LAPORAN TUGAS BESAR I
IF2211 STRATEGI ALGORITMA

PEMANFAATAN ALGORITMA GREEDY DALAM PEMBUATAN BOT PERMAINAN DIAMONDS


Dipersiapkan oleh:

Kelompok Savage (48)

Nelsen Putra	13520130
Muhammad Gilang Ramadhan	13520137
Naufal Baldemar Ardanni	13521154

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

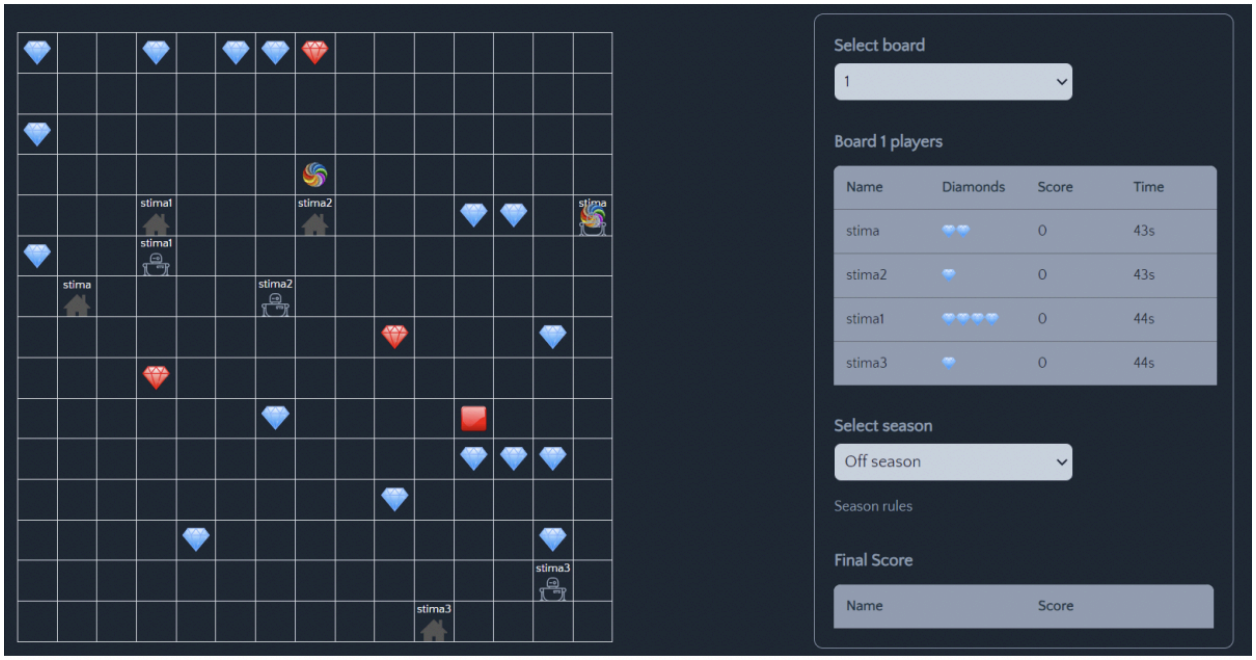
Institut Teknologi Bandung

2024

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

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.



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. Program permainan Diamonds terdiri atas:

1. Game engine, yang secara umum berisi:
    a. Kode backend permainan, yang berisi logic permainan secara keseluruhan serta API yang disediakan untuk berkomunikasi dengan frontend dan program bot

b. Kode frontend permainan, yang berfungsi untuk memvisualisasikan permainan

2. Bot starter pack, yang secara umum berisi:
    a. Program untuk memanggil API yang tersedia pada backend
    b. Program bot logic (bagian ini yang akan kalian implementasikan dengan algoritma greedy untuk bot kelompok kalian)
    c. Program utama (main) dan utilitas lainnya

Untuk mengimplementasikan algoritma pada bot tersebut, mahasiswa dapat menggunakan game engine dan membuat bot dari bot starter pack yang telah tersedia pada pranala berikut.

- Game engine: https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0
- Bot starter pack: https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1

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.

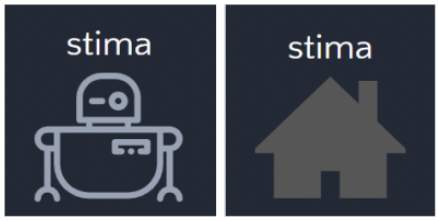
2. Red Button/Diamond Button

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

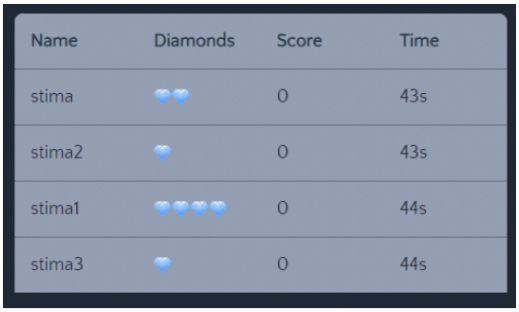
Terdapat 2 teleporter yang saling terhubung satu sama lain. Jika bot melewati sebuah teleporter maka bot akan berpindah menuju posisi teleporter yang lain.

4. Bots and Bases



Pada game ini kita akan menggerakkan bot untuk mendapatkan diamond sebanyak banyaknya. 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.

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.

Adapun panduan mengenai cara instalasi, menjalankan permainan, membuat bot, melihat visualizer/frontend, dan mengatur konfigurasi permainan dapat dilihat melalui tautan berikut.

Get Started with Diamonds

BAB II
LANDASAN TEORI

2.1. Algoritma *Greedy*

Algoritma *Greedy* merupakan salah satu algoritma/metode yang paling populer dalam pencarian solusi dari permasalahan yang membutuhkan optimisasi atau dengan kata lain, mencari solusi optimal. Permasalahan optimisasi dapat dibagi menjadi dua jenis, yaitu permasalahan memaksimasi (*maximization*) dan permasalahan meminimasi (*minimization*).

Algoritma *Greedy* yang memiliki prinsip "*Take what you can get now*!" dapat didefinisikan sebagai algoritma yang memecahkan permasalahan dengan cara membentuk himpunan solusi langkah demi langkah dari himpunan kosong. Pada setiap langkah, algoritma ini, sebagaimana prinsipnya, mengambil pilihan terbaik yang dapat diperoleh saat itu, tanpa memperhatikan konsekuensi ke depan dan tidak dapat mundur ke langkah sebelumnya. Kemudian, algoritma ini akan memilih solusi optimum lokal, dengan harapan, setiap langkah tersebut akan mengarah kepada optimum global. Dengan demikian, hasil solusi algoritma *greedy* belum tentu merupakan solusi optimum global (terbaik), bisa jadi merupakan solusi sub-optimum atau pseudo-optimum.

Berikut merupakan elemen-elemen yang menggambarkan sebuah Algoritma Greedy.

1. Himpunan Kandidat (*C*)
   Himpunan yang berisi kandidat yang akan dipilih pada setiap langkah.
2. Himpunan Solusi (*S*)
   Himpunan yang berisi kandidat yang sudah dipilih.
3. Fungsi Solusi
   Fungsi yang digunakan untuk menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi paling optimal.
4. Fungsi Seleksi (*Selection Function*)

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

5. Fungsi kelayakan (Feasibility)

   Memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi yang layak atau tidak.

6. Fungsi Obyektif

   Fungsi yang digunakan untuk memaksimumkan atau meminimumkan solusi yang kita miliki.

2.2. Cara Kerja Program Secara Umum

Pada projek permainan Diamonds, telah disediakan 2 jenis *starter pack* yang di dalamnya berisi masing-masing 2 bagian besar dari permainan ini, yaitu *Game Engine* dan Bot. Kedua bagian program ini dapat berkomunikasi menggunakan API yang telah tersedia pada program. *Game Engine,* seperti namanya, merupakan sebuah mesin yang dirancang secara khusus untuk menjalankan program atau aplikasi permainan. *Game engine* yang disediakan telah memiliki konfigurasi *default* permainan yang disimpan dalam *file* dengan format tersebut. Artinya, kita tidak perlu lagi mengatur *settings* yang ada dalam permainan tersebut dan bisa langsung menjalankan permainan yang ada. Dalam hal ini, bot dasar permainan sudah ada melalui konfigurasi tersebut dan tidak perlu ditambahkan dengan sendirinya. Hal yang perlu diubah atau dimodifikasi pada permainan ini hanyalah *logic* dari bot saja yang ditujukan untuk memperbesar kemungkinan menang dalam suatu permainan.

