterasi pencarian teleport dan redbutton
           if game_object.type = 'TeleportGameObject' then
                 pair_id <- game_object.properties.pair_id
                 position <- game_object.position
                 if pair_id in teleport_pair then
```

```
                    teleport_pair[pair_id].append(position)
               else
                    teleport_pair[pair_id <- [position]
          elif game_object.type = 'DiamondButtonGameObject' then
               redbutton <- game_object.position
     -> [teleport_pair],redbutton
     #return posisi beberapa pasangan teleport dan posisi red button
```

Fungsi findDistance

(Fungsi untuk mencari posisi jarak dari suatu posisi menuju posisi yang ingin dituju dengan melewati teleport)

```
function findDistance (current: Position, teleport: List[Tuple[Position, Position]], target: Position) -> Position, Integer, Boolean
      #inisialiasi variabel sebagai pembanding
      tp_close_to_current <- [null for i in range(length(teleport))]
      distance_to_tp <- [100 for i in range(length(teleport))]
      otherside_tp <- [null for i in range(length(teleport))]
      otherside_tp_to_target <- [100 for i in range(length(teleport))]
      for i in range(length(teleport)) do
      #iterasi teleport sebagai pembanding jarak
          for j in range(2) do
               total <- |teleport[i][j].x - current.x| + |teleport[i][j].y - current.y|
               if total < distance_to_tp[i] and total != 0 then
                    distance_to_tp[i] <- total tp_close_to_current[i] <- teleport[i][j]
                  for k in range(2) do
                        if k != j then
                              otherside_tp[i] <- teleport[i][k]
                              other <- |teleport[i][k].x - target.x| + |teleport[i][k].y - target.y|
                              otherside_tp_to_target[i] <- other
       tp_most_efficient <- 100
       tp_entry <- null
       for i in range(length(teleport)) do
       #iterasi mencari teleport dengan jarak terefisien
           if distance_to_tp[i] + otherside_tp_to_target[i] < tp_most_efficient then
                tp_most_efficient <- distance_to_tp[i] + otherside_tp_to_target[i]
                tp_entry <- tp_close_to_current[i]
       current_to_target <- |current.x - target.x| + |current.y - target.y|
       if tp_most_efficient > current_to_target then
       #kondisi jarak terefisien tanpa teleport
           distance <- current_to_target
           -> target, distance, false
       else
       #kondisi jarak terefisien dengan teleport
           distance <- tp_most_efficient
           -> tp_entry, distance, true
```

Fungsi findCluster

(Fungsi untuk mencari kumpulan diamond dengan memasukkan indikator petak diamond untuk mencari tahu kumpulan diamond)

```
function findCluster (current: GameObject, diamonds: List[GameObject],
cluster_distance_threshold: Integer) -> GameObject, Float, Integer
      clusters <- []
      #Untuk setiap diamond dilakukan pencarian cluster
      for diamond in diamonds do
           added_to_cluster <- false
           for cluster in clusters do
               #Mengecek untuk diamond lainnya selain diri sendiri apakah
jaraknya lebih kecil dari threshold
               if any(findDistance(diamond.position, [],
other_diamond.position)[1] <= cluster_distance_threshold for other_diamond in
cluster) then
                    cluster.append(diamond)
                    added_to_cluster <- true
                    break
       if not added_to_cluster then
           clusters.append([diamond])
       #Menghitung efisiensi yang ada berdasarkan jarak ke cluster, jarak dalam
cluster dan jarak ke base
       function cluster_efficiency(cluster: List[GameObject]) -> Float
           distance_to_cluster <- min(findDistance(current.position, [],
d.position)[1] for d in cluster)
           internal_cluster_distance <- sum(findDistance(d1.position, [],
d2.position)[1] for d1 in cluster for d2 in cluster if d1 != d2)
           distance_to_base <- min(findDistance(d.position, [],
current.properties.base)[1] for d in cluster)
           total_value <- sum(d.properties.points for d in cluster)
           -> total_value / (distance_to_cluster + internal_cluster_distance +
distance_to_base)
        #Memilih cluster sesuai dengan efisiensi tertinggi
        best_cluster <- max(clusters, key=cluster_efficiency, default=[])
           if best_cluster then
               target_diamond <- min(best_cluster, key=lambda d:
findDistance(current.position, [], d.position)[1])
               efficiency <- cluster_efficiency(best_cluster)
               total_score <- sum(d.properties.points for d in best_cluster)
               -> target_diamond, efficiency, total_score
           else
               -> null, 0, 0
```

Fungsi findTarget

(Fungsi mengembalikan posisi diamond sesuai dengane efisiensi yang dihitung)

```
function findTarget(current: GameObject, teleport: List[Tuple[Position,
Position]], diamonds: List[GameObject], redButton: Position, board: Board) ->
Position, Boolean
      remaining_diamonds <- length(diamonds)
      remaining_inventory <- current.properties.inventory_size -
current.properties.diamonds
     #mempertimbangkan jarak redbutton jika diamond tinggal sedikit
      if remaining_diamonds <= 3 then
          red_button_target, distance_to_red_button, is_tp_red_button <-
findDistance(current.position, teleport, redButton)
          if distance_to_red_button < remaining_diamonds then
               -> red_button_target, is_tp_red_button
      cluster_target, cluster_efficiency, expected <- findCluster(current,
diamonds, 3)
      #inisialisasi diamond terdekat
      distance_diamond_blue <- 100
      distance_diamond_red <- 100
      target_blue <- null
      target_red <- null
      is_tp_red <- false
      is_tp_blue <- false
      for diamond in diamonds do #iterasi mencari diamond terdekat merah/biru
           temp_target, temp_distance, temp_tp <- findDistance(current.position,
teleport, diamond.position)
           if diamond.properties.points == 1 then
                if temp_distance < distance_diamond_blue then
                     distance_diamond_blue <- temp_distance
                     target_blue <- temp_target
                     is_tp_blue <- temp_tp
           else
                if temp_distance < distance_diamond_red then
                     distance_diamond_red <- temp_distance
                     target_red <- temp_target
                     is_tp_red <- temp_tp
      #memperhitungkan jarak menuju base
       base_target, _, is_back_tp <- findDistance(current.position, teleport,
current.properties.base)
       if target_red then
            base_target, distance_base_red, _ <- findDistance(target_red,
teleport, current.properties.base)
```

