

Laporan Tugas Besar I
IF2211 Strategi Algoritma Semester II Tahun 2022/2023
Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan “Galaxio”



Disusun Oleh :

Kelompok greed.axio

Husnia Munzayana	13521077
Bagas Aryo Seto	13521081
Cetta Reswara Parahita	13521133

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Bandung
2022/2023

DAFTAR ISI

DAFTAR ISI	2
BAB 1 DESKRIPSI TUGAS	3
BAB 2 LANDASAN TEORI	4
3.1. Algoritma Greedy	4
3.2. Permainan Galaxio	4
3.3. Cara Kerja Program	6
BAB 3 APLIKASI STRATEGI GREEDY	8
4.1. Elemen Algoritma Greedy Pada Permainan Galaxio	8
4.2. Alternatif Solusi Greedy	9
4.3. Analisis Solusi Greedy Terbaik	13
BAB 4 IMPLEMENTASI DAN PENGUJIAN	15
4.1. Pseudocode	15
4.2. Flowchart	23
4.3. Struktur Data	24
4.4. Pengujian	25
BAB 5 PENUTUP	27
5.1. Kesimpulan	27
5.2. Saran	27
5.3. Refleksi	27
DAFTAR PUSTAKA	28
LAMPIRAN	29

BAB 1

DESKRIPSI TUGAS

Galaxio adalah sebuah *game battle royale* yang mempertandingkan bot kapal Anda dengan beberapa bot kapal yang lain. Setiap pemain akan memiliki sebuah bot kapal. Tujuan dari permainan adalah agar bot kapal Anda tetap hidup hingga akhir permainan. Agar dapat memenangkan pertandingan, setiap bot harus mengimplementasikan strategi tertentu untuk dapat memenangkan permainan.

Pada tugas besar pertama IF2211 Strategi Algoritma ini, gunakanlah sebuah game engine pada <https://github.com/EntelectChallenge/2021-Galaxio> yang mengimplementasikan permainan Galaxio. Tugas mahasiswa adalah mengimplementasikan bot kapal dalam permainan *Galaxio* dengan menggunakan strategi greedy untuk memenangkan permainan. Untuk mengimplementasikan bot tersebut, mahasiswa disarankan melanjutkan program yang terdapat pada *starter-bots* dalam *starter-pack*. Spesifikasi permainan yang digunakan pada tugas besar ini disesuaikan dengan spesifikasi yang disediakan oleh *game engine Galaxio*.

Strategi greedy yang diimplementasikan tiap kelompok harus dikaitkan dengan fungsi objektif dari permainan itu sendiri, yaitu memenangkan permainan dengan cara mempertahankan kapal pemain paling terakhir untuk hidup. Salah satu contoh pendekatan greedy yang bisa digunakan (pendekatan tak terbatas pada contoh ini saja) adalah menghindari objek yang dapat mengurangi ukuran kapal. Buatlah strategi greedy terbaik, karena setiap bot dari masing-masing kelompok akan diadu dalam suatu kompetisi. Strategi greedy harus dijelaskan dan ditulis secara eksplisit pada laporan, karena akan diperiksa pada saat demo apakah strategi yang dituliskan sesuai dengan yang diimplementasikan. Tiap kelompok dapat menggunakan kreativitas mereka dalam menyusun strategi greedy untuk memenangkan permainan. Implementasi pemain harus dapat dijalankan pada game engine yang telah disebutkan pada spesifikasi tugas besar, serta dapat dikompetisikan dengan pemain dari kelompok lain.

BAB 2

LANDASAN TEORI

3.1. Algoritma Greedy

Algoritma Greedy adalah metode pemecahan masalah dengan pendekatan optimasi. Permasalahan optimasi ini dapat berupa permasalahan maksimasi atau minimasi. Greedy secara bahasa berarti rakus atau tamak. Proses pemecahan permasalahan dengan Algoritma Greedy dilakukan langkah demi langkah sedemikian sehingga pada setiap langkah kita dapat menentukan solusi paling optimal yang saat itu memungkinkan tanpa menghiraukan apa yang akan terjadi di masa mendatang. Prinsip ini juga dikenal sebagai prinsip "*Take what you can get now!*". Pemilihan nilai optimum lokal pada setiap langkah ini diharapkan dapat mengarah pada solusi optimum global untuk keseluruhan permasalahan. Namun, tidak semua persoalan memiliki solusi greedy yang lebih optimum dari strategi penyelesaian permasalahan yang lain. Pendekatan algoritma greedy dalam penyelesaian suatu persoalan bersifat intuitif sehingga setiap orang mungkin memiliki hasil pendekatan yang berbeda satu sama lain.

Algoritma greedy melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C; yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu S menyatakan suatu solusi dan S dioptimisasi oleh fungsi objektif. Dari pengertian algoritma greedy tersebut, didapatkan elemen-elemen algoritma greedy sebagai berikut:

1. Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap Langkah (misal: simpul/sisi di dalam graf, job, task, koin, benda, karakter, dsb)
2. Himpunan solusi, S : berisi kandidat yang sudah dipilih
3. Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi
4. Fungsi seleksi (selection function): memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.
5. Fungsi kelayakan (feasible): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak)
6. Fungsi objektif : memaksimumkan atau meminimumkan

3.2. Permainan Galaxio

Galaxio adalah permainan *battle royale* yang mempertandingkan minimum 2 bot kapal dengan setiap bot menjalankan command-command yang telah diimplementasikan dengan algoritma yang berbeda-beda. Pemenang dari permainan ini adalah pemain terakhir yang berhasil bertahan hidup hingga akhir

game. Jika terdapat dua pemain terakhir yang kalah secara bersamaan, pemenang akan ditentukan berdasarkan skor yang didapat sepanjang permainan.

Permainan ini dimainkan pada peta kartesian berbentuk lingkaran dengan radius tertentu dengan pusat peta berada pada koordinat (0,0). Radius peta akan terus mengecil sepanjang permainan. Terdapat beberapa objek pada peta, baik berupa object yang dapat menguntungkan player maupun object rintangan yang dapat menghambat player, antara lain :

1. *Players* adalah kumpulan beberapa bot kapal yang bertanding.
2. *Food* adalah objek yang dapat dikumpulkan untuk menambah ukuran kapal.
3. *Superfood*, sama seperti food tetapi dapat menambah ukuran kapal sebanyak dua kali lipat dari tambahan ukuran dari konsumsi food.
4. *Torpedo Salvo* dapat ditembakkan oleh setiap kapal untuk mengurangi ukuran setiap objek yang dilaluinya.
5. *Supernova* adalah senjata yang muncul sekali dalam satu permainan. Supernova dapat ditembakkan untuk memunculkan daerah gas cloud pada koordinat tertentu.