Pada bagian Bot, program akan memanggil API yang tersedia pada backend. Bot melakukan POST request terhadap endpoint API untuk mendaftarkan dirinya berdasarkan email dan password. Setelah terdaftar, bot dapat bergabung/join ke dalam permainan dengan melakukan POST terhadap endpoint join dan mendapatkan informasi dari board. Berdasarkan informasi tersebut, Bot dapat melakukan kalkulasi move berdasarkan logic masing-masing bot dan mengirimkannya melalui POST request ke backend melalui *endpoint move*. Backend akan memberikan response dengan body berisi kondisi board

setelah move tersebut, dan bot kembali mengalkulasi move selanjutnya dan diulang sampai permainan selesai.

Ketika kita ingin melakukan *testing* atau *sparring* dengan bot baru selain dari *Reference Bot* yang tersedia, kita bisa menambahkan bot ini dengan menyesuaikan jumlah pemanggilan command pada script run-bots.bat/run-bots.sh. Kemudian, kita cukup menjalankan script seperti biasa. Kita juga dapat mengubah konfigurasi board yang terdapat pada database dari aplikasi yang sudah memiliki aturan/nilai default pada konfigurasinya. Kita dapat menggunakan docker desktop dan PostgreSQL untuk melakukan update nilai pada konfigurasi board tersebut untuk memodifikasi permainan untuk berbagai jenis pengujian/percobaan.

BAB III

APLIKASI STRATEGI *GREEDY*

3.1. Pemetaan Algoritma *Greedy* pada Persoalan Permainan Diamonds

Dalam permainan Diamonds terdapat beberapa bot yang akan dipertandingkan dalam satu match. Bot-bot ini akan saling bertarung satu sama lain dalam memperebutkan skor tertinggi di match tersebut, dimana skor tertinggi akan dicapai melalui banyaknya diamond yang diperoleh oleh *player*. Sebagai perancang bot pada permainan ini, tentu kelompok mengimplementasikan berbagai strategi *Greedy* yang optimal yang kemudian dapat dipetakan ke dalam himpunan kandidat, himpunan solusi, fungsi seleksi, fungsi kelayakan, dan fungsi objektif. Untuk selengkapnya dapat dilihat melalui sub-bagian dibawah ini.

3.1.1. Pemetaan Umum Permasalahan Permainan

Elemen Algoritma *Greedy*	Pemetaan pada Permainan Diamonds
Himpunan Kandidat (*C*)	Himpunan kandidat merupakan pemetaan terhadap semua kemungkinan yang terjadi pada pengambilan keputusan untuk melangkah pada empat arah di sekitar Bot. Dalam pemetaan himpunan kandidat tersebut dilakukan pemetaan terhadap kemungkinan nilai profit yang dapat dihasilkan jika jika melangkah ke grid tersebut. Selain itu juga dilakukan heuristik untuk pengecekan state lawan dan kemungkinan potensial untuk mendapatkan profit pada satu state selanjutnya.
Himpunan Solusi (*S*)	Untuk setiap *state*, dilakukan pemilihan terhadap pergerakan bot pada suatu grid yang memiliki profit terbesar pada 4 arah di sekitar Bot, jika memang tidak memungkinkan atau dapat merugikan bisa saja tidak dipilih pergerakan apapun dalam pemilihan solusi untuk setiap langkah Bot.
Fungsi Solusi	Dilakukan pengecekan terhadap suatu solusi valid atau tidak di dalam pergerakan bot selanjutnya. Jika tidak, tentunya pergerakkan tersebut tidak akan dilakukan oleh bot.

Fungsi Seleksi (*Selection Function*)	Pemilihan pergerakan dilakukan sesuai dengan prioritas strategi yang telah dikonstruksi melalui Algoritma *Greedy* pada permainan Diamonds. Pembangunan prioritas dilakukan dalam bahasa pemrograman disusun berdasarkan nilai profit yang dihasilkan dari terbesar ke terkecil
Fungsi Kelayakan (*Feasibility*)	Memeriksa bahwa semua kondisi yang diperlukan untuk mengeksekusi perintah yang dipilih sudah terpenuhi atau belum. Validasi dilakukan terhadap setiap komponen pergerakan pada Bot, misalnya pergerakan bot terhadap batas *Board* pada permainan, pengecekan terhadap isi inventory bot yang jika penuh harusnya tidak akan mengambil diamond lagi karena akan *useless,* serta pengambilan langkah yang dilakukan akan berpotensi di-*tackle* oleh lawan atau tidak yang dapat merugikan Bot.
Fungsi Obyektif	Memenangkan permainan Diamonds dengan cara mengumpulkan diamond sebanyak banyaknya untuk berusaha mendapatkan skor tertinggi di antara pemain yang lain dalam pertandingan permainan yang dilaksanakan.

3.1.2. Pemetaan Strategi Pemrioritasan Pengumpulan Diamond ke dalam Algoritma *Greedy*

Elemen Algoritma *Greedy*	Pemetaan pada Permainan Diamonds
Himpunan Kandidat (*C*)	Pergerakan-pergerakan yang menghasilkan diamond tambahan pada gerakan selanjutnya. Untuk value yang didapatkan adalah 1 atau 2 untuk setiap grid.
Himpunan Solusi (*S*)	Pergerakan yang menghasilkan *diamond* paling banyak dan tidak di-*tackle* oleh lawan sesaat setelah mengambil *diamond* tersebut.
Fungsi Solusi	Memeriksa pergerakan yang dipilih adalah gerakan yang valid atau tidak dalam permainan. Jika tidak maka pergerakan tersebut tidak akan dipilih untuk pergerakan selanjutnya.
Fungsi Seleksi (*Selection*	Pemilihan perintah yang sesuai dengan prioritas

Function)	strategi greedy yang dilakukan, dalam hal ini prioritas yang dilakukan terhadap pengumpulan diamond adalah dengan cara mengecek untuk 4 arah pada bot apakah terdapat diamond atau tidak. Jika terdapat, maka ambil grid tetangga yang memiliki nilai diamond paling besar sambil memeriksa apakah pada grid tersebut musuh dapat melakukan tackle atau tidak. Kalau musuh dapat melakukan *tackle,* assign dengan nilai *profit* menjadi 0 sama dengan *profit* saat kondisi inventory bot kosong, jika tidak assign profit sementara bot menjadi jumlah diamond sekarang ditambah dengan diamond tambahan yang dihasilkan.
Fungsi Kelayakan (*Feasibility*)	Dilakukan pemeriksaan terhadap pengambilan diamond terhadap jumlah diamond yang tersedia untuk disimpan ke dalam inventory. Jika tidak tersedia maka, bot tidak akan berusaha mengambil diamond tersebut karena *useless*.
Fungsi Obyektif	Memaksimalkan jumlah diamond yang ada di inventory dengan cara mempertimbangkan besar profit yang dihasilkan jika mengambil suatu pergerakan. Karena jika bot tidak aman (di-*tackle* musuh maka bot di inventory-nya pun habis atau menjadi 0).

3.1.3. Pemetaan Strategi Pemrioritasan Perlindungan ke dalam Algoritma *Greedy*

Elemen Algoritma *Greedy*	Pemetaan pada Permainan Diamonds
Himpunan Kandidat (*C*)	Pergerakan-pergerakan yang melindungi bot dari tackle lawan, yaitu suatu state yang jika dilakukan pergerakan ke dalamnya maka akan ditabrak musuh dan semua diamond pun akan hilang. Berkaitan dengan pemetaan grid yang jarak dengan bot musuh adalah tidak sama dengan 2 satuan. Dilakukan penghindaran terhadap pendekatan 1 satuan terhadap Bot yang jaraknya 2 satuan tersebut.
Himpunan Solusi (*S*)	Pergerakan yang paling aman jika dipilih tidak menghasilkan kondisi dimana Bot akan ditabrak dalam kondisi terdapat diamond yang sedang

	dibawah oleh Bot (Karena akan menghasilkan profit kepada musuh).
Fungsi Solusi	Memeriksa keamanan bot kita dan memastikan pergerakan yang akan dilakukan terdapat pada grid yang ada, serta melakukan gerakan yang valid untuk dijalankan oleh *game engine*.
Fungsi Seleksi (*Selection Function*)	Pemilihan perintah dilakukan sesuai dengan prioritas strategi yang digunakan dalam bot, yaitu memeriksa semua state bot musuh yang bisa saja berpotensi menabrak bot kita, dengan cara menghitung jarak antara semua bot musuh dengan bot kita. Jika jaraknya adalah 2 satuan maka bot kita akan ditabrak bot musuh, jika tidak maka aman.
Fungsi Kelayakan (*Feasibility*)	Memeriksa bahwa semua kondisi yang diperlukan untuk melakukan pergerakan selanjutnya adalah sudah valid atau belum. Validasi dilakukan terhadap validitas dari potensi musuh yang bisa melakukan tackle terhadap bot kita di sepanjang 4 arah grid di sekitar bot kita.
Fungsi Obyektif	Memaksimalkan penghindaran terhadap pergerakan ke grid yang berpotensi untuk di-*tackle* oleh lawan jika bergerak menuju grid tersebut. Kemudian, dilakukan juga pengambilan grid yang berpotensi menghasilkan *profit*.