```
            value <- 1
            calculated_distance <- distance_base_red
      if target_blue then
            base_target, distance_base_blue, _ <- findDistance(target_blue,
teleport, current.properties.base)
            value <- 2
            calculated_distance <- distance_base_blue
      #mempertimbangkan jarak red button
      red_button_target, distance_to_red_button, is_tp_red_button <-
findDistance(current.position, teleport, redButton)
      should_press_red_button <- false
      if min(distance_diamond_blue, distance_diamond_red) >=
distance_to_red_button + 3 then
            should_press_red_button <- true
      if should_press_red_button then
            -> red_button_target, is_tp_red_button
      remaining_inventory <- current.properties.inventory_size -
current.properties.diamonds
      if remaining_inventory == 0 and target_red then
            -> target_red, is_tp_red
      if cluster_target and cluster_efficiency > value / calculated_distance and
expected <= remaining_inventory then
            _, calculated_distance, temp <- findDistance(current.position,
teleport, cluster_target.position)
            -> cluster_target.position, temp
     #kondisi yang dipilih berdasarkan data pertimbangan sebelumnya
      if distance_diamond_blue < distance_diamond_red - 2 then
            if distance_base_blue + distance_diamond_blue >
round(current.properties.milliseconds_left / 1000) then
                  -> base_target, is_back_tp else -> target_blue, is_tp_blue
      else
            if distance_base_red + distance_diamond_red >
round(current.properties.milliseconds_left / 1000) then
                  -> base_target, is_back_tp
            else
                  if current.properties.diamonds <
current.properties.inventory_size - 1 then
                        -> target_red, is_tp_red
                  else
                        if distance_base_blue + distance_diamond_blue >
round(current.properties.milliseconds_left / 1000) then
                              -> base_target, is_back_tp
                        else
                              -> target_blue, is_tp_blue
```

Fungsi findQuadran

(Fungsi untuk mencari target berada di suatu kuadran dari titik asal, semisal berada di pojok kanan atas maka kuadran 1, pojok kiri bawah kuadran 3, 10 yang menyatakan berada di x yang sama dan –10 yang menyatakan berada di posisi y yang sama, dan seterusnya yang akan digunakan untuk menghindari teleport)

```
function findQuadran(current: Position, target: Position) -> Integer
     # 10 berarti vertikal, -10 berarti horizontal
     direction_quadran <- 10
     if current.x > target.x then
          if current.y > target.y then
               direction_quadran <- 2
          elif current.y < target.y then
               direction_quadran <- 3
          else
               direction_quadran <- -10
     elif current.x < target.x then
          if current.y > target.y then
               direction_quadran <- 1
          elif current.y < target.y then
               direction_quadran <- 4
          else direction_quadran <- -10
     -> direction_quadran #mengembalikan ada di kuadran berapa
```

Fungsi isTeleportInterupt

(Fungsi untuk mencari apakah dalam perjalanan menuju suatu target, ada teleport yang menghalangi dan akan mengembalikan true bila ada yang menghalangi)

```
function isTeleportInterupt(current: Position, teleport: Position, target:
Position) -> Boolean, Integer
      quadran <- findQuadran(current, target)

      #beberapa kasus untuk mencari apakah teleport menghalangi
      if quadran = 10 then
          if current.x = teleport.x and current.x = target.x then
                 if current.y > target.y then
                       if teleport.y < current.y and teleport.y > target.y then
                             -> true, quadran
                       else
                             -> false, quadran
                 else
                       if teleport.y > current.y and teleport.y < target.y then
                             -> true, quadran
```

```
                        else
                              -> false, quadran
            else
                  -> false, quadran
       elif quadran = -10 then
            if current.y = teleport.y and current.y = target.y then
                  if current.x > target.x then
                        if teleport.x < current.x and teleport.x > target.x then
                              -> true, quadran
                        else
                              -> false, quadran
                  else
                        if teleport.x > current.x and teleport.x < target.x then
                               -> true, quadran
                        else
                             -> false, quadran
             else
                  -> false, quadran
       else
             if current.y = teleport.y then
                  if quadran = 1 or quadran = 4 then
                        if teleport.x > current.x and teleport.x < target.x then
                               -> true, quadran
                        else
                               -> false, quadran
                  elif quadran = 2 or quadran = 3 then
                        if teleport.x < current.x and teleport.x > target.x then
                               -> true, quadran
                        else
                               -> false, quadran
             else
                  if teleport.x = target.x then
                        if quadran = 1 or quadran = 2 then
                              if teleport.y < current.y and teleport.y > target.y
then
                                   -> true, quadran
                              else
                                   -> false, quadran
                        elif quadran = 3 or quadran = 4 then
                              if teleport.y > current.y and teleport.y < target.y
then
                                   -> true, quadran
                              else
                                  -> false, quadran
                   else
```

```
                    -> false, quadran
```

Fungsi ignoreTeleport

(Fungsi untuk menghindari teleport jika tidak ingin mengincar teleport dengan bergerak sesuai kemauan kita)

```
function ignoreTeleport(current: Position, teleport: List[Tuple[Position,
Position]], target: Position, board: Board) -> Position
     result <- target
     is_teleport_interupt, quadran <- false, null
     teleport_crash <- null
     for i in range(length(teleport)) do
         for j in range(2) do
              temp_will, temp_quadran <- isTeleportInterupt(current,
teleport[i][j], target)
         if temp_will then
              teleport_crash <- teleport[i][j]
              is_teleport_interupt <- temp_will
              quadran <- temp_quadran
              break
     #beberapa kasus untuk menghindar berdasarkan kuadran dan letak teleport
     if is_teleport_interupt then
         if quadran = 10 then
             if current.x = 0 then
                  result.x <- current.x + 1
                  result.y <- current.y
             elif current.x = board.width - 1 then
                  result.x <- current.x - 1
                  result.y <- current.y
             else
                  result.x <- random choice between (current.x + 1) and
(current.x - 1)
                  result.y <- current.y
         elif quadran = -10 then
             if current.y = 0 then
                  result.x <- current.x
                  result.y <- current.y + 1
             elif current.y = board.height - 1 then
                  result.x <- current.x
                  result.y <- current.y - 1
             else
                  result.x <- current.x
                  result.y <- random choice between (current.y + 1) and
(current.y - 1)
        elif quadran = 1 then
             if current.x = teleport_crash.x - 1 then
```