Pada permainan ini terdapat beberapa obstacle yang dapat memengaruhi posisi, kecepatan, maupun perubahan ukuran bot kapal yang melaluinya, antara lain:

1. *Wormholes* adalah objek berpasangan yang saling terhubung sehingga dapat memindahkan player yang mendekatinya
2. *Gas Cloud* adalah zona berbahaya yang dapat mengurangi ukuran player yang melaluinya
3. *Asteroid Fields* adalah zona yang dapat memperlambat kecepatan player yang melaluinya.
4. *Torpedo Salvo* adalah objek yang dapat mengurangi ukuran player maupun objek lain kecuali wormHoles sejumlah ukuran torpedo pada saat itu. Torpedo Salvo memiliki ukuran 10 ketika diluncurkan kemudian bergerak lurus dengan kecepatan 60.
5. *Teleport* adalah objek yang dapat diluncurkan setiap player untuk berpindah tempat.
6. *Supernova* adalah senjata yang muncul sekali dalam satu permainan. Pengambilan Supernova memungkinkan player untuk menembakkan Supernova Bomb. Supernova Bomb meluncur seperti Torpedo, tetapi tidak bertabrakan dengan objek lain. Player yang menembakkan Supernova Bomb kemudian dapat meledakannya dan menyebabkan kerusakan pada player lain di zona ledakan. Supernova juga meninggalkan Gas Cloud di zona ledakannya.

Pada awal permainan, setiap bot kapal memiliki ukuran 10 dan berjalan dengan kecepatan 20. Pergerakan bot selanjutnya ditentukan melalui method `computeNextPlayerAction` yang akan dipanggil setiap tick untuk menentukan heading atau arah bot berjalan serta action setiap langkahnya. Pada fungsi inilah akan diimplementasikan algoritma greedy untuk menentukan aksi bot kapal setiap tick. Pada setiap tick akan dilakukan analisis terhadap keadaan sekitar untuk kemudian program menentukan heading dan action bot. Pilihan action atau command yang dapat dilakukan bot antara lain :

- Forward
- Stop
- StartAfterburner
- StopAfterburner
- FireTorpedoes
- FireSupernova
- DetonateSupernova
- FireTeleporter
- Teleport
- ActivateShield

Permainan ini berakhir jika tersisa maksimum 1 bot kapal sebagai pemenang. Namun, terdapat kemungkinan dua bot terakhir mati secara bersamaan. Jika hal ini terjadi, penentuan pemenang dilakukan berdasarkan skor dengan ketentuan sebagai berikut :

- a) Memangsa kapal pemain lain = 10 poin
- b) Mengonsumsi makanan = 1 poin
- c) Melintasi wormhole = 1 poin

3.3. Cara Kerja Program

Program Galaxio terdiri dari beberapa komponen antara lain :

1. Engine, berisi implementasi aturan dalam permainan serta implementasi action kapal.
2. Runner, berisi implementasi untuk menjalankan permainan dengan memanggil setiap bot untuk kemudian diteruskan ke Engine.
3. Logger, berguna untuk menangkap setiap perubahan status game dan menyimpan jalannya bot pada file log di akhir game.
4. Starter-Bots, berisi fungsi-fungsi dasar untuk membangun bot.
5. en-compose , berisi file yang diperlukan untuk menjalankan game di docker.

Program ini dijalankan dengan urutan aktivasi komponen sebagai berikut :

1. Jalankan runner untuk membuat permainan pada *hostname* tertentu.
2. Jalankan engine untuk melakukan koneksi semua bot dengan permainan.
3. Jalankan logger untuk melakukan koneksi dengan runner.
4. Jalankan setiap bot.
5. Setelah semua bot telah terkoneksi, game akan dimulai.
6. Bot yang telah terkoneksi akan menerima status setiap tick dari runner, serta bot juga akan mengirim aksinya pada runner.
7. Permainan akan berakhir jika tersisa maksimum satu bot kapal sebagai pemenang.
8. Setelah permainan berakhir, akan tercipta sebuah file log file berisi status permainan setiap tick.

BAB 3

APLIKASI STRATEGI GREEDY

4.1. Elemen Algoritma Greedy Pada Permainan Galaxio

Pada permainan Galaxio, setiap pemain memiliki sebuah bot kapal yang saling bertahan dan menyerang agar dapat menjadi pemain terakhir yang masih bertahan hidup. Dalam proses bertahan dan menyerang tersebut terdapat beberapa obstacle maupun objek yang dapat dimanfaatkan player. Ketika kapal melewati obstacle akan mendapatkan damage sesuai dengan jenis obstacle yang dilaluinya. Selain itu, player juga dapat menyerang player lain dengan memanfaatkan beberapa objek yang tersedia.

Pada setiap tick atau langkah, pemain akan menentukan aksi yang akan dilakukan oleh bot kapal melalui program. Aksi tersebut mencakup heading serta command aksi. Penentuan heading serta aksi pemain dilakukan dengan mengimplementasikan *algoritma greedy*. Elemen Algoritma greedy dari permainan Galaxio antara lain :

1. Himpunan Kandidat

Himpunan kandidat meliputi pilihan *action* berupa command-command yang dapat dilakukan oleh bot secara bebas yang meliputi:

- 1) Forward
- 2) Stop
- 3) StartAfterburner
- 4) StopAfterburner
- 5) FireTorpedoes
- 6) FireSupernova
- 7) DetonateSupernova
- 8) FireTeleporter
- 9) Teleport
- 10) ActivateShield

Selain itu himpunan kandidat solusi juga meliputi himpunan 360 derajat arah/*heading* yang dapat dipilih oleh bot untuk menjadi arah geraknya pada tick berikutnya.

2. Himpunan Solusi

Himpunan solusi adalah himpunan *action* yang terpilih dan *heading* yang dijadikan tujuan pada suatu *tick* tertentu.

3. Fungsi Solusi

Memeriksa apakah *action* dan *heading* saat itu dapat dijalankan pada saat itu sesuai dengan *state* yang sedang berjalan pada *tick* tersebut.

4. Fungsi Seleksi

Memilih *action* dan *heading* terbaik berdasarkan strategi *greedy algorithm* yang memperhatikan *obstacle* beserta damage yang didapat dan game object yang berada di sekitar beserta *advantage* yang dapat dimanfaatkan dengan melakukan *action* terhadap game object tersebut.

5. Fungsi Kelayakan

Memeriksa *action* dan *heading* yang akan dipilih dapat dilakukan dan tidak meletakkan *bot* pada kerugian-kerugian atau posisi yang membahayakan.

6. Fungsi Obyektif

Bot dapat bertahan hingga *tick* terakhir dengan skor terbesar.

4.2. Alternatif Solusi Greedy

4.2.1. Greedy By Size

Ukuran bot atau size menjadi salah satu parameter yang dapat menentukan keberlangsungan hidup bot dan penilaian akhir bot. Strategi greedy yang didasarkan oleh size melakukan optimalisasi pengambilan *gameobject* terdekat yang dapat menambah ukuran bot. Strategi ini diimplementasikan dengan memprioritaskan object terdekat dengan jenis yang lebih memberikan banyak tambahan size. Prioritas utama object yang diambil adalah enemy yang ukurannya lebih kecil daripada bot. Enemy diutamakan karena size minimal dari sebuah *player* yang hidup adalah 5, yang mana jauh lebih besar dari makanan biasa yang dapat dikonsumsi. Kemudian, bot akan mengutamakan *superfood* yang dapat melipatgandakan kemampuan absorpsi makanan bot. Terakhir, bot akan memprioritaskan *food* yang berada pada lokasi terdekat.