3.1.4. Pemetaan Strategi Pemrioritasan Kembali ke *Base* ke dalam Algoritma *Greedy*

Elemen Algoritma *Greedy*	Pemetaan pada Permainan Diamonds
Himpunan Kandidat (*C*)	Pergerakan-pergerakan yang jika dihitung jarak antara mencapai *base* atau mendapatkan *profit* tersebut, maka akan lebih dekat ke arah *base* sambil mengecek apakah terdapat diamond yang bisa ditaruh di *base* atau tidak. Jika tidak maka *Bot* tidak akan mungkin kembali ke *base*.
Himpunan Solusi (*S*)	Pergerakan yang paling dekat ke base dan terdapat diamond yang diperoleh di inventory Bot. Untuk setiap pergerakan yang paling dekat ke base tersebut merupakan pergerakan yang lebih dekat untuk menyimpan diamond daripada

	mengumpulkan profit selanjutnya.
Fungsi Solusi	Memeriksa bahwa perintah yang dipilih adalah perintah yang valid dan terdefinisi pada permainan Diamonds. Jika tidak maka perintah tersebut bukan merupakan perintah yang valid pada permainan.
Fungsi Seleksi (*Selection Function*)	Pemilihan pergerakan yang sesuai dengan prioritas strategi yang digunakan oleh bot, yang dalam hal ini dilakukan pembangunan prioritas yang mendahulukan bot untuk melakukan pergerakan yang lebih dekat terlebih dahulu dan memeriksa apakah menyimpan atau mengumpulkan profit dapat segera dilakukan dalam satu langkah pengambilan keputusan untuk langkah selanjutnya.
Fungsi Kelayakan (*Feasibility*)	Memeriksa bahwa semua pergerakan yang akan dilakukan untuk berpindah *grid* dalam memutuskan untuk kembali ke *base* adalah langkah yang valid. Validasi dilakukan terhadap validitas jarak terdekat antara bot, ketersediaan bot untuk segera menuju *base*, jumlah *profit*, dan keamanan Bot.
Fungsi Obyektif	Memaksimalkan kemungkinan untuk melakukan gerakan perpindahan menuju base berdasarkan jarak terdekat antara bot dan *profit*, keamanan bot, serta ketersediaan Diamond yang akan disimpan ke *base* pada setiap state yang akan dilalui bot.

3.1.5. Pemetaan Strategi Pemrioritasan Penyerangan terhadap Lawan ke dalam Algoritma *Greedy*

Elemen Algoritma *Greedy*	Pemetaan pada Permainan Diamonds
Himpunan Kandidat (*C*)	Pergerakan-pergerakan yang berkaitan penyerangan Bot lawan yang dilakukan untuk berusaha men-*tackle* lawan tersebut sehingga semua diamond yang dimiliki dapat diambil.
Himpunan Solusi (*S*)	Pergerakan terbaik dalam menyerang lawan yang menghasilkan profit maksimal sesuai dengan ketersediaan diamond Bot kita atau pun Bot lawan. Perintah terbaik yang dipilih sesuai dengan kondisi

Fungsi Solusi	Memeriksa bahwa pergerakan yang akan dilakukan merupakan perintah yang valid atau terdefinisi dalam permainan. Jika tidak, tentunya perintah ini bukan solusi yang valid untuk diambil oleh Bot.
Fungsi Seleksi (*Selection Function*)	Pemilihan pergerakan yang bisa mentackle lawan yang memiliki *profit diamond* yang didukung ketersediaan oleh kapasitas *diamond* Bot kita. Pembangunan prioritas dilakukan berdasarkan kondisi Bot kita dan Bot lawan yang ada pada suatu state. Untuk lawan yang bisa di-*tackle* itu adalah lawan yang berada pada jarak 1 satuan dari grid Bot kita dan memiliki diamond pada *inventory* mereka.
Fungsi Kelayakan (*Feasibility*)	Memeriksa bahwa semua pergerakan yang akan dilakukan adalah telah valid sebagai gerakan yang akan dilakukan selanjutnya. Validasi dilakukan terhadap jumlah diamond lawan yang akan di-*tackle* dan ketersediaan lawan di sepanjang *grid* tetangga (Dalam 4 arah) terhadap Bot kita.
Fungsi Obyektif	Memaksimalkan pengambilan grid yang dapat berpotensi mengambil diamond terbanyak dari Bot musuh di sekitar. Dalam hal ini, juga harus diminimalisir kemungkinan Bot kita untuk ditabrak oleh Bot lainnya, mengingat pertandingan yang dilakukan melibatkan lebih dari 2 Bot.

3.1.6. Pemetaan Strategi Pemrioritasan Potensi *Profit 1 State* ke Depan ke dalam Algoritma *Greedy*

Elemen Algoritma *Greedy*	Pemetaan pada Permainan Diamonds
Himpunan Kandidat (*C*)	Pergerakan-pergerakan yang dapat menghasilkan diamond untuk langkah-langkah kedepannya. Akan diambil pergerakan yang pada langkah kedepannya terdapat lebih banyak diamond pada tempat tersebut dengan memperhitungkan jarak yang diperoleh ke grid tersebut.
Himpunan Solusi (*S*)	Pergerakan terbaik dipilih sesuai dengan nilai terbesar dan jarak terpendek yang dapat diperoleh jika mencapai grid tersebut. Perintah terbaik ini

	dilakukan setelah tidak ada profit lain yang bisa didapatkan lagi, maka caranya ialah mengambil potensi profit terbesar yang bisa didapatkan.
Fungsi Solusi	Memeriksa bahwa pergerakan yang dipilih adalah perintah yang terdefinisi atau tidak di dalam permainan. Jika tidak, tentunya perintah ini bukan solusi yang valid untuk dijalankan oleh *Bot.*
Fungsi Seleksi (*Selection Function*)	Pemilihan pergerakan yang sesuai dengan prioritas strategi yang telah dibangun dalam implementasi permainan *Bot diamond* yaitu dengan memetakan setiap kemungkinan grid tetangga pada 1 state selanjutnya yaitu dapat melalui teleportasi item ataupun melalui pengambilan *Diamond Button* yang dapat mereset diamond yang telah ada.
Fungsi Kelayakan (*Feasibility*)	Memeriksa bahwa semua kondisi yang diperlukan untuk mengeksekusi perintah yang dipilih sudah terpenuhi atau belum. Validasi dilakukan terhadap validitas dari posisi diamond yang berpotensi mengandung profit kedepannya sambil memperhitungkan jaraknya *possible* atau tidak.
Fungsi Obyektif	Memaksimalkan jumlah potensi profit yang akan dihasilkan pada 1 state selanjutnya melalui peninjauan terhadap state setelah teleport atau terhadap suatu grid yang pada suatu state tetangganya tidak terdapat diamond. Maka sebenarnya state tersebut dapat menghasilkan profit pada state tetangga dari state tetangga tersebut.

3.2. Eksplorasi Alternatif Strategi *Greedy* pada Persoalan Permainan Diamonds

Pada permainan Diamonds, mekanisme permainan terbilang cukup random, karena terdapat sistem generate secara berkala pada setiap item yang ada pada board. Serta terdapat sistem gerakan yang dilakukan hampir bersamaan, dipengaruhi oleh waktu eksekusi masing-masing Bot untuk menentukan langkah selanjutnya yang akan diambil.