```
                    result.x <- current.x
                    result.y <- current.y - 1
               elif current.x = teleport_crash.x then
                    result.x <- current.x
                    result.y <- current.y - 1
          elif quadran = 2 then
               if current.x = teleport_crash.x + 1 then
                    result.x <- current.x
                    result.y <- current.y - 1
               elif current.x = teleport_crash.x then
                    result.x <- current.x
                    result.y <- current.y - 1
          elif quadran = 3 then
               if current.x = teleport_crash.x + 1 then
                    result.x <- current.x
                    result.y <- current.y + 1
               elif current.x = teleport_crash.x then
                    result.x <- current.x
                    result.y <- current.y + 1
          elif quadran = 4 then
               if current.x = teleport_crash.x - 1 then
                    result.x <- current.x
                    result.y <- current.y + 1
               elif current.x = teleport_crash.x then
                    result.x <- current.x
                    result.y <- current.y + 1
          -> result
```

Fungsi next_move

(Fungsi untuk mengatur gerakan bot dengan memanfaatkan semua fungsi yang telah disiapkan di atas untuk mencari diamond terbanyak dengan algoritma greedy)

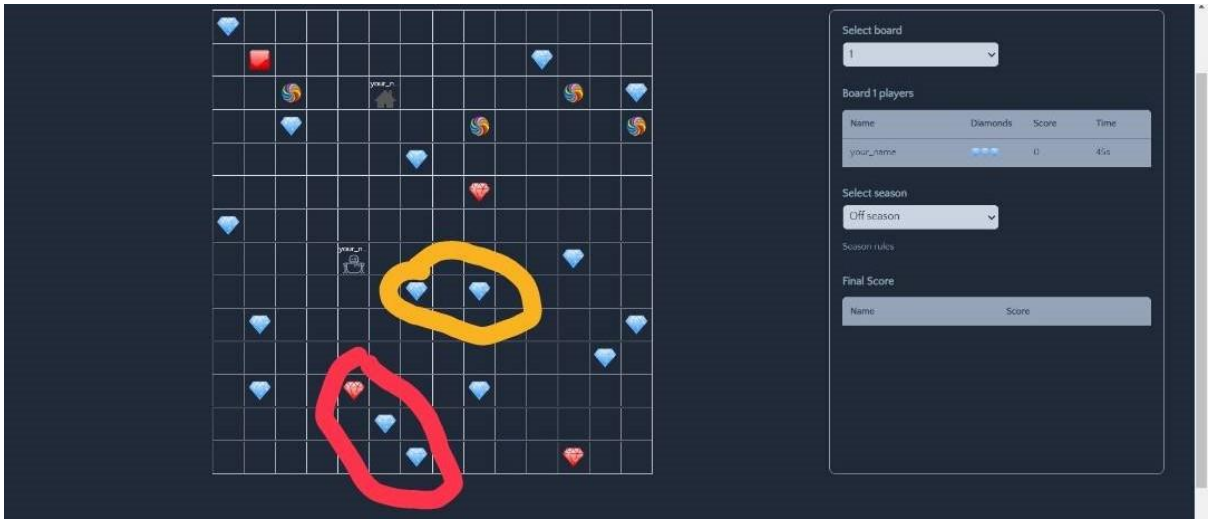
```
function next_move(self, board_bot: GameObject, board: Board) -> Integer, Integer
      teleport, redbutton <- findTeleportandRedButton(board.game_objects)
      target, is_teleport <- findTarget(board_bot, teleport, board.diamonds,
redbutton, board) #pencarian target

      if not is_teleport then
      #kondisi jika tidak masuk teleport, maka akan menghindar jika menghalangi
            target <- ignoreTeleport(board_bot.position, teleport, target,board)

      if target = board_bot.position then #indikasi jika terdapat gerakan ilegal
            target <- board_bot.properties.base

      props <- board_bot.properties
      if props.diamonds = props.inventory_size then
```

```
      #pulang jiika diamond di saku penuh
            self.goal_position <- board_bot.properties.base
      else
            self.goal_position <- null

      current_position <- board_bot.position
      if self.goal_position then #kondisi tujuan jika diamond penuh
            delta_x, delta_y <- get_direction(
                  current_position.x,
                  current_position.y,
                  self.goal_position.x,
                  self.goal_position.y
             )
      else #kondisi tujuan normal
            delta_x, delta_y <- get_direction(
                  current_position.x,
                  current_position.y,
                  target.x,
                  Target.y
             )