Kompleksitas algoritma ini dipengaruhi oleh pemilihan target yang dituju sesuai dengan prioritas dan keberadaan target-target yang ada di sekitar bot. Apabila terdapat n target pada senarai objek pada prioritas tertentu, algoritma akan melakukan pengecekan terhadap setiap elemen dalam senarai hingga didapatkan target yang sesuai dengan kriteria yang diperlukan. Apabila tidak ada, maka akan berpindah ke prioritas selanjutnya, hingga prioritas terakhir. Maka, dapat disimpulkan kompleksitas algoritma ini adalah $O(n)$ dengan *bestcase* terjadi saat langsung ditemukan enemy terdekat yang ukurannya lebih kecil yakni $O(1)$.

Algoritma ini secara umum dapat mengimplementasikan salah satu fungsi objektif utama permainan yakni mendapatkan skor terbesar dengan mengonsumsi lawan dan makanan. Akan tetapi, algoritma ini masih belum dapat menjamin keselamatan bot untuk bertahan hingga akhir karena tidak mempertimbangkan pergerakan lawan dan obstacle yang ada disekitar bot. Oleh karena itu penting dalam penggunaan greedy by size ini dilakukan kombinasi dengan algoritma greedy berdasarkan paradigma lain.

4.2.2. Greedy by Powerups

Powerups atau peningkatan-peningkatan kemampuan didapatkan oleh bot dengan memanfaatkan game objects yang terdiri dari *torpedo salvo*, *supernova*, *teleport*, dan *afterburner*. Keempat *gameobject* ini memiliki fungsi dan implementasi yang berbeda sesuai dengan keadaan yang terjadi dan akan dicapai.

Torpedo salvo dapat diledakkan dengan ukuran 10 dan kecepatan 60 sesuai heading player. *Torpedo salvo* akan mengurangi ukurang enemy dan menambah ukuran bot sesuai dengan ukuran *torpedo salvo* saat mengenai enemy. *Salvo* akan ter-recharge setiap 10 tick. Oleh karena itu, strategi *greedy* yang dapat dilakukan adalah menembakkan *torpedo* ke arah musuh terdekat sesaat setelah *salvo recharged*. Algoritma ini memiliki kompleksitas O(1) karena hanya akan memperhatikan player terdekat untuk ditembak. Algoritma greedy ini juga terbilang efektif karena secara tidak langsung bertindak selayaknya pencurian size dari lawan terdekat dan tidak memiliki harga yang perlu dibayar oleh *bot* dari segi speed maupun size.

Supernova adalah bom yang hanya muncul sekali dalam permainan dan dapat diambil oleh player tercepat. *Supernova* dapat diledakkan pada arah tertentu, merusak dan membunuh objek-objek di radius penembakan, kemudian berubah menjadi *gas cloud*. Strategi *greedy* yang dapat dilakukan untuk memanfaatkan *supernova* adalah dengan secepat mungkin mengambil *supernova* kemudian menggunakananya untuk menembak sebanyak mungkin enemy yang bisa didapat dalam radius ledakan *supernova*. Algoritma greedy ini memiliki kompleksitas O(n) dengan n merupakan anggota senarai enemy yang ada di sekitar sebuah enemy karena harus menentukan target dengan jumlah enemy terbanyak yang dapat diledakkan. Algoritma ini cukup efektif namun kurang efisien karena besar resiko yang harus dibayar bot. Resiko ini berhubungan dengan keadaan yang menempatkan bot pada posisi sangat membahayakan terutama ketika melakukan *pickup*. Banyak kontra-algoritma yang dapat dibuat seperti strategi-strategi pembunuhan

baik dengan cara *direct-eating* maupun *mass-bombing* di area munculnya *supernova*.

Teleport adalah platform berpindah tempat yang dapat diluncurkan oleh player dengan cost 20 size setiap 100 tick. *Teleporter* akan meledak di ujung map dan secara konstan bergerak dengan kecepatan 20. Strategi *greedy* yang dapat dilakukan dalam penggunaan *teleporter* adalah dengan mengirimnya pada tempat yang perlu dituju sesegera mungkin seperti pengambilan *supernova* atau saat melarikan diri dari musuh. Algoritma *greedy* ini memakan kompleksitas O(1) karena langsung menuju ke arah target tertentu. Akan tetapi, strategi *greedy* ini kurang menguntungkan dibandingkan dengan biaya 20 size yang harus dikeluarkan saat pertama kali meluncurkan *teleporter*. Frekuensi munculnya yang rendah juga membuat objek ini kurang dapat diandalkan dalam setiap saat dia dibutuhkan.

Afterburner adalah kemampuan untuk meningkatkan kecepatan dengan koefisien 2 yang harus dibayar dengan harga 1 unit size bot. Strategi *greedy* yang dapat diimplementasikan dengan menggunakan *powerups* ini adalah dengan mengaktifkannya saat bot perlu bergerak cepat untuk mencapai tujuannya tanpa mengalami kerugian dalam pengejaran yang memerlukan kekuatan sebesar 1 unit size bot ini. Selain itu *greedy* yang dapat dilakukan adalah mengaktifkan kemampuan ini untuk kabur dari suatu musuh atau obstacle. Kompleksitas algoritma *greedy* dengan pemanfaatan *afterburner* adalah O(1) dengan langsung memperhatikan player atau obstacle yang sedang berada di radius terdekat. Pemanfaatan algoritma *greedy* berdasarkan *afterburner* cukup efektif untuk mempertahankan hidup dari musuh, mengurangi jumlah musuh, sekaligus mendapatkan poin sebanyak-banyaknya.

Dari keempat objek ini, dapat dipilih objek mana yang paling diperlukan sesuai dengan kondisi yang ada kemudian dibuat prioritas, yakni *afterburner*, *torpedo salvo*, *supernova*, dan *teleporter*. Secara umum algoritma *greedy* by *powerups* ini masih perlu dikombinasikan dengan algoritma *greedy* lain untuk dapat efektif.

4.2.3. Greedy by Attacking Other Player

Penyerangan terhadap *player* lain dalam permainan ini dapat dilakukan secara *direct* dengan memangsa langsung dengan *collision* maupun *indirect* dengan menembakkan *torpedo salvo* atau melakukan *bombing* dengan *supernova*. Algoritma *greedy* yang dapat dilakukan berdasarkan *attacking other player* adalah dengan mengambil supernova terlebih dahulu kemudian menganalisis jumlah maksimal player yang dapat

dibom dalam satu waktu peledakan, kemudian menembakkan *torpedo salvo* setiap ter-recharge pada player terdekat, serta menjadikan player terkecil terdekat menjadi target untuk dimakan secara *direct*.