Untuk itu kami berusaha untuk membuat algoritma yang cukup efisien dari segi Time Complexity, Memory Complexity, dan Optimality Skor pada kemenangan pada Bot yang kami buat. Adapun untuk penjelasan terkait dengan Algoritma yang kami buat tersebut adalah sebagai berikut.

1. Urutan prioritas yang dilakukan secara terurut dari kecil ke besar, yaitu: pengecekan potensi profit 1 langkah setelah langkah yang akan diambil, pengecekan profit terhadap teleport, pengecekan profit terhadap pengumpulan diamond point 1, pengecekan profit terhadap pengumpulan diamond point 2, pengecekan keamanan dari Bot.
2. Untuk setiap profit pada suatu state akan diberi Bobot, Yang mana bobot tersebut dibagi menjadi 2, yaitu Bobot profit yaitu pada variabel *max_val_dirr* dan bobot potensi profit pada variabel *total_value_objektif*.
3. Untuk bobot profit, akan diinisialisasikan dengan banyaknya diamond awal pada suatu state, jika berhasil mendapatkan diamond, maka bobot tersebut akan ditambah sesuai dengan diamond yang didapatkan. Namun, jika tertabrak musuh sehingga semua diamondnya hilang, maka bobotnya akan langsung menjadi 0. Sedangkan untuk bobot potensial profit dihitung berdasarkan jarak antara Bot dengan item yang mengandung profit tersebut, yang mana value yang didapatkan adalah senilai MAX_GLOBAL_VALUE - Jarak antara kedua item tersebut dengan nilai MAX_GLOBAL_VALUE = 28 sebagai maksimum jarak antara kedua *object* dalam *game*, dan juga terdapat point sebagai nilai *point* yang merupakan nilai diamond yang didapatkan ketika mencapai posisi tersebut.
4. Untuk perbandingan yang dilakukan untuk masing-masing profit ialah diprioritaskan diambil max_val_dirr yang paling besar dalam setiap 4 arah kemungkinan gerakan. Kemudian jika ternyata tidak ada profit yang menguntungkan atau tidak diperoleh value apapun pada saat menyeleksi nilai terbesar pada max_vali_dirr terhadap kandidat solusi yang nilai awalnya diinisialisasikan dengan nilai jumlah diamond sekarang. Maka barulah melakukan seleksi terhadap nilai potensial profit dengan cara yang sama yaitu mengambil nilai yang paling besar, tetapi perbedaannya yaitu pada saat total value objektif potensial profitnya sama dengan nilai maksimum potensial profit yang sudah didapat pada saat pengecekan setiap profit pada setiap arah gerakan, maka ambil nilai potensial profit yang memiliki nilai point terbesar. Sehingga dengan kata lain, prioritas perbandingan yang dilakukan pada potensial *profit* adalah pada jaraknya terlebih dahulu, kemudian barulah pada nilai *point*-nya.

5. Untuk strategi keamanan yang dilakukan skemanya sama persis seperti yang dijelaskan dalam tabel pemetaan yaitu dengan menghindari menaruh Bot berjarak 1 satuan terhadap musuh pada pengambilan suatu langkah. Dan untuk penyerangan yaitu dengan menabrak Bot musuh yang jaraknya satu satuan terhadap Bot kita.
6. Untuk penggunaan teleport, digunakan oleh Bot untuk mempersingkat jarak menuju suatu posisi yang dapat menguntungkan. Jadi, pada implementasi algoritma kami, jika ingin menggunakan teleport, dilakukan observasi terhadap state yang ada pada teleport tujuan tersebut untuk dihitung kondisi *profit* atau keamanannya.

3.3. Analisis Efisiensi dan Efektivitas Strategi *Greedy*

3.3.1. Analisis Efisiensi

3.3.1.1. Functionality Pengecekan Possibility Teleport

Untuk strategi pengecekan kondisi pengambilan teleport berikut adalah Time Complexity yang diperoleh

- *Best case*: kondisi koordinat di sekitar teleport tujuan tidak memenuhi, maka efisiensi *O*(M*N), dengan M = 2, N =4.
- *Worst case*: Kondisi semuanya memenuhi, maka efisiensi O(M*N^2), dengan M = 2, N =4.

3.3.1.2. Functionality Pengecekan Possibility Diamond dengan Point 1 & 2

Dilakukan iterasi dengan stack yang mana untuk melakukan penghematan terhadap pemakaian *memory* program. Untuk setiap kemungkian posisi diamond, akan dilakukan iterasi terhadap 4 Arah tetangga dari Bot, maka Time Complexity yang didapatkan adalah O(4*N) = O(N). Perlu diperhatikan bahwa untuk N sendiri merupakan banyaknya diamond 1 atau 2 saja.

3.3.1.3. Functionality Pengecekan Possibility Potensi bahaya dan Profit terhadap semua Bot Musuh

Dilakukan iterasi dengan stack untuk setiap komponen *object* Bot musuh. Setelah itu dilakukan pengecekan 4 arah untuk setiap profit dan potensi keamanan dari Bot musuh secara bersamaan pada satu loop. Maka Time Complexity yang diperoleh adalah O(4*N) = O(N) dengan N adalah banyaknya musuh.

3.3.1.4. Strategi Pengecekan Fungsi Seleksi

Dilakukan seleksi pada 4 arah kemungkinan posisi Bot selanjutnya, setelah itu pada setiap arah tersebut akan dilakukan dilakukan pengecekan terhadap profit secara linear O(1), potensial profit pada 4 arah, yaitu O(1) (tetap linear), serta iterasi pada setiap kemungkinan posisi diamond button untuk melakukan generate yang menggagalkan profit pada musuh secara O(N). Maka Time Complexity yang diperoleh adalah O(4*N) = O(N) dengan N merupakan banyaknya diamond button.

Untuk keseluruhan strategi algoritma yang telah dibuat, kelompok kami telah menerapkan beberapa strategi seperti penghematan memori dengan *stack* yang dapat dihapus pada setiap iterasi sehingga tidak menumpuk di *memory*, serta penggunaan mapping dengan array dua dimensi pada indeksnya.

3.3.2. Analisis Efektivitas

Alternatif solusi yang telah disebutkan tadi menguntungkan jika Bot ingin mempertahankan kemenangan, walaupun tidak dengan skor yang terlalu mencolok, karena cukup diprioritaskan untuk melakukan *defend* untuk mempertahankan diamond yang telah diperoleh oleh Bot, serta dilakukan prioritas pergerakan ke tempat terdekat yang possible untuk dilakukan dengan mempertimbangkan profit dan keamanan yang diperoleh seperti yang telah dijelaskan di bagian pemetaan solusi dan alternatif strategi yang telah disebutkan sebelumnya.

Untuk kasus yang tidak menguntungkan mungkin lebih kearah randomisasi yang kerap dilakukan secara berkala pada permainan sehingga

memubuat perancangan keputusan yang telah dibuat sebelumnya menjadi tidak begitu akurat karena suatu saat dapat berubah karena terjadi *generate* secara random tiba-tiba.

3.4. Pemilihan Algoritma *Greedy*

Berdasarkan eksplorasi alternatif Algoritma *Greedy* di atas, kelompok kami memutuskan untuk menggabungkan strategi-strategi tersebut. Hal yang menjadi pertimbangan utama atas keputusan itu adalah fakta bahwa setiap strategi di atas hanya efektif dan dapat digunakan dalam beberapa kasus saja untuk setiap state yang dilalui. Dengan demikian, penggabungan strategi-strategi tersebut dinilai lebih mampu secara efektif untuk menangani seluruh kasus atau skema yang ada selama permainan ketimbang memilih salah satu yang terbaik di antaranya.

Pada implementasi program, akhirnya kami menggabungkan dan memodifikasi seluruh strategi yang telah kami buat. Untuk menangani kasus-kasus tertentu, kami berpikir bahwa diperlukan adanya pengaturan urutan prioritas strategi yang kemudian coba kami kombinasikan untuk menemukan mana kombinasi urutan yang paling optimal. Hasil yang kelompok dapatkan ialah kombinasi strategi dengan urutan sebagai berikut: strategi mengumpulkan diamond, strategi perlindungan keamanan, strategi penyerangan, serta strategi pemetaan profit & potensial profit.

Urutan prioritas yang kami tentukan terakhir kali berasal dari proses *trial and error* yang kami ujikan pada *reference bot*, serta melalui analisis matematika dengan meninjau kondisi Bot pada setiap state pada Grid tersebut. Adapun Algoritma *Greedy* yang kami pilih untuk diimplementasikan adalah berdasarkan fokus/*goal* utama dari kelompok untuk memenangkan pertandingan, yakni untuk memenangkan pertandingan yaitu bisa dengan meminimalkan point lawan ataupun memaksimalkan point Bot kami sendiri.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi Algoritma *Greedy*

Pada program kami, hasil implementasi algoritma *greedy* terdapat pada *file* savage.py. Di dalam *file* tersebut terdapat sebuah kelas utama yang berisi implementasi algoritma *Greedy* yang telah dirancang, yakni kelas SavageLogic dengan berbagai fungsi dan prosedur di dalamnya, di antaranya sebagai berikut.

4.1.1. Fungsi move_position

```
FUNCTION move_position(current_position, delta_x, delta_y):
    // Melakukan gerakan dengan mengembalikan posisi baru
    RETURN Position(current_position.y + delta_y, current_position.x +
delta_x)
END FUNCTION
```

4.1.2. Fungsi compute_distance

```
FUNCTION compute_distance(current_position, destination_position):
    // Menghitung jarak dua buah posisi
    RETURN absolute(current_position.x - destination_position.x) +
absolute(current_position.y - destination_position.y)
END FUNCTION
```

4.1.3. Fungsi status_coordinate_on_enemy

```
FUNCTION status_coordinate_on_enemy(current_position, enemy_position):
    // Melakukan pengecekan terhadap status koordinat terhadap musuh (Apakah
untung atau rugi jika bergerak ke koordinat tersebut)
    distance = compute_distance(current_position, enemy_position)
    IF distance EQUALS 1 THEN // Berpotensi ditackle musuh
        RETURN -1
    ELSE IF distance EQUALS 0 THEN // Berpotensi menackle musuh
        RETURN 1
    ELSE
        RETURN 0 // Tidak ada apa-apa
    END IF
END FUNCTION
```

4.1.4. Fungsi is_valid_coordinate

```
FUNCTION is_valid_coordinate(current_position, width, height):
```

```
    // Lakukan pengecekan terhadap koordinat yang valid (Tidak melewati batas
board)
    IF current_position.x < 0 OR current_position.x >= width THEN
        RETURN False
    END IF
    IF current_position.y < 0 OR current_position.y >= height THEN
        RETURN False
    END IF
    RETURN True
END FUNCTION
```

4.1.5. Fungsi store_all_component

```
FUNCTION store_all_component(board, current_name, current_id):
    // Menyimpan setiap item komponen di papan untuk mengurangi kompleksitas
waktu pemeriksaan
    FOR each i IN board.game_objects DO
        IF i.type EQUALS "TeleportGameObject" THEN
            append i.position to self.teleport_pos
        ELSE IF i.type EQUALS "DiamondButtonGameObject" THEN
            append i.position to self.diamond_button_pos
        ELSE IF i.type EQUALS "DiamondGameObject" THEN
            IF i.properties.points EQUALS 1 THEN
                append i.position to self.diamond1_pos
            ELSE
                append i.position to self.diamond2_pos
            END IF
            self.board_mapping_component[i.position.y][i.position.x] =
Item(i.id, i.position, i.type, i.properties.name, i.properties.points)
        ELSE IF i.type EQUALS "BaseGameObject" THEN
            IF i.properties.name EQUALS current_name THEN
                self.base_game_pos = i.position
            END IF
        ELSE IF i.type EQUALS "BotGameObject" THEN
            IF i.id NOT EQUALS current_id THEN
                append i to self.enemies
            END IF
        END IF
    END FOR
END FUNCTION
```

4.1.6. Fungsi next_move

```
FUNCTION next_move(board_bot, board):
    // Inisialisasi variabel lokal komponen
    current_name = board_bot.properties.name
    current_position = board_bot.position
    current_id = board_bot.id
    current_diamond = board_bot.properties.diamonds
    candidate_next_diamond = current_diamond
    max_val_dirr = [current_diamond, current_diamond, current_diamond,
current_diamond]
    height = board.height
    width = board.width
    total_value_objektif = [(0,0), (0,0), (0,0), (0,0)]
    is_valid = [False, False, False, False]
```

```
    next_pos = []
END FUNCTION
```

4.1.7. Prosedur mapping_diamonds

```
PROCEDURE mapping_diamonds():
    // Prosedur untuk memetakan berlian
    FOR i FROM 0 TO 3 DO
        temp_next_pos = move_position(current_position, directions[i][0],
directions[i][1])
        append temp_next_pos to next_pos
        IF is_valid_coordinate(temp_next_pos, width, height) THEN
            is_valid[i] = True
        END IF
    END FOR
END PROCEDURE
```

4.1.8. Prosedur check_possibility_teleport

```
PROCEDURE check_possibility_teleport():
    // Melakukan pengecekan terhadap kemungkinan teleport dan profitnya
    // Time Complexity O(M*N*N), N = 4, M = 2
    num_of_teleport = length(self.teleport_pos)
    FOR i FROM 0 TO num_of_teleport - 1 DO
        FOR k FROM 0 TO 3 DO
            IF is_valid[k] THEN
                // Jika ada teleport di sekitar koordinat bot
                IF compute_distance(next_pos[k], self.teleport_pos[i]) EQUALS
0 THEN
                    // Cek koordinat di sekitar teleport tujuan apakah ada
profit
                    pos_after_teleport = self.teleport_pos[(i + 1) MOD
num_of_teleport]
                    FOR j FROM 0 TO 3 DO
                        pos_geser_arround = move_position(pos_after_teleport,
self.directions[j][0], self.directions[j][1])
                        IF is_valid_coordinate(pos_geser_arround, width,
height) THEN
                            // Kondisi pengecekan jika sudah teleport, dan
ternyata basenya di sekitarnya, masukkan sebagai profit
                            IF current_diamond > 0 AND pos_geser_arround
EQUALS self.base_game_pos THEN
                                total_value_objektif[j] = (MAX_GLOBAL_VALUE -
2, current_diamond) // Profitnya MAX_GLOBAL_VALUE-2 karena jaraknya 2
terhitung sebelum teleport
                            END IF
                            // Jika sudah teleport, ada berlian di sekitarnya
dan mencukupi untuk diambil, masukkan sebagai profit
                            component_in_arround =
self.board_mapping_component[pos_geser_arround.y][pos_geser_arround.x]
                            IF component_in_arround.id IS NOT NULL AND
(current_diamond + component_in_arround.value) <= 5 THEN
                                total_value_objektif[j] = (MAX_GLOBAL_VALUE -
2, component_in_arround.value) // Profitnya MAX_GLOBAL_VALUE-2 karena
jaraknya 2 terhitung sebelum teleport
                            END IF
```

```
                        END IF
                    END FOR
                END IF
            END IF
        END FOR
    END FOR
END PROCEDURE
```

4.1.9. Prosedur check_possibility_of_diamond1

```
PROCEDURE check_possibility_of_diamond1():
    // Melakukan pengecekan terhadap kemungkinan mendapatkan diamond 1
    // Time Complexity O(N*4)
    WHILE length(self.diamond1_pos) != 0 DO
        diamond_pos = self.diamond1_pos.pop()
        FOR i FROM 0 TO 3 DO
            IF is_valid[i] THEN
                distance = compute_distance(next_pos[i], diamond_pos)
                // Jika masih bisa menampung, gass aja
                IF current_diamond + 1 <= 5 THEN
                    // Jika posisi selanjutnya di atas berlian
                    IF distance EQUALS 0 THEN
                        max_val_dirr[i] = current_diamond + 1
                    ELSE
                        // Jika tidak, hitung kemungkinan profit berdasarkan
jarak dan nilai berlian
                        value = MAX_GLOBAL_VALUE - distance
                        IF value > total_value_objektif[i][0] THEN
                            total_value_objektif[i] = (value, 1)
                        END IF
                    END IF
                END IF
            END IF
        END FOR
    END WHILE
END PROCEDURE
```

4.1.10. Prosedur check_possibility_of_diamond2

```
PROCEDURE check_possibility_of_diamond2():
    // Melakukan pengecekan terhadap kemungkinan mendapatkan diamond 2
    // Time Complexity O(4*N)
    WHILE length(self.diamond2_pos) != 0 DO
        diamond_pos = self.diamond2_pos.pop()
        FOR i FROM 0 TO 3 DO
            IF is_valid[i] THEN
                distance = compute_distance(next_pos[i], diamond_pos)
                // Jika masih bisa menampung, gass aja
                IF current_diamond + 2 <= 5 THEN
                    // Jika posisi selanjutnya di atas berlian
                    IF distance EQUALS 0 THEN
                        max_val_dirr[i] = current_diamond + 2
                    ELSE
                        // Jika tidak, hitung kemungkinan profit berdasarkan
jarak dan nilai berlian
                        value = MAX_GLOBAL_VALUE - distance
```

```
                        IF value > total_value_objektif[i][0] THEN
                            total_value_objektif[i] = (value, 2)
                        END IF
                    END IF
                END IF
            END IF
        END FOR
    END WHILE
END PROCEDURE
```

4.1.11. Prosedur check_possibility_enemy

```
PROCEDURE check_possibility_enemy():
    // Time Complexity O(4*N)
    count_potential_increase_diamond_of_enemy = 0
    WHILE length(self.enemies) != 0 DO
        enemy = self.enemies.pop()
        FOR i FROM 0 TO 3 DO
            // Dilakukan pengecekan terhadap status musuh
            // Kemungkinan berapa musuh yang bisa mendapatkan berlian untuk
dilakukan aksi reset berlian (jika ada tombol berlian di sekitarnya)
            enemy.position = move_position(enemy.position,
self.directions[i][0], self.directions[i][1])
            IF is_valid_coordinate(enemy.position, width, height) AND
board_mapping_component[enemy.position.y][enemy.position.x].id IS NOT NULL
THEN
                count_potential_increase_diamond_of_enemy += 1
            END IF

            // Dilakukan pengecekan terhadap bot kita
            IF is_valid[i] THEN
                IF enemy.properties.diamonds > 0 THEN
                    status_on_enemy = status_coordinate_on_enemy(next_pos[i],
enemy.position)
                    IF status_on_enemy EQUALS 1 THEN // Jika profit
                        max_val_dirr[i] += enemy.properties.diamonds
                    ELSE IF status_on_enemy EQUALS -1 THEN // Jika rugi
                        max_val_dirr[i] = 0
                    END IF
                END IF
            END IF
        END FOR
    END WHILE
    RETURN count_potential_increase_diamond_of_enemy
END PROCEDURE
```

4.1.12. Fungsi move_to_get_potential_profit_to_move

```
FUNCTION move_to_get_potential_profit_to_move():
    // Fungsi untuk melakukan pengecekan terhadap kemungkinan mendekati
posisi berlian
    // Time Complexity O(N), N=4
    temp_value_max = -1
    point_max = -1
    FOR i FROM 0 TO 3 DO
        IF current_diamond + total_value_objektif[i][1] <= 5 THEN
```

```
            IF total_value_objektif[i][0] > temp_value_max OR
               (total_value_objektif[i][0] EQUALS temp_value_max AND
total_value_objektif[i][1] > point_max) THEN
                temp_value_max = total_value_objektif[i][0]
                point_max = total_value_objektif[i][1]
                self.value_move = i
            END IF
        END IF
    END FOR
    RETURN temp_value_max
END FUNCTION
```

4.1.13. Fungsi selection_function

```
FUNCTION selection_function(count_potential_increase_diamond_of_enemy,
candidate_next_diamond):
    // Time Complexity O(4*N)
    delta_x = 0
    delta_y = 0

    // Check kemungkinan nilai diamond yang diperoleh dari salah satu dari 4
arah
    FOR i FROM 0 TO 3 DO
        IF max_val_dirr[i] > candidate_next_diamond THEN
            candidate_next_diamond = max_val_dirr[i]
            self.value_move = i
        END IF
    END FOR

    // Jika tidak bisa mendapatkan diamond (jumlah diamond tetap)
    IF candidate_next_diamond EQUALS current_diamond THEN
        temp_value_max = move_to_get_potential_profit_to_move()

        // Jika sudah memiliki diamond, coba kembali ke basis sambil
membandingkan lebih menguntungkan untuk kembali atau mencari potensi untuk
mendapatkan diamond
        IF current_diamond > 0 THEN
            distance_to_base = compute_distance(base_game_pos,
current_position)
            IF distance_to_base > 0 AND (MAX_GLOBAL_VALUE - temp_value_max)
>= distance_to_base THEN
                delta_x, delta_y = get_direction(current_position.x,
current_position.y, base_game_pos.x, base_game_pos.y)
                RETURN delta_x, delta_y
            END IF
        END IF

        // Jika musuh memiliki diamond dan kita dekat dengan tombol diamond
        // Coba ambil tombol diamond untuk mereset diamond sekitarnya (agar
musuh tidak mendapatkan diamond di sekitar atas, bawah, kiri, dan kanan)
        IF count_potential_increase_diamond_of_enemy > 0 THEN
            WHILE length(self.diamond_button_pos) != 0 DO
                diamond_button_pos = self.diamond_button_pos.pop()
                FOR i FROM 0 TO 3 DO
                    IF is_valid[i] THEN
                        distance = compute_distance(next_pos[i],
diamond_button_pos)
                        IF distance EQUALS 0 THEN
```

```
                            RETURN self.directions[i][0],
self.directions[i][1]
                        END IF
                    END IF
                END FOR
            END WHILE
        END IF
    END IF

    // Jika bergerak (tidak diam)
    IF self.value_move != -1 THEN
        delta_x = self.directions[self.value_move][0]
        delta_y = self.directions[self.value_move][1]
    END IF

    RETURN delta_x, delta_y

// Main Logic of the Savage Bot Move
IF current_diamond EQUALS 5 THEN // Kondisi khusus kalo udah punya 5 diamond,
balik aja ke base
    // Move to base
    SET base TO board_bot.properties.base
    SET goal_position TO base
    CALCULATE delta_x, delta_y USING get_direction(
        current_position.x,
        current_position.y,
        goal_position.x,
        goal_position.y
    )
    RETURN delta_x, delta_y
END IF

// Simpan semua komponen diamond supaya bisa dilakukan pengecekan dengan
cepat
// Pengecekan dengan indeks array, Time Complexity O(1)
FOR EACH i IN range(height) DO
    FOR EACH j IN range(width) DO
        board_mapping_component[i][j] = Item()
    END FOR
END FOR

// Candidate Checking Part
CALL store_all_component(board, current_name, current_id)

CALL mapping_diamonds()

CALL check_possibility_teleport()

CALL check_possibility_of_diamond1()

CALL check_possibility_of_diamond2()

SET count_potential_increase_diamond_of_enemy TO check_possibility_enemy()

// Selection Part
SET delta_x, delta_y TO
selection_function(count_potential_increase_diamond_of_enemy)

RETURN delta_x, delta_y
```

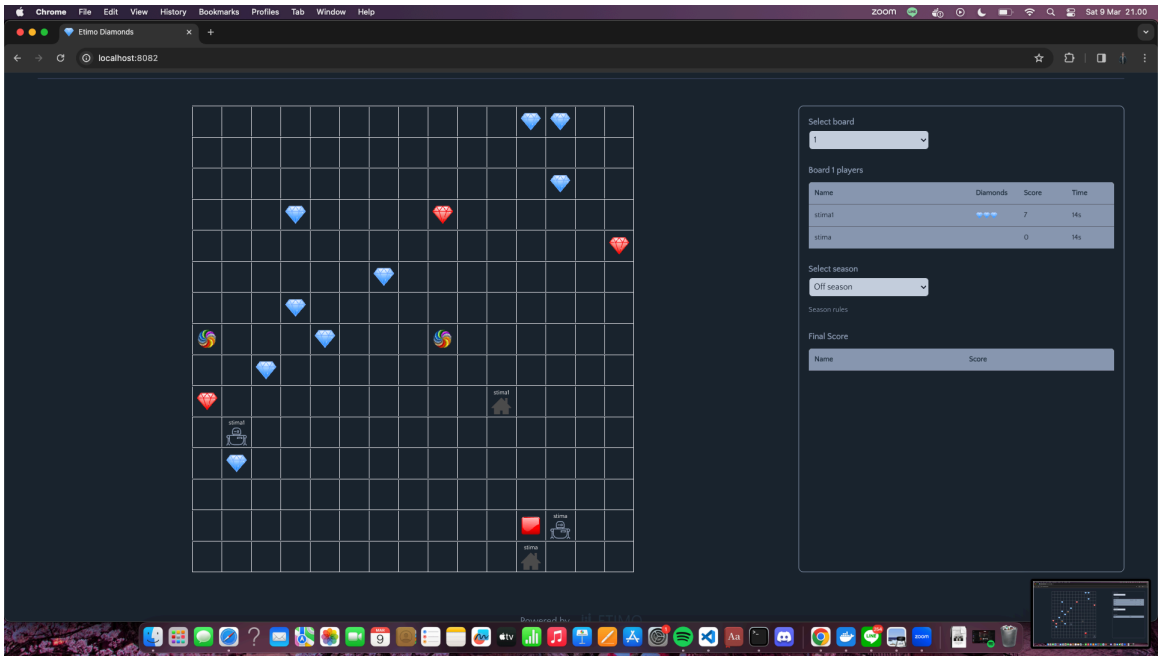
4.2. Struktur Data Program Bot Diamonds

Pada permainan “Diamonds” ini, struktur data yang digunakan berbasis pada kelas. Di dalam permainan ini, telah terdefinisi 2 kelas yaitu kelas untuk Bot logic kami yaitu Savage dan kelas Items sebagai data class. Untuk struktur kelas yang dibuat dapat dilihat melalui pseudocode yang telah ditulis pada bagian sebelumnya. Sementara untuk struktur data yang digunakan meliputi array dan stack untuk menampung kontainer pada setiap object yang diklasifikasikan seperti diamond, posisi teleport item, data musuh, dan mapping terhadap komponen diamond dalam bentuk array dua dimensi untuk memperkecil time *complexity* yang diperlukan. Untuk main logic dari Bot terdapat pada method next_move yang mengimplementasikan gerakan selanjutnya yang akan dilkukan.

4.3. Analisis Pengujian Algoritma *Greedy*

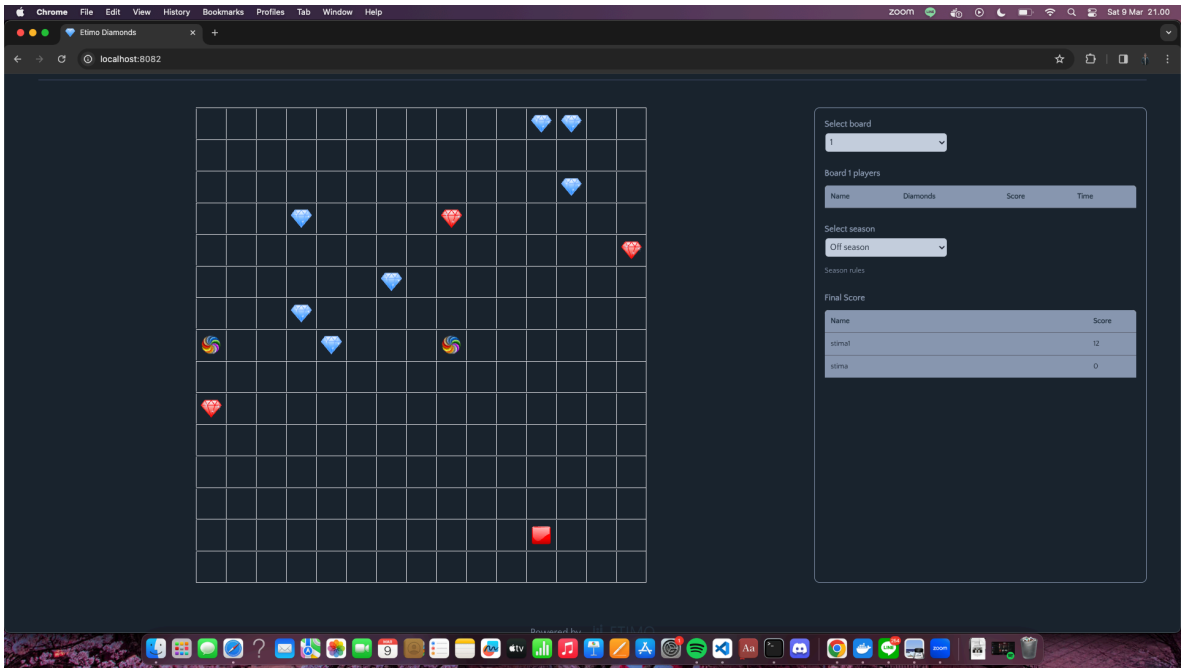
Percobaan untuk mengambil data uji ini dilakukan cukup dengan menjalankan `run-test-bots.sh` pada folder starter-pack (src jika sudah diganti) yang berisi bot yang telah dikembangkan. Pada kesempatan kali ini, kami melakukan tiga kali pengujian terhadap bot yang telah kami rancang.

4.3.1. Pengujian I



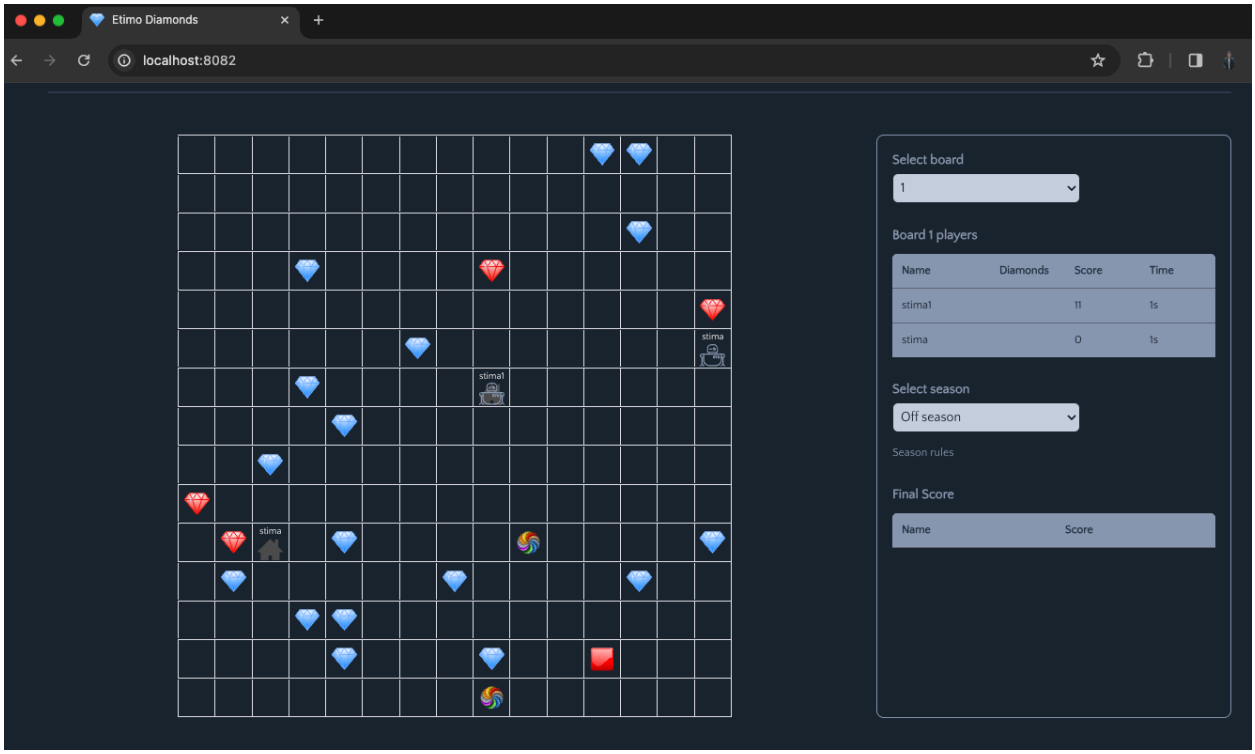
Pada pengujian pertama, *bot* yang kami rancang berhasil mengalahkan *reference bot* yang disediakan oleh denga 7 diamond.

4.3.2. Pengujian II



Namun, pada pengujian kedua, *bot* yang kami rancang menang terhadap *reference bot* dalam 12 diamond. Dapat dilihat bahwa stratei kami membuat bot musuh tidak mendapatkan diamond sama sekali karena sudah disetting untuk meminimumkan skor lawan.

4.3.3. Pengujian III



Namun, pada pengujian kedua, *bot* yang kami rancang kembali memberikan kemenangan telak 11 diamond terhadap bot musuh dengan 0 diamond.

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Telah berhasil diimplementasikan sebuah program berupa bot pada permainan Diamonds yang dirancang dan dikembangkan untuk memenangkan permainan sesuai dengan yang diminta dalam spesifikasi Tugas Besar 1 IF2211 Strategi Algoritma Semester 2 Tahun 2023/2024. Hal mengenai bot permainan yang berhasil diimplementasikan dalam program ini meliputi:

1. Konsep algoritma Greedy dan penerapannya dalam implementasi bot pada permainan Diamonds,
2. Eksplorasi alternatif strategi yang berkaitan dengan permasalahan optimisasi, baik itu *maximization* maupun *minimization,*
3. Analisis efisiensi dan efektivitas strategi *Greedy* yang berhasil dieksplorasi dan ditemukan untuk kemudian ditentukan mana yang layak dipakai pada suatu kondisi tertentu,
4. Pemetaan dan pengurutan prioritas strategi berdasarkan kondisi bot, termasuk *inventory* dan lingkungan di sekitar bot,
5. Implementasi program dengan paradigma pemrograman berorientasi objek untuk mengembangkan bot pada permainan Diamonds.

Semua implementasi dari konsep-konsep di atas kemudian berhasil digunakan untuk menyelesaikan persoalan yang ada di dalam spesifikasi maupun *game rules* dari permainan Diamonds itu sendiri. Setidaknya terdapat 9 *command* utama (pada spesifikasi permainan terdapat 11 *command*, namun yang kami pilih untuk kami gunakan hanya 9) yang dapat digunakan pada bot permainan Diamonds kami. *Command-command* ini dapat digunakan dan telah diatur pada bot permainan Diamonds kami yang sudah diimplementasikan sedemikian rupa dengan memanfaatkan strategi *Greedy* yang telah kelompok kami eksplorasi. Dengan demikian, strategi yang kami gunakan diharapkan dapat memampukan *player* untuk memenangkan pertandingan.

Dengan mengimplementasikan berbagai strategi pemrioritasan pada bot permainan Diamonds, khususnya dengan menggunakan konsep algoritma *Greedy*, kita dapat memecahkan permasalahan optimisasi yang dapat ditemukan ketika melakukan eksplorasi alternatif strategi untuk memenangkan permainan. Konsep yang telah diajarkan di perkuliahan IF2211 dapat dengan baik diterapkan dalam pengerjaan Tugas Besar 1 IF2211 Strategi Algoritma ini. Selain melakukan eksplorasi terhadap berbagai alternatif algoritma *Greedy* yang dapat dipakai, kelompok juga mengatur skala prioritas terhadap setiap strategi yang telah ditemukan. Bahkan, ada pula strategi-strategi yang

dianggap *feasible* dan mungkin untuk direalisasikan, namun tidak kami implementasikan mengingat strategi tersebut tidak memberikan *impact* yang signifikan terhadap kemenangan bot pada permainan Diamonds ini. Hal ini tentu kami putuskan setelah melakukan analisis efisiensi dan efektivitas strategi *Greedy* yang ada.

Dengan demikian, kelompok menyimpulkan bahwa dengan mengerjakan Tugas Besar 1 IF2211 Strategi Algoritma Semester 2 Tahun 2023/2024 ini, dapat diketahui bahwa untuk menyelesaikan suatu masalah yang mungkin ditemukan dalam kehidupan sehari-hari, dalam hal ini misalnya, memenangkan permainan mengumpulkan diamond sebanyak-banyaknya pada permainan Diamonds, dapat diimplementasikan berbagai strategi pada bot dengan mencari solusi paling optimal dari suatu permasalahan yang membutuhkan optimisasi sebagai bentuk penerapan dari konsep algoritma *Greedy* yang telah dipelajari pada kuliah IF2211.

5.2. Saran

Tugas Besar 1 IF2211 Strategi Algoritma Semester 2 Tahun 2023/2024 menjadi salah satu proses pembelajaran bagi kelompok dalam menerapkan ilmu-ilmu yang diajarkan pada kuliah maupun melakukan eksplorasi materi secara mandiri. Berikut ini merupakan sejumlah saran dari kelompok untuk pihak-pihak yang ingin melakukan atau mengerjakan hal serupa.

1. Program yang diminta adalah program dengan menggunakan bahasa pemrograman Python. Kelompok merekomendasikan agar disediakan waktu yang cukup untuk melakukan eksplorasi terkait bahasa sekaligus paradigma pemrograman yang digunakan sebelum mengimplementasikannya ke dalam sebuah program. Pemilihan paradigma yang tepat serta penguasaan terhadap *syntax* bahasa pemrograman yang baik akan meningkatkan efektivitas kerja tim dalam pembuatan suatu program. Di samping itu, perlu dipertimbangkan pula waktu yang dimiliki untuk melakukan eksplorasi terhadap suatu bahasa pemrograman atau *framework* tertentu sehingga tidak membebani *programmer* dalam pengerjaan proyek dengan jangka waktu yang singkat. Dengan demikian, manajemen waktu juga menjadi bagian yang sangat penting dalam pembuatan program ini. Gunakan kemampuan berpikir serta bekerja yang elaboratif dan koordinatif di antara seluruh anggota proyek yang terlibat.
2. Modularitas menjadi hal yang krusial dalam menciptakan suatu program secara efektif dan efisien. Dalam jangka waktu yang singkat, pemrograman secara modular dapat membantu *programmer* untuk memudahkan proses pencarian kesalahan/*error* serta *debugging*. Pada dasarnya, memrogram secara modular berarti memecah-mecah program menjadi modul-modul kecil di mana masing-masing modul berinteraksi melalui antarmuka modul. Masalah yang awalnya kompleks dapat dibagi menjadi bagian-bagian kecil yang lebih sederhana

dan dapat diselesaikan dalam lingkup yang lebih kecil. Akibatnya, apabila terdapat *error/bug* pada program, kesalahan dapat dengan mudah ditemukan karena alur logika yang jelas serta dapat dilokalisasi dalam satu modul. Lebih dari pada itu, modifikasi program dapat dilakukan tanpa mengganggu *body* program secara keseluruhan. Oleh karena itu, kelompok sangat menyarankan untuk melakukan pemrograman secara modular dalam mengimplementasikan fungsi-fungsi pada bot dalam permainan Diamonds.

3. Penting bagi kelompok untuk memiliki strategi serta distribusi tugas yang baik. Ketika membuat program dalam sebuah tim, kesamaan cara menulis kode serta kemampuan untuk menulis komentar menjadi hal yang sangat penting. Hal ini diperlukan agar memudahkan anggota kelompok dalam menyatukan dan melanjutkan sebuah program. Kemampuan tersebut tentunya didukung juga dengan adanya *version control system* yang baik yang dapat digunakan oleh *programmer* dalam membuat sebuah program secara bersama-sama. Untuk itu, kami sangat menyarankan GitHub untuk digunakan sebagai *version control system* dalam pengerjaan tugas-tugas besar pada mata kuliah IF2211 ini, maupun pada pembuatan program dan pengerjaan proyek yang lainnya.
4. Kelompok menyadari bahwa pada implementasi bot pada permainan Diamonds yang telah kami buat, masih banyak aspek yang dapat dikembangkan lebih lagi. Salah satunya ialah dengan mengoptimalkan algoritma *Greedy* yang digunakan agar bot pada permainan Diamonds kami ini dapat memenangkan permainan dengan mengalahkan bot lawan atau *player* lain. Hal ini tentu menjadi ruang untuk *programmer* agar dapat melakukan improvisasi terhadap implementasi dan pengembangan program pada bot permainan Diamonds yang telah diciptakan, terutama dalam hal eksplorasi serta analisis efisiensi strategi.

LAMPIRAN

Link *repository* GitHub:

https://github.com/NoDrop3011/Tubes1_Savage

Link *video* YouTube:

https://youtu.be/GgN5ofoHDuU

DAFTAR PUSTAKA

Munir, Rinaldi. (2020). Algoritma Greedy (Bagian 1). Institut Teknologi Bandung. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf. Diakses pada 2 Maret 2024.

Munir, Rinaldi. (2020). Algoritma Greedy (Bagian 2). Institut Teknologi Bandung. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf. Diakses pada 2 Maret 2024.

Munir, Rinaldi. (2020). Algoritma Greedy (Bagian 3). Institut Teknologi Bandung. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag3.pdf. Diakses pada 2 Maret 2024.