      if delta_x = 0 and delta_y = 0 then
      #antisipasi gerakan ilegal dengan gerakan random
            delta_x, delta_y <- random choice from[(1, 0),(-1, 0),(0, 1),(0, -1)]
      -> delta_x, delta_y
```

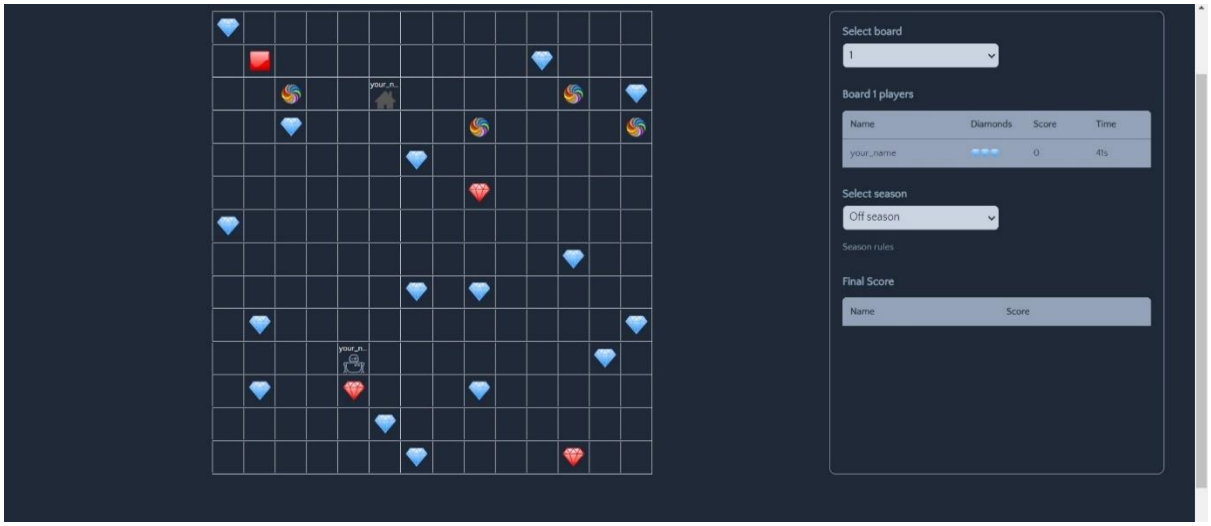
Analisis Algoritma Greedy

Pengujian ketepatan algoritma kami dilakukan dengan menguji kemampuan bot dalam melaksanakan tugasnya serta memastikan bot efektif dalam mengambil diamond dalam sebuah board dengan berbagai variasi kondisi (test case). Dalam hal mengecek apakah algoritma sesuai, tentu pertama yang diperiksa yaitu apakah clustering sesuai dan efisien, kemudian mengecek apakah bot dapat kembali ke base sesuai dengan ketentuan, serta apakah portal bekerja sebagaimana mestinya.

Tes Kemampuan Pencarian Clustering



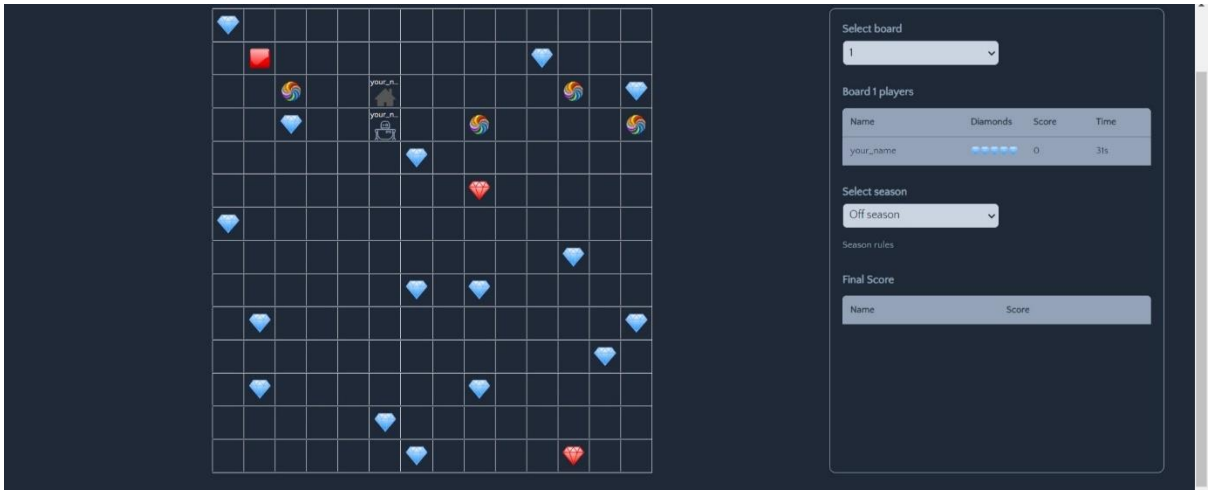
Gambar 4. 1 Tes Kemampuan Clustering (Before)



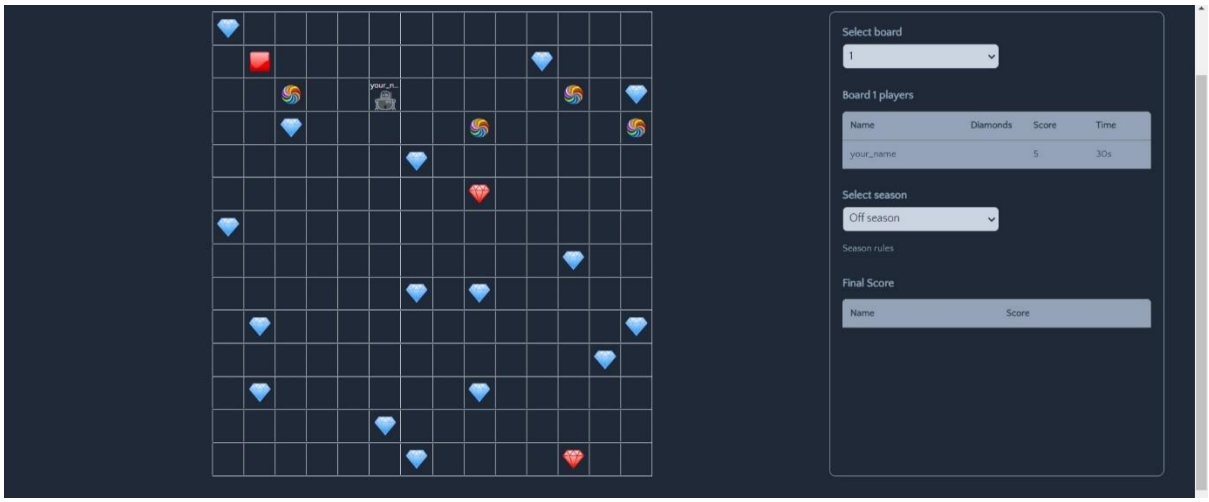
Gambar 4. 2 Tes Kemampuan Clustering (After)

Sesuai Gambar diatas, terbukti bahwa meskipun cluster kuning lebih dekat dari bot juga base, namun cluster merah yang sedikit lebih jauh namun memilih keunggulan value sebanyak 2 justru menjadi prioritas clustering. Hal ini membuktikan bahwa bot tidak hanya memeriksa cluster terdekat, namun juga memeprtimbangkan jumlahnya agak memenuhi inventory secepat mungkin.

Tes Kemampuan Kembali ke Base



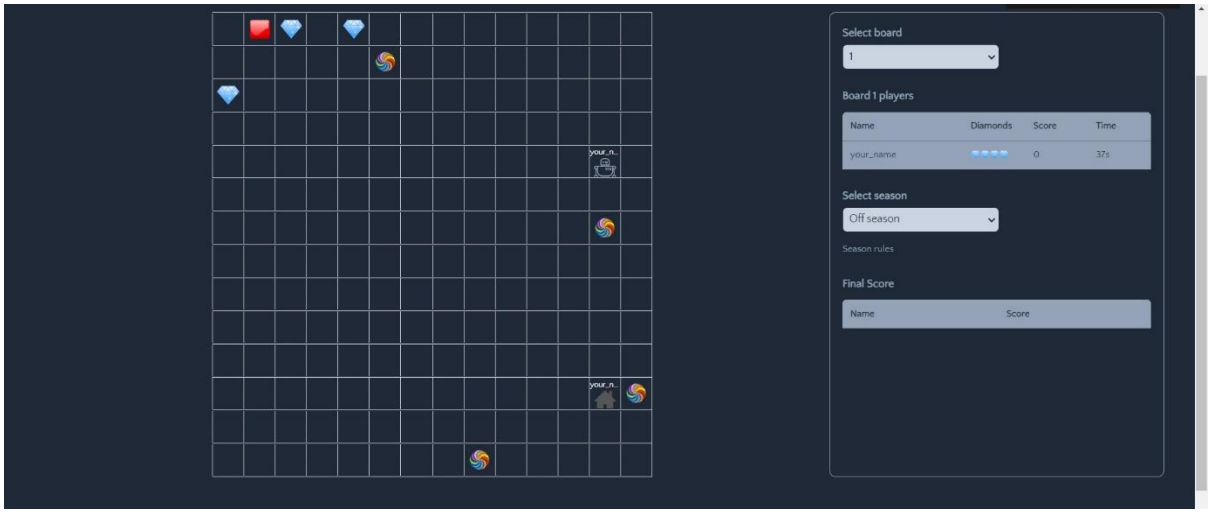
Gambar 4. 3 Tes Kemampuan Return to Base (Before)



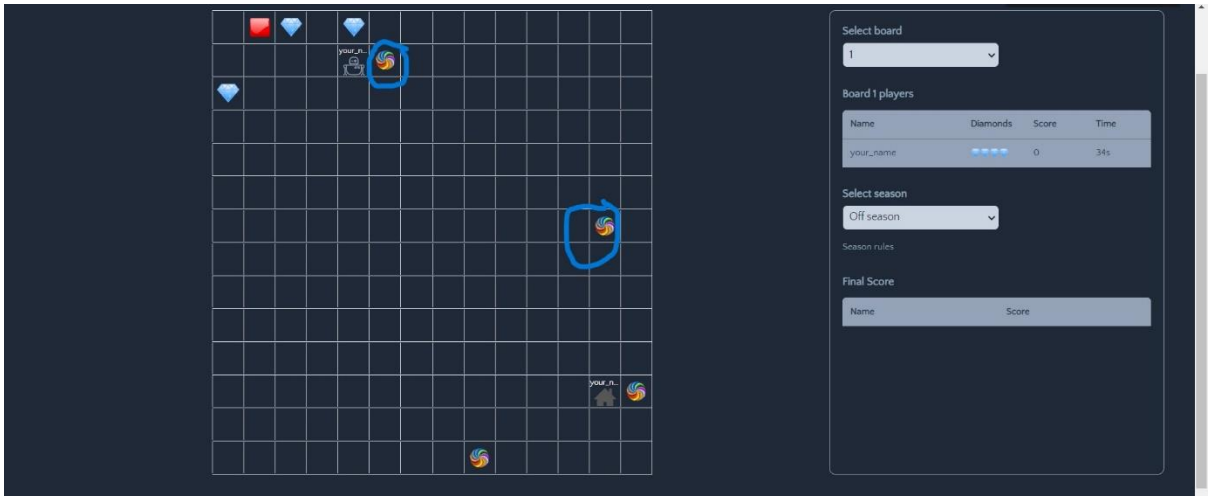
Gambar 4. 4 Tes Kemampuan Return to Base (After)

Sesuai gambar, Ketika diamond berjumlah 5, bot segera mengarahkan dirinya ke base. Ini membuktikkan kebenaran dari konsep pengembalian bot ke base. Pendekatan ini memiliki kelemahan dimana terkadang posisi base linear dengan kedua portal dan mengakibatkan looping tiada henti dari pergerakan bot. Namun kami merasa hal tersebut sudah diluar penanganan kami karena kondisi tersebut hanya muncul beberapa kali saja.