Kompleksitas algoritma untuk menjalankan ini adalah $O(n)$ dengan n adalah jumlah analisis terhadap sebanyak n anggota senarai *playerList* untuk menentukan target, baik target bom, target tembakan, maupun target pemakanan. Algoritma greedy dengan *attacking other player* ini dapat dibilang menjadi algoritma yang paling efektif melihat dia menjawab dua fungsi objektif dari permainan ini secara bersamaan, yakni untuk menjadi yang paling akhir bertahan dan mendapatkan poin paling maksimal dengan memakan *player* lain. Namun pastinya agar lebih efektif, masih perlu dilakukan kombinasi dengan *greedy-greedy* dengan paradigma lain, mengingat penyerangan memiliki *cost* yang cukup besar, baik secara penembakan dan pengeboman maupun syarat memangsa yang harus menjadi lebih besar dari yang dimangsa.

4.2.4. Greedy by Defense

Defense adalah upaya bertahan dari serangan-serangan eksternal yang dapat mengancam keberadaan, benda. Ancaman-ancaman ini antara lain dapat berbentuk *obstacle* maupun keberadaan musuh terdekat yang lebih besar ukurannya sedang mengintai kita. *Obstacle damage* atau kerusakan yang disebabkan oleh tantangan-tantangan tertentu harus dihindari atau diminimalisir sekecil mungkin agar *spaceship* kita tidak mati dan bertahan sampai akhir. *Obstacle* yang dapat menyebabkan kerusakan antara lain terdiri dari *gas clouds*, *world bound*, dan *asteroid fields*. Serangan dari musuh seperti *torpedo*, *teleporter* yang diluncurkan, dan *supernova* juga termasuk *obstacle* yang perlu dihindari.

Fitur *defense* yang dapat dimanfaatkan dalam membentuk algoritma *greedy* antara lain mencakup *shield*, *afterburner*, dan pemindahan *heading* ke *direction* yang aman. Algoritma *greedy* yang digunakan untuk melakukan *defense* dilakukan dengan mengenali *gameobject* terdekat dan melakukan penghindaran sesuai dengan prioritas berdasarkan damage yang dapat diterima. Prioritas penghindaran secara terurut adalah *world bound*, *teleporter*, *supernova*, *gas cloud*, *enemy*, *torpedo*, dan *asteroid fields*. Berdasarkan dampak yang dapat dihasilkan, terurut dari pengurangan ukuran terbesar hingga kecepatan.

Implementasi penggunaan algoritma *greedy* ini memiliki algoritma $O(n)$ dengan n adalah jumlah *obstacle* yang perlu dihindari. Algoritma ini secara efektif dapat menjawab objektif permainan untuk bertahan menjadi player terakhir yang bertahan hidup. Algoritma ini juga perlu diikuti oleh

strategi penyerangan dan pengumpulan ukuran karena banyak *tools defense* menghabiskan size.

4.2.5. Greedy by Score

Scoring menjadi penentu akhir ketika terjadi draw atau terdapat 2 bot kapal terakhir yang mati secara bersamaan. Scoring memiliki aturan sebagai berikut :

- a) Memangsa kapal pemain lain = 10 poin
- b) Mengonsumsi makanan = 1 poin
- c) Melintasi wormhole = 1 poin

Algoritma *greedy by score* ini secara tidak langsung akan mengombinasikan algoritma-algoritma *greedy* yang sebelumnya dibahas. Untuk menjadi dua kapal terakhir yang bertahan, diperlukan prioritas *defense* sehingga menghindari *spaceship* hancur di awal permainan. Untuk menghindari terjadinya serangan, perlu dilakukan serangan awal pada musuh sehingga dalam kondisi yang lebih aman, *greedy by attacking other player* akan menjadi prioritas. Apabila tidak ada *possible nearest enemy* untuk diserang, maka *greedy by size* menjadi prioritas selanjutnya. *Greedy by powerups* dapat menjadi prioritas sekunder karena hanya dapat digunakan pada saat-saat tertentu.

Secara umum, implementasi algoritma ini memiliki kompleksitas $O(n)$ karena hanya akan mengubah dan menambah parameter kondisional pengambilan keputusan tanpa melakukan komputasi yang lebih pelik dibandingkan algoritma-algoritma yang lain. Algoritma ini sangat efektif karena *scoring* adalah objektif utama permainan, yakni untuk bertahan terakhir dan memiliki skor terbesar.

4.3. Analisis Solusi Greedy Terbaik

Dari seluruh algoritma greedy yang sudah dianalisis, algoritma *greedy by attacking other player* yang dikombinasikan dengan berbagai algoritma *greedy by size* dan *greedy by defense* menjadi pilihan yang akan diimplementasikan oleh kelompok kami.

Pertama-tama ditetapkan prioritas target menggunakan prioritas player terdekat, superfood terdekat, atau makanan terdekat terlebih dahulu dengan kondisi target sedang tidak dikejar oleh player lain dan target juga memiliki jarak lebih jauh dari obstacle terdekat. Kemudian, akan diperhatikan obstacle-obstacle yang dapat dihindari terlebih dahulu, apabila memungkinkan digunakan juga powerups untuk mempercepat penghindaran. Obstacle seperti supernova dan

teleporter hanya akan muncul di saat-saat tertentu saja, oleh karena itu implementasinya tergolong sekunder dan tidak akan diimplementasikan. Selain itu, adanya supernova akan menarik pemain lain untuk berlomba mengambilnya sehingga dapat menimbulkan kerumunan. Kerumunan yang terjadi akan mengundang player lain dalam peluncuran powerup karena dapat memberikan damage pada lebih banyak player. Oleh karena itu, supernova hendaknya tidak menjadi prioritas dalam penentuan target. Di sisi lain, gas cloud sangat perlu dihindari dengan berjalan ke target terdekat secepat mungkin memanfaatkan afterburner. Kemudian world bound menjadi prioritas selanjutnya untuk dihindari dengan mengubah arah player menjadi ke arah pusat menjauhi boundaries yang ada. Terakhir menghindari musuh yang jauh lebih besar juga perlu untuk dilakukan dengan mengubah arah menjadi searah tangan posisi musuh dan bot. Obstacle berupa serangan torpedo dari musuh lain dapat di-counter dengan melakukan attack torpedo balik. Adanya powerups shield tidak diimplementasikan dalam program dikarenakan adanya pengurangan ukuran bot yang dinilai kurang menguntungkan. Obstacle berupa asteroid fields tidak perlu dihindari karena sifatnya yang hanya mengurangi kecepatan tidak terlalu berpengaruh besar pada kondisi bot. Apabila semua obstacle telah berhasil dihindari atau tidak ada obstacle yang perlu dihindari, selanjutnya dapat dilakukan targeting pada target yang telah ditentukan di awal.

Di antara implementasi defense dan attack ini, apabila kondisi powerups berupa torpedo salvo dapat digunakan dan kondisi bot mungkin melakukan penembakan, maka akan langsung ditembakkan.