Tes Kemampuan Portal



Gambar 4. 5 Tes Kemampuan Portal (Before)



Gambar 4. 6 Tes Kemampuan Portal (After)

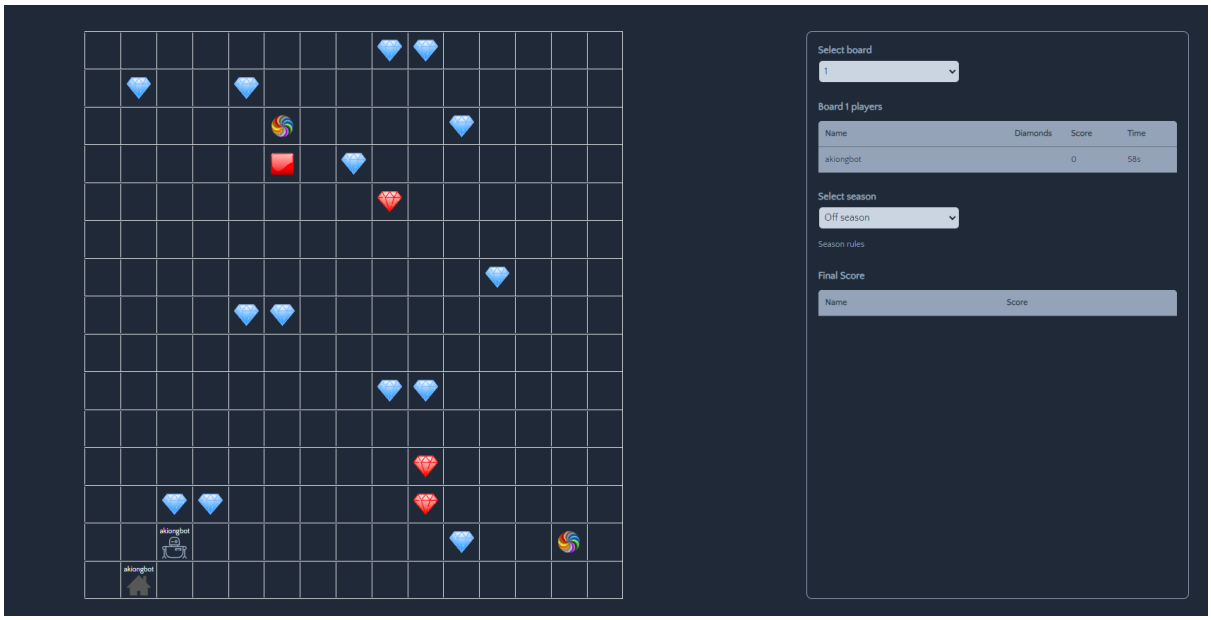
Gambar diatas membuktikan bahwa bot lebih memilih untuk melewati portal daripada berjalan ke arah diamond, menandakan algoritma portal telah berfungsi dengan baik. Dengan melewati portal, dalam percobaan diatas, sudah menghemat sangat banyak langkah, belum lagi memperhatikan bahwa setelah selesai, akan kembali ke base pula melewati portal tersebut sehingga efektivitas dialami 2 kali.

Berikut Beberapa test case konsistensi dari pendekatan *Greedy by Clustering Efficiency:*

Test Case Posisi Base diujung map dan hanya ada 1 portal

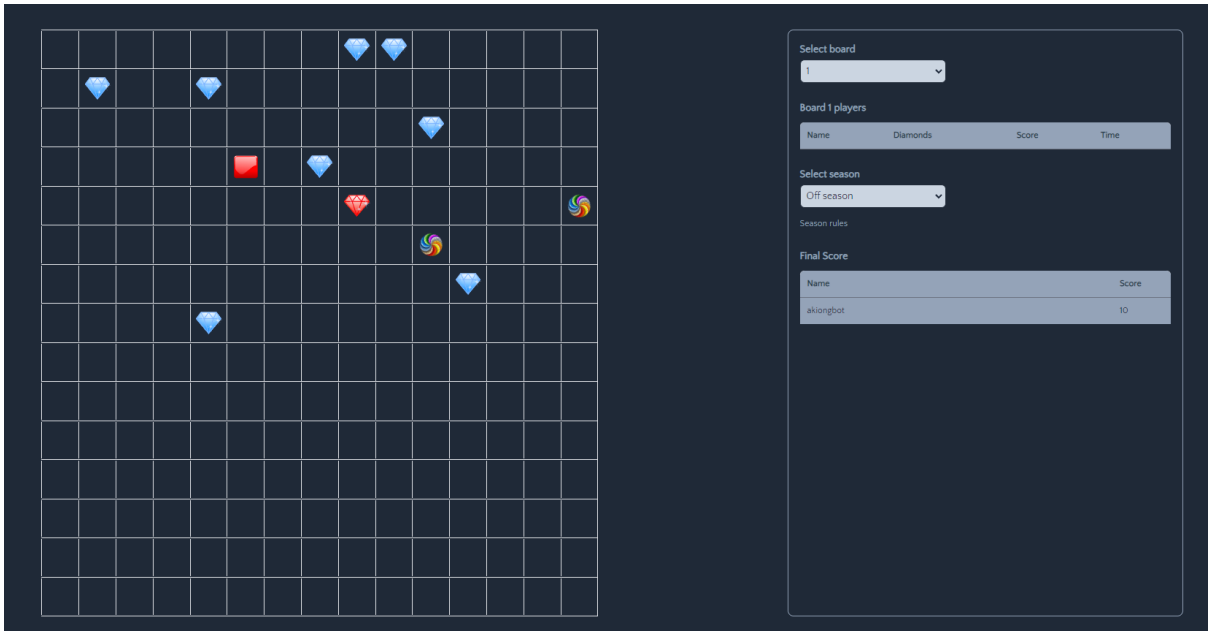
Pada test case ini, dicoba jika akses ke tiap diamond susah karena hanya ada 1 portal dan dibandingkan dengan posisi bas ebot di paling ujung sehingga membuat jarak menjadi sangat tidak efisien.

Foto Awal:



Gambar 4. 7 Test Case 1 Portal Base diujung (Before)

Foto akhir (Skor):



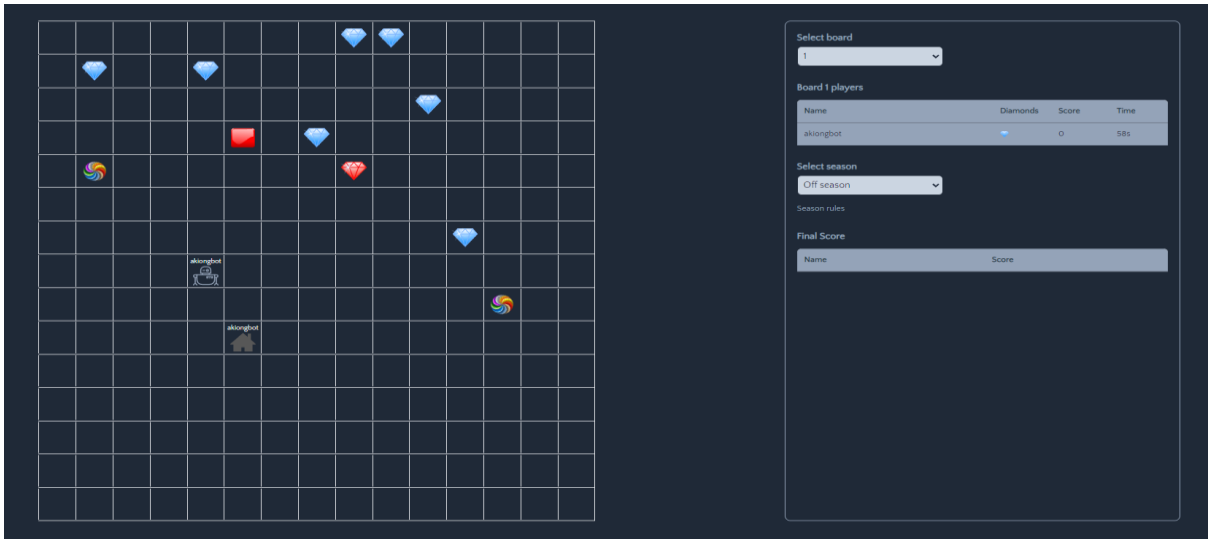
Gambar 4. 8 Test Case 1 Portal Base diujung (After)

Seperti yang terlihat, didapat skor sebanyak 10 untuk kasus ini. Kami menganggap test case in sebagai testcase paling tidak efisien. Namun, sebuah analisa yang kami dapatkan yakni berupa konsistensi yang ada dimana untuk testcase seburuk apapun, kami masih dapat memperoleh nilai 10. Pendekatan ini mungkin bukan merupakan alternatif paling bagus dalam kondisi ini dikarenakan sebuah cluster pasti akan berjarak jauh dari base sehingga kurang efektif.

Test Case Posisi Base diujung ditengah map dan hanya ada 1 portal

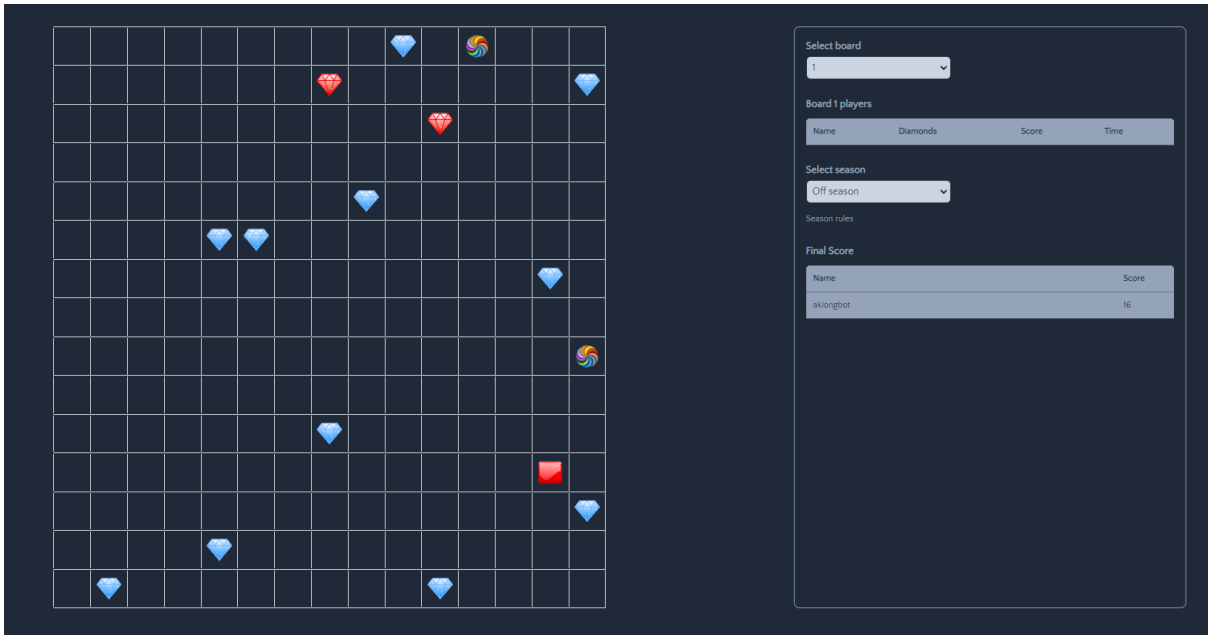
Test Case ini merupakan test case yang lebih menguntungkan dimana meskipun diamond lebih susah karena hanya ada 1 portal, namun lebih gampang untuk kembali ke base karena base berada ditengah sehingga memberikan kesempatan lebih banyak bagi bot untuk kembali ke base. Kondisi ini terbilang lumayan baik dimana dengan posisi base ditengah, bot dapat mensortir efisiensi yang ada dan menerapkan algoritmanya.

Foto Awal:



Gambar 4. 9 Test Case 1 Portal Base ditengah (Before)

Foto Akhir:



Gambar 4. 10 Test Case 1 Portal Base ditengah (After)

Seperti yang terlihat, dengan posisi base ditengah, bot lebih memiliki kesempatan untuk mengambil diamond dan menyimpannya dengan lebih cepat sehingga memungkinkan untuk menyimpan lebih banyak diamond. Dalam kondisi ini, diamond yang disimpan yaitu sebanyak 16, lebih unggul 6 diamond dari kondisi map diujung.

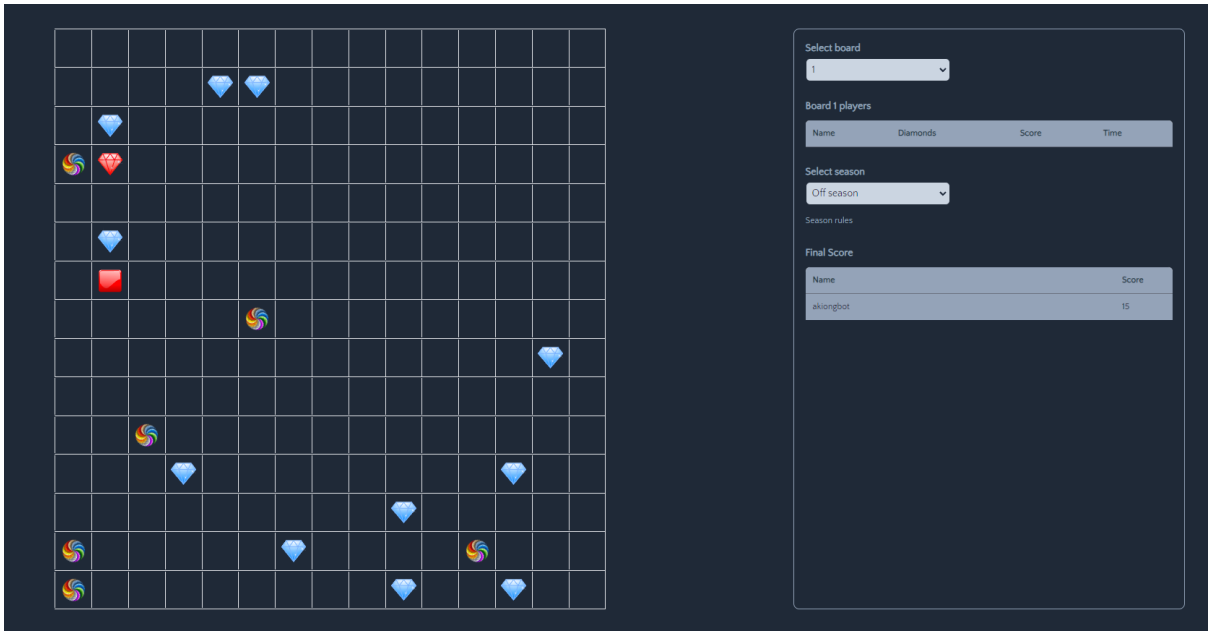
Test Case Posisi Base Di ujung dengan 3 portal

Test Case ini ditujukan untuk membuktikan bahwa greedy kita ini memang ditentukan pada faktor jarak dimana pada tahap clustering mempertimbangkan jarak ke cluster, jarak internal, dan jarak ke base. Dengan banyaknya portal, maka dapat memperpendek jarak ke base sehingga clustering lebih efektif. Kondisi ini bersifat netral karena meskipun posisi base diujung, banyaknya portal membuat bot mampu untuk menempuh jarak yang jauh.

Foto awal:

Gambar 4. 11 Test Case 3 Portal Base diujung (Before)

Foto akhir:



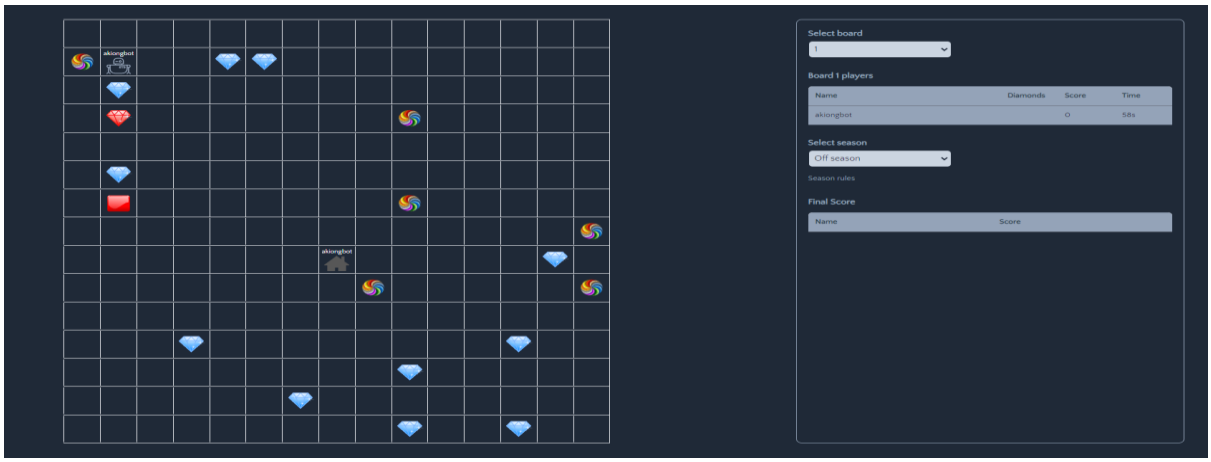
Gambar 4. 12 Test Case 3 Portal Base diujung (After)

Dengan ini, terbukti bahwa memang jarak berperan penting dan juga bagaimana pendekatan ini memeprhitungkan jarak karena dengan banyaknya portal maka hasil yang didapat juga lebih besar.

Test Case Posisi Base Ditengah dengan 3 portal

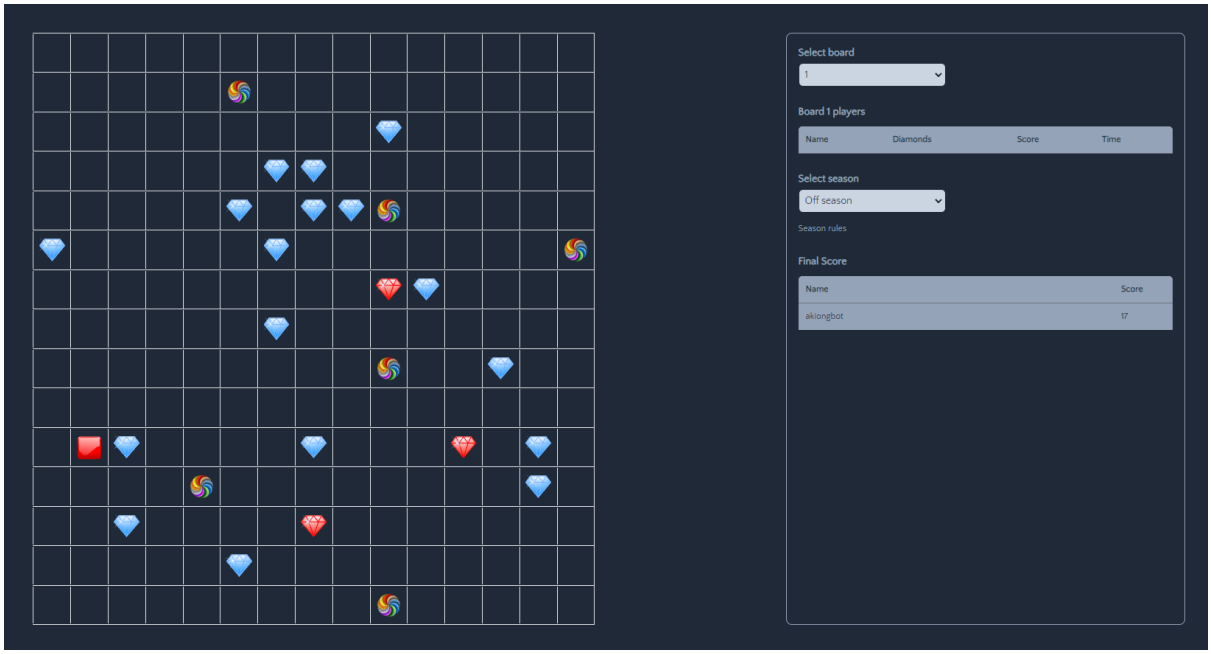
Sekarang akan dilakukan test case untuk posisi yang menguntungkan bagi bot dimana pada test case ini posisi ada ditengah dan portal lebih banyak sehingga lebih memungkinkan untuk mengambil diamond.