Dalam implementasi ini digunakan beberapa fungsi sebagai berikut:

1) computeNextPlayerAction

Fungsi ini menganalisis kemungkinan kondisi dan mengembalikan *heading* dan *action* player sesuai dengan kondisi.

2) setTarget

Fungsi ini menganalisis objek terdekat yang dapat dijadikan target berdasarkan pertimbangan-pertimbangan algoritma *greedy* di atas.

3) getHeadingBetween

Fungsi ini menganalisis arah antara dua objek dan mengembalikan arah objek yang akan dituju.

4) getHeadingAway

Fungsi ini menganalisis arah agar suatu objek menjauhi suatu objek tertentu berdasarkan analisis arah yang aman.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

Implementasi solusi greedy telah diimplementasikan pada laman berikut:

https://github.com/munzayanahusn/Tubes1_greed.axio.git

4.1. Pseudocode

KAMUS GLOBAL

```
bot : GameObject
playerAction : PlayerAction
gameState : GameState
```

```
procedure computeNextPlayerAction(playerAction : PlayerAction)
{Menggerakkan bot untuk melakukan Action paling optimal sesuai
Algoritma Greedy pada Game State saat ini}
```

KAMUS LOKAL

```
nearestSuperFoodList, nearestFoodList : List of GameObject
nearestPlayerList, nearestObstacleList : List of GameObject
Enemy, obstacle, target : GameObject
```

ALGORITMA

```
{Membuat List Game Objek}
nearestSuperFoodList <- getSuperFoodList()
nearestFoodList <- getFoodList()
nearestPlayerList <- getPlayerList()
nearestObstacleList <- getObstacleList()

{Menentukan Objek Fokus Aksi}
enemy <- nearestPlayerList.get(0)
obstacle <- nearestObstacleList.get(0)
target <- setTarget(nearestSuperFoodList, nearestFoodList,
                     nearestPlayerList)

{Masuki Gas Cloud untuk pertama kali, memulai Afterburner}
if isEnteringGasCloud() then
    playerAction.action <- StartAfterBurner
    playerAction.heading <- getHeadingBetween(bot, target)

{Masih di dalam Gas Cloud, tetap mengejar target}
else if isInsideGasCloud() then
    playerAction.action <- Forward
    playerAction.heading <- getHeadingBetween(bot, target)

{Afterburner tidak lagi diperlukan, menghentikan Afterburner}
else if isAfterburnerNotNeeded() then
    playerAction.action <- StopAfterBurner
    playerAction.heading <- getHeadingBetween(bot, target)
```

```

{Torpedo dapat ditembakkan, tembakkan ke musuh terdekat}
else if isTorpedoFireable() then
    playerAction.action <- FireTorpedoes
    playerAction.heading <- getHeadingBetween(bot, enemy)

{Mendekati batas dunia, bergerak ke tengah}
else if isApproachinBound() then
    playerAction.action <- Forward
    playerAction.heading <- goToCenter()

{Terdapat musuh yang lebih besar dari arah tengah,
berbelok 45 derajat}
if isBiggerEnemyApproachinFromCenter() then
    playerAction.heading <- playerAction.heading + 45

{Musuh yang lebih besar mendekat, bergerak menjauh}
else if isBiggerEnemyNearby() then
    playerAction.action <- Forward
    playerAction.heading <- getHeadingAway(bot, enemy)

{Kejar target awal}
else
    {Mengejar musuh yang jauh lebih kecil, memulai Afterburner}
    if (target = enemy) and isTargetALotSmaller() then
        playerAction.action <- StartAfterburner
    {Mengejar target}
    else
        playerAction.action <- Forward
    playerAction.heading <- getHeadingBetween(bot, target)

{Bergerak menuju Obstacle, ubah arah gerakan}
if isHeadingTowardsObstacle() then
    playerAction.heading <- playerAction.heading + 90

{Mengatur aksi player}
setPlayerAction(playerAction)

```

```

function getNearestPlayerList() -> List of GameObject
{Mengembalikan list Player terurut dari yang terdekat}

```

KAMUS LOKAL

-

ALGORITMA

```

-> gameState.getPlayerGameObjects().stream()
    .filter(item->item.id != bot.id)
    .sorted(Comparator.comparing(getDistanceBetween(bot, item)))
    .collect(Collectors.toList())

```

```
function getFoodList() -> List of GameObject
{Mengembalikan list Food terurut dari yang terdekat}
```

KAMUS LOKAL

-

ALGORITMA

```
-> gameState.getPlayerGameObjects().stream()
    .filter(item->item.getGameObjectType() = FOOD)
    .sorted(Comparator.comparing(getDistanceBetween(bot, item)))
    .collect(Collectors.toList())
```

```
function getSuperFoodList() -> List of GameObject
{Mengembalikan list Superfood terurut dari yang terdekat}
```

KAMUS LOKAL

-

ALGORITMA

```
-> gameState.getPlayerGameObjects().stream()
    .filter(item->item.getGameObjectType() = SUPERFOOD)
    .sorted(Comparator.comparing(getDistanceBetween(bot, item)))
    .collect(Collectors.toList())
```

```
function getObstacleList() -> List of GameObject
{Mengembalikan list Obstacle terurut dari yang terdekat}
```

KAMUS LOKAL

-

ALGORITMA

```
-> gameState.getPlayerGameObjects().stream()
    .filter(item->item.getGameObjectType() = GAS_CLOUD
        || item->item.getGameObjectType() = WORMHOLE
        || item->item.getGameObjectType() = ASTEROID_FIELD)
    .sorted(Comparator.comparing(getDistanceBetween(bot, item)))
    .collect(Collectors.toList())
```

```
function setTarget(superFoodList, foodList, playerList
                    : List of GameObject) -> GameObject
{Mengembalikan target paling optimal menggunakan Algoritma Greedy}
```

KAMUS LOKAL

```
enemyDistance : List of GameObject
retObject, temp, enemy, superFood, food : GameObject
i : integer
```

ALGORITMA

```
enemy <- playerList.get(0)
```

```
{Musuh lebih kecil daripada bot, target musuh}
```

```

if enemy.getSize() < bot.getSize() then
    retObject <- enemy

{Terdapat Superfood}
else if not superFoodList.isEmpty() then
    i <- 0
    retObject <- NULL

    {Bot merupakan player terdekat dari Superfood atau tidak ada
    player lain yang mengarah ke Superfood, target Superfood}
    while (retObject = NULL) and (i < superFoodList.size()) do
        superfood <- superFoodList.get(i)
        enemy <- getEnemyDistance(playerList, superFood).get(0)
        if getDistance(bot, superFood) < getDistance(enemy,
            superFood) or enemy.heading != getHeading(enemy,
            superfood) then
            retObject <- superFood
        else
            i <- i+1

{Terdapat food}
else if not foodList.isEmpty() then
    i <- 0
    retObject <- NULL

    {Bot merupakan player terdekat dari food atau tidak ada player
    lain yang mengarah ke food, target food}
    while (retObject = NULL) and (i < foodList.size()) do
        food <- foodList.get(i)
        enemy <- getEnemyDistance(playerList, superFood).get(0)
        if getDistance(bot, food) < getDistance(enemy, food) or
            enemy.heading != getHeading(enemy, food) then
            retObject <- food
        else
            i <- i+1

{target food terdekat sebagai basis}
if retObject = NULL then
    retObject <- foodList.get(0)

{Kembalikan retObject}
-> retObject

function getEnemyDistance(playerList : List of GameObject,
                           object : GameObject) -> List of GameObject
{Mengembalikan List GameObject Player yang diurutkan berdasarkan
jaraknya ke GameObject object}

```

KAMUS LOKAL

-

ALGORITMA

```
-> playerList.stream()
    .sorted(Comparator.comparing(item ->
        getDistanceBetween(item, object)))
    .collect(Collectors.toList())
```

function isEnteringGasCloud(bot : GameObject) -> boolean

{Mengembalikan True jika bot berada dalam Gas Cloud dan Afterburner mati}

KAMUS LOKAL

-

ALGORITMA

```
-> (bot.effectsCode = 4 or bot.effectsCode = 6)
```

function isInsideGasCloud(bot : GameObject) -> boolean

{Mengembalikan True jika bot berada dalam Gas Cloud dan Afterburner aktif}

KAMUS LOKAL

-

ALGORITMA

```
-> (bot.effectsCode = 5 or bot.effectsCode = 7)
```

function isAfterBurnerNotNeeded(bot, enemy : GameObject) -> boolean

{Mengembalikan True jika Afterburner aktif dan ukuran bot kurang dari ukuran musuh atau ukuran bot kurang dari 20}

KAMUS LOKAL

-

ALGORITMA

```
-> (bot.effectsCode%2 = 1 and (bot.getSize() <= enemy.getSize() + 5
or bot.getSize() <= 20))
```

function isTorpedoFireable(bot, enemy : GameObject,

obstacle : List of GameObject) -> boolean

{Mengembalikan True jika Torpedo Salvo tersedia, ukuran bot lebih dari 20, bot dan musuh tidak berada dalam obstacle, kecepatan musuh kurang dari kecepatan Torpedo, dan tidak ada obstacle dalam lintasan Torpedo}

KAMUS LOKAL

-

ALGORITMA

```
-> (bot.torpedoSalvoCount > 0 and bot.getSize() > 20 and
    bot.effectsCode <= 1 and enemy.effectsCode = 0 and
    enemy.speed <= 60 and !obstacleInTheWay(bot, enemy, obstacle))
```

```
function obstacleInTheWay(bot, enemy : GameObject,  
                           obstacle : List of GameObject) -> boolean  
{Mengembalikan True jika terdapat obstacle di antara bot dan enemy}
```

KAMUS LOKAL

headingTo, i : integer

ALGORITMA

```
headingTo <- getHeadingBetween(bot, enemy)  
i iterate [0..obstacle.size())  
  if (Math.abs(headingTo -  
         getHeadingBetween(bot, obstacle.get(i))) < 30  
    and getDistanceBetween(bot,  
                           obstacle.get(i)) < getDistanceBetween(bot, enemy)) then  
    -> true  
-> false
```

```
function isApproachingBound(bot : GameObject) -> boolean  
{Mengembalikan True jika bot sedang berada sangat dekat dengan  
world's bound}
```

KAMUS LOKAL

-

ALGORITMA

```
-> ((getDistanceFromCenter() + 2 * bot.getSize()) >  
gameState.world.getRadius())
```

```
function isEnemyApproachingFromCenter(bot, enemy : GameObject)  
                           -> boolean  
{Mengembalikan True jika terdapat enemy yang berada di antara pusat  
map dengan bot}
```

KAMUS LOKAL

-

ALGORITMA

```
->(enemy.getSize() > bot.getSize() and  
  (Math.abs(goToCenter() - getHeadingBetween(bot, enemy)) < 30))
```

```
function isBiggerEnemyNearby(bot, enemy : GameObject) -> boolean  
{Mengembalikan True jika musuh berukuran lebih besar dari bot dan  
musuh berada di dekat bot}
```

KAMUS LOKAL

-

ALGORITMA

```
->(enemy.getSize() >= bot.getSize() and getDistanceBetween(bot,  
enemy)  
  < enemy.getSize()*5 + bot.getSize()*2)
```

```
function isTargetALotSmaller(bot, target : GameObject) -> boolean
{Mengembalikan True jika ukuran target jauh lebih kecil daripada bot}
```

KAMUS LOKAL

-

ALGORITMA

```
-> (bot.effectsCode%2 = 0 and
    bot.getSize() - target.getSize() >
    getDistanceBetween(bot, target)/(bot.speed) + 20)
```

```
function isHeadingTowardsObstacle(bot, obstacle : GameObject,
                                    action: PlayerAction) -> boolean
{Mengembalikan True jika bot bergerak menuju obstacle}
```

KAMUS LOKAL

-

ALGORITMA

```
-> ((action.action = PlayerActions.Forward and
      Math.abs(action.heading -
              getHeadingBetween(bot, obstacle)) <= 30 and
              getDistanceBetween(bot, obstacle) <
              bot.getSize()*2 + obstacle.getSize()*2) and
      (! (bot.getSize() > obstacle.getSize()) and
          obstacle.getGameObjectType() =
          ObjectTypes.GAS_CLOUD)))
```

```
function getDistanceBetween(object1, object2 : GameObject) -> long
{Mengembalikan jarak antara dua objek}
```

KAMUS LOKAL

triangleX, triangleY : long

ALGORITMA

```
triangleX <- Math.abs(object1.getPosition().x -
object2.getPosition().x)
triangleY <- Math.abs(object1.getPosition().y -
object2.getPosition().y)
-> Math.sqrt(triangleX * triangleX + triangleY * triangleY)
```

```
function getDistanceFromCenter() -> long
{Mengembalikan jarak dari bot ke pusat}
```

KAMUS LOKAL

triangleX, triangleY : long

ALGORITMA

```
triangleX <- Math.abs(bot.getPosition().x -
gameState.world.getCenterPoint().x)
triangleY <- Math.abs(bot.getPosition().y -
```

```
gameState.world.getCenterPoint().y  
-> Math.sqrt(triangleX * triangleX + triangleY * triangleY)
```

```
function goToCenter() -> integer  
{Mengembalikan arah heading menuju ke pusat world map}
```

KAMUS LOKAL

```
worldCenter : GameObject  
function getHeadingBetween(object, otherObject: GameObject) ->  
integer
```

ALGORITMA

```
-> getHeadingBetween(bot, worldCenter)
```

```
function getHeadingBetween(object, otherObject: GameObject) ->  
integer
```

```
{Mengembalikan arah heading dari suatu object menuju object tertentu  
yang lain}
```

KAMUS LOKAL

```
direction : integer  
function toDegrees(v : long) -> integer
```

ALGORITMA

```
direction <- toDegrees(Math.atan2  
    (otherObject.getPosition().y - object.getPosition().y,  
     otherObject.getPosition().x - object.getPosition().x))  
-> (direction + 360) % 360
```

```
function getHeadingAway(bot, otherObject : GameObject) -> integer  
{Mengembalikan arah heading menjauhi sebuah game object yang harus  
dihindari}
```

KAMUS LOKAL

```
direction : integer  
function toDegrees(v : long) -> integer
```

ALGORITMA

```
direction <- toDegrees(Math.atan2  
    (otherObject.getPosition().y - object.getPosition().y,  
     otherObject.getPosition().x - object.getPosition().x))  
-> direction
```

```
function toDegrees(v : long) -> integer  
{Mengembalikan nilai perhitungan dalam satuan derajat}
```

KAMUS LOKAL

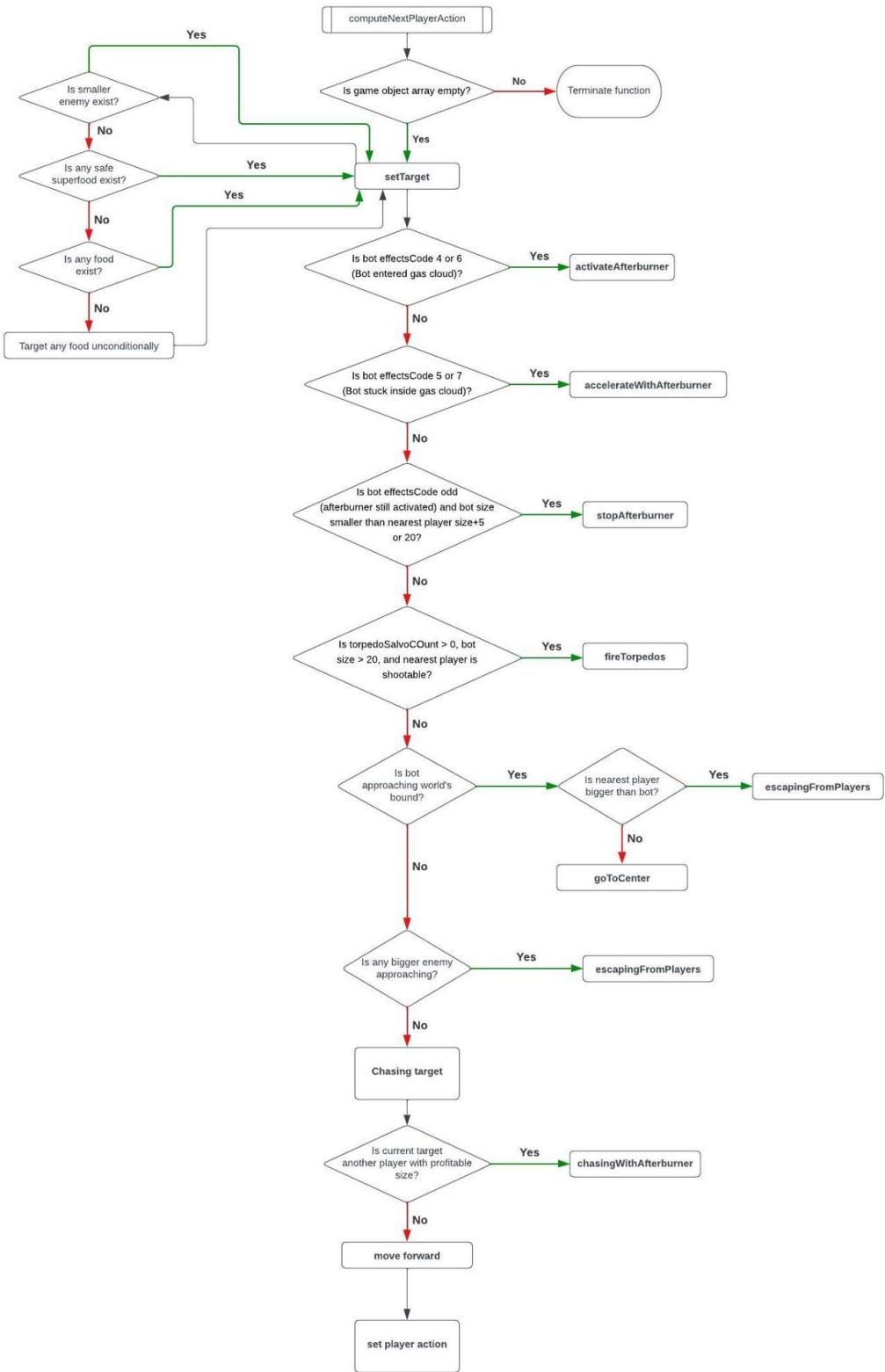
```
returnVal : integer
```

ALGORITMA

```
returnVal <- (v * (180/ Math.PI))
```

```
-> retVal
```

4.2. Flowchart



4.3. Struktur Data

Program implementasi algoritma greedy pada bot permainan Galaxio terdiri dari beberapa struktur data, antara lain:

1. Game State

Struktur data game state merupakan struktur data yang berisi informasi keseluruhan permainan, termasuk didalamnya inisiasi peta permainan, inisiasi setiap object yang ada pada peta, serta inisialisasi informasi bot kapal setiap pemain. Selain itu, terdapat implementasi method-method yang berkaitan dengan informasi objek permainan.

2. Game Object

Struktur data game object merupakan struktur data yang berisi informasi lebih detail setiap objek pada peta. Informasi tersebut terdiri dari ID, ukuran objek (size), kecepatan objek (speed), arah pergerakan objek (currentHeading), posisi objek pada peta (position), tipe objek (gameObjectType), banyaknya amunisi torpedo salvo yang dimiliki (torpedoSalvoCount), status keberadaan supernova (supernovaAvailable), banyaknya amunisi teleporter yang dimiliki (teleporterCount), serta banyaknya shield yang dimiliki (shieldCount). Tipe objek pada peta dapat berupa PLAYER, FOOD, WORMHOLE, GAS_CLOUD, ASTEROID_FIELD, TORPEDO_SALVO, SUPERFOOD, SUPERNOVA_PICKUP, SUPERNOVA_BOMB, TELEPORT, dan SHIELD. Selain itu, diimplementasikan pula method-method yang dapat melakukan aksi terhadap komponen dalam GameObject.

3. PlayerAction

Struktur data Player Action merupakan struktur data yang berisi informasi terkait masing-masing player. Informasi tersebut terdiri dari playerId, aksi yang dilakukan bot (action), dan arah gerak bot (heading). Aksi yang dapat dilakukan oleh bot antara lain bergerak maju (Forward), berhenti (Stop), menyalakan afterburner (StartAfterburner), berhenti menggunakan afterburner (StopAfterburner), meluncurkan torpedo (FireTorpedoes), meluncurkan supernova (FireSupernova), meledakkan supernova (DetonateSupernova), meluncurkan teleporter (FireTeleporter), melakukan teleportasi (Teleport), dan mengaktifkan *shield* (ActivateShield). Selain itu, diimplementasikan pula method-method untuk melakukan aksi terhadap setiap komponen dalam PlayerAction.

4. Position

Struktur data Position merupakan struktur data yang berisi informasi koordinat posisi x dan y dalam sistem koordinat kartesian. Struktur data ini berkaitan dengan representasi map yang menggunakan sistem koordinat kartesian dengan pusat peta berada pada koordinat (0,0). Selain itu, diimplementasikan pula method-method untuk melakukan aksi terhadap komponen dalam struktur Position.

5. World

Struktur data World merupakan struktur data yang berisi informasi dasar terkait peta serta ronde permainan. Informasi tersebut terdiri dari informasi posisi pusat lingkaran (centerPoint), jari-jari atau radius peta pada suatu saat (radius), serta ronde yang sedang berjalan (currentTick). Pada permainan ini, peta berbentuk lingkaran yang direpresentasikan menggunakan sistem koordinat kartesian dengan pusat peta berada pada koordinat (0,0) dan memiliki radius yang terus mengecil seiring berjalannya permainan. Selain itu, diimplementasikan pula method-method untuk melakukan aksi terhadap komponen dalam World.

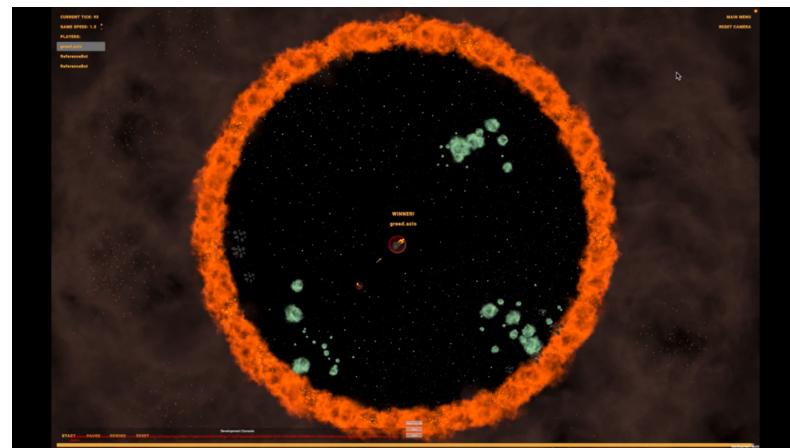
6. BotServices

Struktur data BotServices merupakan struktur data yang berisi method-method untuk mengendalikan kapal dalam permainan. Dalam struktur data ini pula diimplementasikan strategi untuk menentukan aksi bot setiap tick dengan algoritma greedy.

4.4. Pengujian

Sebagai bahan pengujian, penulis mempertandingkan bot greed.axio dengan reference bot yang tersedia pada starter-pack permainan Galaxio. Pengujian dilakukan dengan mempertandingkan bot greed.axio dengan dua buah reference bot sebanyak 10 permainan. Persentase kemenangan bot greed.axio pada pengujian adalah sebesar 100%. Sesuai dengan program yang dibangun, bot greed.axio akan memprioritaskan skema greedy by attacking other player, greedy by size, dan greedy by defense. Dengan strategi tersebut, bot greed.axio mampu menyelesaikan permainan uji dengan durasi rata-rata 150 tick.

Hasil pertandingan yang bervariatif tersebut terjadi karena map posisi launch obstacle pada map yang bervariatif setiap permainan. Kondisi terbaik bot greed.axio terjadi ketika mayoritas obstacle muncul di daerah sekitar batas boundaries peta. Hal ini dikarenakan penembakan torpedo yang menjadi prioritas utama dalam bot greed.axio tidak akan hancur karena menabrak obstacle yang menghalanginya. Sedangkan kondisi terburuk terjadi ketika mayoritas obstacle muncul diantara player atau tersebar di tengah peta karena torpedo yang ditembakkan bot menuju player lawan akan hancur terlebih dahulu akibat menabrak obstacle. Dari pengujian tersebut didapatkan bahwa strategi yang penulis kembangkan berhasil diimplementasikan pada bot greed.axio secara optimum untuk mencapai tujuan permainan tersebut.



Gambar 4.4. Contoh Hasil Pengujian

BAB 5

PENUTUP

5.1. Kesimpulan

Penentuan action player pada permainan Galaxio dilakukan setiap tick, sehingga strategi penentuan action cocok diimplementasikan dengan dengan konsep Algoritma Greedy, yaitu dengan menentukan solusi optimum lokal setiap langkahnya. Bot yang penulis kembangkan pada [Jaman berikut](#) mengutamakan greedy by attacking other players, greedy by size, dan greedy by defense dimana bot memprioritaskan penembakan torpedo menuju lawan, pengejaran lawan dengan ukuran lebih kecil, serta pengambilan superfood maupun food. Dengan strategi tersebut bot kapal pada permainan Galaxio dapat berjalan cukup optimal dengan persentase kemenangan 100% melawan bot referensi yang tersedia pada starter-pack.

5.2. Saran

Dalam pengembangan bot kapal pada permainan Galaxio, penulis menyarankan agar bot dapat mempertimbangkan lebih banyak kondisi dalam pemilihan keputusan aksi yang akan dilakukan. Pemilihan keputusan aksi tidak hanya berfokus untuk melakukan pertahanan dan penyerangan, namun juga dapat lebih efektif bila dapat dibuat suatu algoritma untuk melakukan penyerangan balik atau *counter* baik terhadap serangan maupun obstacle lain. Penulis menyarankan untuk mengutamakan menjauhi kerumunan, dikarenakan hal tersebut mengundang player lain untuk menggunakan *powerups* agar dapat memberi damage pada banyak pemain sekaligus.

5.3. Refleksi

Melalui tugas besar ini, penulis dapat banyak mengeksplorasi berbagai hal terkait algoritma greedy serta penerapannya. Selain penentuan strategi algoritma yang paling optimum, penulis juga banyak mengeksplorasi penerapan algoritma greedy dalam program komputer, terutama dengan Bahasa Pemrograman Java. Diperlukannya waktu yang relatif lebih lama dalam pemahaman struktur dasar program permainan *Galaxio* mengajarkan penulis dalam pentingnya membagi waktu dengan baik serta tidak menunda pekerjaan. Melalui tugas besar ini pula penulis belajar untuk bekerja sama dalam tim, peduli satu sama lain, dan saling membantu.

DAFTAR PUSTAKA

GeeksforGeeks. (Februari 2023). *Greedy Algorithms*. Diakses pada 14 Februari 2023

pukul 18.45, dari <https://www.geeksforgeeks.org/greedy-algorithms/>

Greedy Algorithms | Brilliant Math & Science Wiki. (n.d.). Diakses pada 14 February

2023 pukul 19.03, dari <https://brilliant.org/wiki/greedy-algorithm/>

Munir, Rinaldi. (2021). *Algoritma Greedy Bagian 1*. [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf)

LAMPIRAN

Repository Github : github.com/munzayanahusn/Tubes1_greed.axio

Video YouTube : youtu.be/riFXiDZj0CM