Foto Awal:



Gambar 4. 13 Test Case 3 Portal Base ditengah (Before)

Foto Akhir:



Gambar 4. 14 Test Case 3 Portal Base ditengah (After)

Terbukti bahwa test case ini memperoleh jumlah diamond terbanyak. Dengan jarak ke diamond yang dapat diperpendek akibat adanya diamond, serta posisi base yang menguntungkan. Kondisi ini merupakan kondisi yang optimal bagi bot karena bukan hanya bot mendapatkan

efisiensi yang baik, namun bot juga dapat menempuh jarak jauh dengan baik akibat adanya portal.

Kesimpulan Analisis

Dari berbagai analisis dan percobaan diatas, dapat diambil kesimpulan bahwa setiap fitur yang diharapkan kami untuk dijalankan pada bot sudah berjalan dengan sangat baik. Ditambah lagi, bot juga terbukti bisa memanfaatkan keuntungan yang ada serta menyesuaikan untuk berbagai kondisi sesuai dengan yang telah dibuktikan di test case diatas.

BAB V
KESIMPULAN DAN SARAN

KESIMPULAN

Sesuai dengan uji coba alternatif serta juga konsistensi jika dihadapkan dengan alternatif lain. Kami merasa bahwa alternatif yang baik yakni alterenatif yang memperhitungkan berbagai faktor terkait diamond, yakni jarak ke diamond, pelacakan kumpulan diamond, serta posisi diamond ke base. Alternatif ini terbilang optimal karena memastikan bahwa bot akan selalu bersifat *greedy* untuk nilai efektivitas yang ada pada tiap diamond sehingga memberikan alur jalan yang efektif untuk bot.

Kami mendapati bahwa berbagai algoritma greedy juga dapat memberikan hasil yang baik. Sesuai kondisi variabel lainnya (Game Object) dalam permainan, tentu akan mendukung maupun menghambat kemenangan. Namun, kami memilih *Clustering By Efficiency* sebagai pilihan akhir kami karena kami merasa bahwa pendekatan ini cukup konsisten dan tidak bergantung banyak pada keadaan game.

Clustering By Efficiency mengandalkan jumlah diamond pada custer yang dibagi dengan jarak ke cluster, jarak ke base, dan jarak antar diamond cluster guna memberikan efisiensi optimal. Dengan begitu bot tidak akan melangkah secara sia sia maupun semakin jauh dari base.

Kesimpulannya adalah kami berhasil memberikan pendekatan greedy yang baik dan juga optimal sesuai yang sudah dijelaskan di Bab IV dengan mengimplementasikan *Greedy By Clustering Efficiency.*

SARAN

Beberapa pelajaran yang kami dapatkan selama melakukan pengerjaan tugas besar ini yakni berupa pentingnya untuk selalu terfokus pada satu tujuan. Tujuan dari pembuatan bot ini yakni berupa pembuatan bot dengan efisisensi sebesar-besarnya sehingga faktor utama yang harus dipertimbangkan yakni berupa konsep pengambilan diamond. Namun sering kali kami merasa terlalu kompetitif sehingga melupakan konsep awal kami. Padahal, dalam permainan kompetitif terdapat beberapa faktor yang dapat menyebabkan kemenangan sehingga menurut kami tidak murni dari algoritma greedy. Oleh karena itu, kami memastikan bahwa dalam pemilihan alternatif bot kami merupakan bot yang dapat mengambil diamond dengan jumlah terbanyak dalam selang waktu tertentu, sesuai tujuan awal kami dalam konsep algoritma greedy.

LAMPIRAN

Pranala Repository:

https://github.com/FarelW/Tubes1_Akiong

Pranala Video Youtube:

https://youtu.be/8Kr5eX_Ookg?si=to-oYuELe5ywHZWH

DAFTAR PUSTAKA

Diamonds Game Engine. Accessed March 3, 2024. https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0.

Munir, Rinaldi. n.d. “Algoritma Greedy (Bagian 1).” Informatika. Accessed March 3, 2024. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf.

Munir, Rinaldi. n.d. “Algoritma Greedy (Bagian 2).” Informatika. Accessed March 3, 2024. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf.

Munir, Rinaldi. n.d. “Algoritma Greedy (Bagian 3).” Informatika. Accessed March 3, 2024. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf.