

LAPORAN TUGAS BESAR 1
IF2211 STRATEGI ALGORITMA
“PEMANFAATAN ALGORITMA GREEDY DALAM APLIKASI
PERMAINAN GALAXIO”



Kelompok TheLegend27

Brian Kheng 13521049

Michael Utama 13521137

Johann Christian Kandani 13521138

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

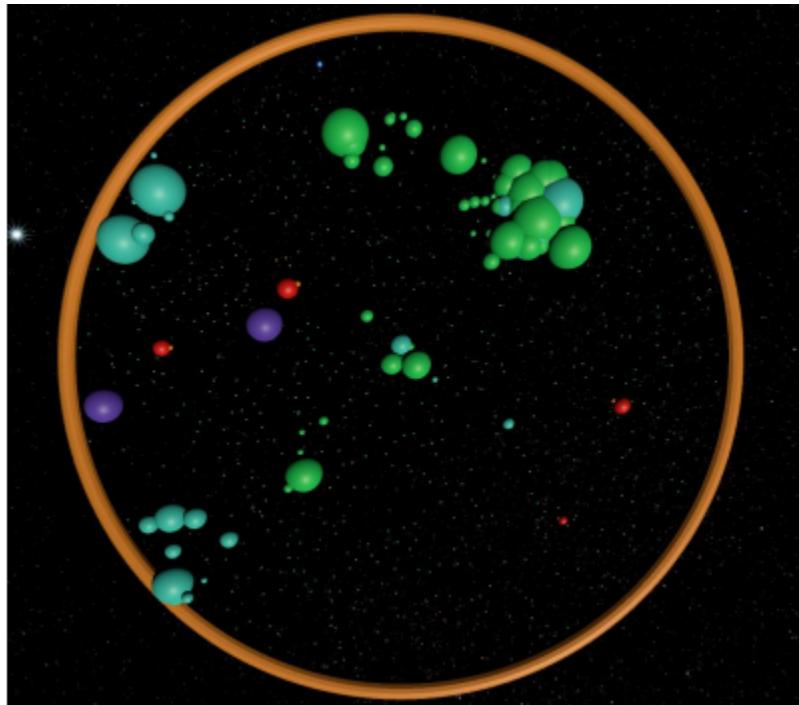
DAFTAR ISI	2
BAB 1	4
BAB 2	9
2.1. Algoritma Greedy	9
2.2. Sudut Antara Garis Pusat ke Pusat Lingkaran dan Garis Singgung Lingkaran	10
2.3. Cara Kerja Program	11
BAB 3	14
3.1. Mapping Persoalan Galaxio Menjadi Elemen-Elemen Greedy	14
3.2. Eksplorasi Alternatif Solusi Greedy yang Mungkin	15
3.3. Analisis Efisiensi dan Efektivitas dari Alternatif Solusi Greedy yang Dirumuskan dan Strategi Terbaik	16
BAB 4	18
4.1. Implementasi	18
4.1.1. Implementasi metode computeNextPlayerAction pada kelas BotService	18
4.1.2. Implementasi kelas PlayerActionValuesList	18
4.1.3. Implementasi kelas PlayerActionValues	21
4.2. Struktur Data	24
4.2.1. Modul Models	24
4.2.2. Modul Enums	25
4.2.3. Modul Services	26
4.2.4. Kelas Main	26
4.3. Analisis Pengujian	26
4.3.1. Pengujian ke-1	27
4.3.2. Pengujian ke-2	28
4.3.3. Pengujian ke-3	29
IF2211 Strategi Algoritma - Tugas Besar 1	2

4.3.4. Pengujian ke-4	30
BAB 5	31
5.1. Kesimpulan	31
5.2. Saran	31
Repository Github	32
Daftar Pustaka	32

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 dan tujuan dari permainan adalah agar bot kapal anda yang tetap hidup hingga akhir permainan. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah. Agar dapat memenangkan pertandingan, setiap bot harus mengimplementasikan strategi tertentu untuk dapat memenangkan permainan.



Gambar 1.1. Ilustrasi Permainan Galaxio

Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah game engine yang mengimplementasikan permainan Galaxio. Game engine dapat diperoleh pada laman berikut:

<https://github.com/EntelectChallenge/2021-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 di dalam starter-pack pada laman berikut ini:

<https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2>

Spesifikasi permainan yang digunakan pada tugas besar ini disesuaikan dengan spesifikasi yang disediakan oleh game engine Galaxio pada tautan di atas. Beberapa aturan umum adalah sebagai berikut.

1. Peta permainan berbentuk kartesius yang memiliki arah positif dan negatif. Peta hanya menangani angka bulat. Kapal hanya bisa berada di integer x,y yang ada di peta. Pusat peta adalah 0,0 dan ujung dari peta merupakan radius. Jumlah ronde maximum pada game sama dengan ukuran radius. Pada peta, akan terdapat 5 objek, yaitu Players, Food, Wormholes, Gas Clouds, Asteroid Fields. Ukuran peta akan mengecil seiring batasan peta mengecil.
2. Kecepatan kapal dilambangkan dengan x . Kecepatan kapal akan dimulai dengan kecepatan 20 dan berkurang setiap ukuran kapal bertambah. Ukuran (radius) kapal akan dimulai dengan ukuran 10. Heading dari kapal dapat bergerak antar 0 hingga 359 derajat. Efek afterburner akan meningkatkan kecepatan kapal dengan faktor 2, tetapi mengecilkan ukuran kapal sebanyak 1 setiap tick. Kemudian kapal akan menerima 1 salvo charge setiap 10 tick. Setiap kapal hanya dapat menampung 5 salvo charge. Penembakan slavo torpedo (ukuran 10) mengurangkan ukuran kapal sebanyak 5.
3. Setiap objek pada lintasan punya koordinat x,y dan radius yang mendefinisikan ukuran dan bentuknya. Food akan disebarluaskan pada peta dengan ukuran 3 dan dapat dikonsumsi oleh kapal player. Apabila player mengkonsumsi Food, maka Player akan bertambah ukuran yang sama dengan Food. Food memiliki peluang untuk berubah menjadi Super Food. Apabila Super Food dikonsumsi maka setiap makan Food, efeknya akan 2 kali dari Food yang dikonsumsi. Efek dari Super Food bertahan selama 5 tick.

4. Wormhole ada secara berpasangan dan memperbolehkan kapal dari player untuk memasukinya dan keluar di pasangan satu lagi. Wormhole akan bertambah besar setiap tick game hingga ukuran maximum. Ketika Wormhole dilewati, maka wormhole akan mengecil sebanyak setengah dari ukuran kapal yang melewatinya dengan syarat wormhole lebih besar dari kapal player.
5. Gas Clouds akan tersebar pada peta. Kapal dapat melewati gas cloud. Setiap kapal bertabrakan dengan gas cloud, ukuran dari kapal akan mengecil 1 setiap tick game. Saat kapal tidak lagi bertabrakan dengan gas cloud, maka efek pengurangan akan hilang.
6. Torpedo Salvo akan muncul pada peta yang berasal dari kapal lain. Torpedo Salvo berjalan dalam lintasan lurus dan dapat menghancurkan semua objek yang berada pada lintasannya. Torpedo Salvo dapat mengurangi ukuran kapal yang ditabraknya. Torpedo Salvo akan mengecil apabila bertabrakan dengan objek lain sebanyak ukuran yang dimiliki dari objek yang ditabraknya.
7. Supernova merupakan senjata yang hanya muncul satu kali pada permainan di antara quarter pertama dan quarter terakhir. Senjata ini tidak akan bertabrakan dengan objek lain pada lintasannya. Player yang menembakkannya dapat meledakannya dan memberi damage ke player yang berada dalam zona. Area ledakan akan berubah menjadi gas cloud.
8. Player dapat meluncurkan teleporter pada suatu arah di peta. Teleporter tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun. Player tersebut dapat berpindah ke tempat teleporter tersebut. Harga setiap peluncuran teleporter adalah 20. Setiap 100 tick player akan mendapatkan 1 teleporter dengan jumlah maximum adalah 10.
9. Ketika kapal player bertabrakan dengan kapal lain, maka kapal yang lebih besar akan dikonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran maximum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.

10. Terdapat beberapa command yang dapat dilakukan oleh player. Setiap tick, player hanya dapat memberikan satu command. Berikut jenis-jenis dari command yang ada dalam permainan:

- a. FORWARD
- b. STOP
- c. START_AFTERCLOUD
- d. STOP_AFTERCLOUD
- e. FIRE_TORPEDOES
- f. FIRE_SUPERNOVA
- g. DETONATE_SUPERNOVA
- h. FIRE_TELEPORTER
- i. TELEPORTUSE_SHIELD

11. Setiap player akan memiliki score yang hanya dapat dilihat jika permainan berakhir. Score ini digunakan saat kasus tie breaking (semua kapal mati). Jika mengonsumsi kapal player lain, maka score bertambah 10, jika mengonsumsi food atau melewati wormhole, maka score bertambah 1. Pemenang permainan adalah kapal yang bertahan paling terakhir dan apabila tie breaker maka pemenang adalah kapal dengan score tertinggi.

Adapun peraturan yang lebih lengkap dari permainan Galaxio, dapat dilihat pada laman :

<https://github.com/EntelectChallenge/2021-Galaxio/blob/develop/game-engine/game-rules.md>

Spesifikasi tugas:

Pada tugas besar kali ini, anda diminta untuk membuat sebuah bot untuk bermain permainan Galaxio yang telah dijelaskan sebelumnya. Untuk memulai, anda dapat mengikuti panduan singkat sebagai berikut.

1. Download latest release starter pack.zip dari tautan berikut
<https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2>
2. Untuk menjalankan permainan, kalian butuh beberapa requirement dasar sebagai berikut.
 - a. Java (minimal Java 11):
<https://www.oracle.com/java/technologies/downloads/#java>
 - b. IntelliJ IDEA: <https://www.jetbrains.com/idea/>
 - c. NodeJS: <https://nodejs.org/en/download/>
 - d. .Net Core 3.1: <https://dotnet.microsoft.com/en-us/download/dotnet/3.1.3>

Adapun panduan mengenai cara menjalankan permainan, membuat bot, build src code, dan melihat visualizer bisa dicek melalui tautan berikut:

https://docs.google.com/document/d/1Ym2KomFPLIG_KAbm3A0bnhw4_XQAsOKzpTa70IgnLNU/edit#

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 Tubes 1 (TBD).

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

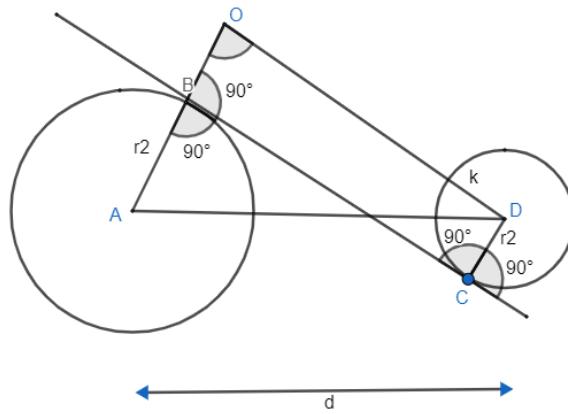
2.1. Algoritma Greedy

Algoritma Greedy merupakan sebuah metode dalam menyelesaikan masalah optimasi yang dilakukan secara bertahap (*step by step*) dengan tujuan untuk memperoleh hasil terbaik yang memenuhi kriteria yang telah ditentukan. Metode ini melibatkan pencarian himpunan bagian kandidat yang harus memenuhi kriteria solusi dan optimasi dari suatu objek. Tujuan dari Algoritma Greedy adalah untuk mencari solusi yang mendekati nilai optimal dengan memaksimumkan atau meminimumkan hasil penyelesaian yang didapat.

Salah satu aplikasi dari Algoritma Greedy adalah dalam pencarian lintasan terpendek untuk menuju suatu tempat tujuan. Dengan menggunakan metode ini, kita dapat menentukan kemungkinan jalan tersingkat untuk sampai ke tujuan yang diinginkan. Namun, perlu diingat bahwa Algoritma Greedy memiliki prinsip "*take what you can get now!*", yang berarti memilih opsi terbaik pada saat itu tanpa mempertimbangkan konsekuensi ke depan. Oleh karena itu, perlu dilakukan evaluasi secara teliti terhadap pilihan yang telah diambil, agar tidak terjebak dalam solusi suboptimal.

Proses Algoritma Greedy melibatkan beberapa tahapan dalam menyelesaikan masalah. Tahap pertama adalah mengambil pilihan terbaik pada saat itu tanpa memperhatikan konsekuensi ke depan. Kemudian, pada tahap kedua, diharapkan bahwa memilih opsi lokal optimum pada setiap langkah akan menghasilkan opsi global optimum. Terakhir, pada tahap ketiga, dilakukan langkah optimal lokal dengan harapan langkah yang diambil berikutnya akan membawa pada solusi global optimum. Dengan demikian, meskipun Algoritma Greedy memiliki keterbatasan, namun jika digunakan dengan benar dan bijak, metode ini dapat memberikan hasil yang optimal dalam menyelesaikan masalah optimasi.

2.2. Sudut Antara Garis Pusat ke Pusat Lingkaran dan Garis Singgung Lingkaran



Gambar 2.1. Garis singgung dua lingkaran (sumber:

[https://www.vedantu.com/question-answer/find-the-length-of-the-transverse-common-tangent-cl
ass-10-maths-cbse-5f08679b5071a97ba17b492b\)](https://www.vedantu.com/question-answer/find-the-length-of-the-transverse-common-tangent-cl)

Dua lingkaran yang pusatnya berjarak d yang masing-masing memiliki jari-jari r_1 dan r_2 ($d > r_1 + r_2$), maka sudut α yang dibentuk antara garis yang menghubungkan kedua pusat lingkaran (AD) dan garis singgung kedua lingkaran yang diproyeksikan ke pusat salah satu lingkaran (OD) memenuhi persamaan:

$$\sin(\alpha) = \frac{r_1 + r_2}{d}$$

Sehingga sudut α dapat diperoleh dengan persamaan

$$\alpha = \arcsin\left(\frac{r_1 + r_2}{d}\right)$$

2.3. Cara Kerja Program

Penjelasan mengenai cara kerja program game Galaxio dapat diuraikan sebagai berikut:

1. Runner saat dijalankan akan meng-host sebuah match pada sebuah hostname tertentu. Untuk koneksi lokal, runner akan meng-host pada localhost:5000.
2. Engine kemudian dijalankan untuk melakukan koneksi dengan runner. Setelah terkoneksi, Engine akan menunggu sampai bot-bot pemain terkoneksi ke runner.
3. Logger juga melakukan hal yang sama, yaitu melakukan koneksi dengan runner.
4. Pada titik ini, dibutuhkan beberapa bot untuk melakukan koneksi dengan runner agar match dapat dimulai. Jumlah bot dalam satu pertandingan didefinisikan pada atribut BotCount yang dimiliki file JSON "appsettings.json". File tersebut terdapat di dalam folder "runner-publish" dan "engine-publish".
5. Permainan akan dimulai saat jumlah bot yang terkoneksi sudah sesuai dengan konfigurasi.
6. Bot yang terkoneksi akan mendengarkan event-event dari runner. Salah satu event yang paling penting adalah ReceiveGameState karena memberikan status game.
7. Bot juga mengirim event kepada runner yang berisi aksi bot.
8. Permainan akan berlangsung sampai selesai. Setelah selesai, akan terbuat dua file json yang berisi kronologi match.

Penjelasan mengenai cara menjalankan program game Galaxio secara lokal di Windows dapat diuraikan sebagai berikut:

1. Lakukan konfigurasi jumlah bot yang ingin dimainkan pada file JSON "appsettings.json" dalam folder "runner-publish" dan "engine-publish"
2. Buka terminal baru pada folder runner-publish.

3. Jalankan runner menggunakan perintah “dotnet GameRunner.dll”
4. Buka terminal baru pada folder engine-publish
5. Jalankan engine menggunakan perintah “dotnet Engine.dll”
6. Buka terminal baru pada folder logger-publish
7. Jalankan engine menggunakan perintah “dotnet Logger.dll”
8. Jalankan seluruh bot yang ingin dimainkan
9. Setelah permainan selesai, riwayat permainan akan tersimpan pada 2 file JSON “GameStateLog_{Timestamp}” dalam folder “logger-publish”. Kedua file tersebut diantaranya GameComplete (hasil akhir dari permainan) dan proses dalam permainan tersebut.

Starter pack menyediakan starter bot sebagai titik awal dari pengembangan bot milik kita sendiri. Starter bot sudah dibekali kode yang dibutuhkan oleh bot untuk melakukan koneksi dengan runner sehingga kita dapat fokus kepada strategi permainan. Pada tugas besar ini, bot dituliskan dalam bahasa Java (JavaBot). Pengubahan nama bot dapat dilakukan dengan cara mengubah string “Coffee Bot” pada file Main.java baris ke-61. Baris 73 - 77 kode Main.java menunjukkan bahwa bot mengirim aksi ke runner menggunakan metode “send” dengan argumen aksi bot. Aksi ini dihitung dan diperbarui oleh bot menggunakan metode computeNextPlayerAction milik kelas BotService. Algoritma greedy diimplementasikan di dalam kelas BotService. Untuk dapat menjadikan kode menjadi bot yang dapat dipakai, source code harus di-build menjadi sebuah file jar terlebih dahulu. Setelah beberapa saat, maka folder target akan terbentuk yang berisi executable file jar. Kemudian, untuk menggunakan file jar tersebut ke permainan, dapat dijalankan script java -jar path.

Penjelasan mengenai cara menjalankan program game Galaxio melalui visualizer dapat diuraikan sebagai berikut:

1. Lakukan ekstrak pada file zip Galaxio dalam folder “visualiser” sesuai dengan OS yang digunakan.
2. Jalankan aplikasi Galaxio.
3. Buka menu “Options”.
4. Salin path folder “logger-publish” kalian pada “Log Files Location”, lalu “Save”.
5. Buka menu “Load”.
6. Pilih file JSON yang ingin di load pada “Game Log”, lalu “Start”.
7. Setelah masuk ke visualisasinya, kalian dapat melakukan start, pause, rewind, dan reset.

BAB 3

Aplikasi Strategi Greedy dalam Aplikasi Permainan Galaxio

3.1. Mapping Persoalan Galaxio Menjadi Elemen-Elemen Greedy

Aplikasi strategi *greedy* yang digunakan berupa pemilihan aksi berdasarkan nilai keuntungan (sebagai *heuristic value*) yang dihasilkan. Dengan metode yang didasari nilai *heuristic value*, implementasi *greedy* menjadi lebih fleksibel dan memudahkan dalam memodifikasi prioritas pemilihan aksi sebagai alternatif lain.

1. Himpunan kandidat: Aksi-aksi yang dapat diambil oleh pemain dalam suatu *tick*:
 $\{FORWARD, STOP, START_AFTERBURNER, STOP_AFTERBURNER,$
 $FIRE_TORPEDOES, FIRE_SUPERNOVA, DETONATE_SUPERNOVA,$
 $FIRE_TELEPORTER, TELEPORT, USE_SHIELD\}$
2. Himpunan solusi: Aksi yang memberikan keuntungan terbesar pada *tick* saat ini.
3. Fungsi solusi: Memeriksa pesan ‘permainan selesai’ dari *game engine*.
4. Fungsi seleksi (selection function): Pilih aksi yang memberikan keuntungan maksimum pada saat ini (pada suatu *tick* tertentu). Keuntungan dikategorikan sebagai menambah ukuran kapal, mengonsumsi kapal lain, dan menghindari hal-hal yang mengakibatkan ukuran kapal berkurang. Fungsi perhitungan nilai keuntungan bergantung pada masing-masing aksi.
5. Fungsi kelayakan (feasible): Jika aksi yang dilakukan menyebabkan kapal menjadi terkonsumsi/hancur, Hapus aksi dari kandidat aksi yang layak
 - a. FORWARD / TELEPORT pada suatu destinasi tidak akan dipilih jika:
 - i. Destinasi FORWARD / TELEPORT pada tick ini terdapat kapal lain yang ukurannya $2 \times$ ukuran kapal pemain. (tidak memerhatikan atau memprediksi gerak kapal tersebut)
 - b. FIRE_TORPEDOES tidak akan dipilih jika:

- i. Target memiliki efek SHIELD yang aktif
 - ii. Ukuran kapal ≤ 25
 - iii. Jumlah *torpedo salvo* yang dimiliki 0
- c. *USE_SHIELD* tidak akan dipilih jika:
- i. Ukuran kapal ≤ 50
 - ii. Jumlah shield yang dimiliki 0 (atau efek shield sedang aktif)
- d. *FIRE_TELEPORTER* tidak akan dipilih jika:
- i. Ukuran kapal ≤ 45
 - ii. Jumlah teleporter yang dimiliki 0
6. Fungsi objektif: Memaksimumkan ukuran kapal **dan** menjadi kapal terakhir yang bertahan.

3.2. Eksplorasi Alternatif Solusi Greedy yang Mungkin

Alternatif solusi *greedy* yang telah dieksplorasi adalah dengan memvariasikan fungsi *heuristic value* masing-masing aksi, sehingga mengubah prioritas-prioritas masing-masing aksi.

1. Greedy agresif
 - a. Prioritas FIRE_TORPEDOES

Menggunakan fungsi heuristik yang menghasilkan FIRE_TORPEDOES sebagai prioritas tertinggi untuk mengeliminasi kapal-kapal lain lebih awal.
2. Greedy defensif
 - a. Prioritas USE_SHIELD

Menggunakan fungsi heuristik yang mengutamakan (atau hanya menggunakan) shield sebagai pelindung dari torpedo.

b. Prioritas FIRE_TORPEDOES

Menggunakan fungsi heuristik yang mengutamakan menembak *torpedo salvo* ke arah torpedo yang akan menabrak kapal.

3. Greedy akumulasi

Menggunakan fungsi heuristik yang menghasilkan akumulasi ukuran sebagai prioritas tertinggi, dan mengurangi prioritas aksi-aksi yang menyebabkan ukuran kapal berkurang. Strategi termasuk memprioritaskan menghindari hal-hal seperti gas cloud. (mengimplikasikan strategi yang mengutamakan navigasi kapal)

4. Greedy poin

Menggunakan fungsi heuristik yang mengutamakan akumulasi poin.

3.3. Analisis Efisiensi dan Efektivitas dari Alternatif Solusi Greedy yang Dirumuskan dan Strategi Terbaik

Dari kumpulan alternatif strategi *greedy* yang dirumuskan, semua alternatif memiliki kompleksitas yang sama, akibat metode implementasi yang hanya memiliki perbedaan dalam fungsi heuristik. Fungsi heuristik hanya memiliki perbedaan formula matematis yang secara algoritmik tidak mengubah kompleksitas, namun dapat menambah beban komputasi berdasarkan fungsi matematis yang digunakan. Menghilangkan modul-modul dalam menghitung fungsi heuristik akan menambah efisiensi program, namun modul-modul aksi yang relatif jarang digunakan dalam strategi yang diimplementasikan (terkait *afterburner* dan *supernova*) juga membutuhkan komputasi yang relatif sedikit.

Berdasarkan simulasi-simulasi yang dijalankan, alternatif *greedy* yang terbaik adalah **Greedy akumulasi**. Berdasarkan analisis, hipotesis keuntungan terbesar dari akumulasi adalah ukuran merupakan sumber daya utama dalam menjalankan aksi-aksi dan merupakan penentu keberlangsungan kapal. Selain itu, ukuran yang besar juga memungkinkan kapal mengonsumsi kapal lain, dan mengurangi kemungkinan kapal dikonsumsi oleh kapal lain. Kerugian kecepatan dari kapal berukuran besar dapat diatasi dengan *teleporter* dan *shield* sebagai bentuk agresi maupun pertahanan terhadap kapal lain.

Implementasi *afterburner* relatif sulit diimplementasikan dengan algoritma *greedy* yang hanya mengevaluasi keadaan pada suatu saat saja. Selain itu, penggunaan *afterburner* sebagai bagian dari strategi bertentangan dengan strategi *greedy* akumulasi karena mengurangi ukuran kapal. Implementasi *supernova* juga tidak direalisasikan akibat sifat *supernova* yang tidak stabil terhadap *game engine*. (menyebabkan *game engine crash* ketika *supernova* yang ditembakkan keluar dari map sebelum diaktifkan.) Implementasi *supernova* juga menambah beban komputasi untuk fitur yang relatif jarang aktif dalam permainan, sehingga diputuskan untuk tidak diimplementasikan (meskipun implementasi relatif murah secara komputasi).

BAB 4

Implementasi dan Pengujian

4.1. Implementasi

4.1.1. Implementasi metode *computeNextPlayerAction* pada kelas *BotService*

```
Procedure computeNextPlayerAction (input/output playerAction : PlayerAction)
{ Menjalankan algoritma greedy untuk menentukan aksi player berikutnya }
```

Kamus Lokal

valuesList : PlayerActionValuesList

Algoritma

```
valuesList ← PlayerActionValuesList(playerAction, gamestate, bot, teleporter)
playerAction ← valuesList.bestAction()
```

If(playerAction.getAction() = FIRETELEPORT) then

 teleporter.update()

If(playerAction.getAction() = TELEPORT) then

 teleporter.reset()

4.1.2. Implementasi kelas *PlayerActionValuesList*

```
Class PlayerActionValuesList
```

```
{ Kelas yang berisikan metode-metode dalam memanipulasi list bertipe PlayerActionValues }
```

Kamus

values : array of PlayerActionValues

Algoritma

PlayerActionValuesList (input playerAction : PlayerAction, input gameState : GameState, input bot : GameObject, input teleporter : Teleporter)

{ Konstruktor saat pertama kali kelas dipanggil }

 fill(playerAction, gameState, bot, teleporter)

 compute(gameState, bot)

Procedure fill (input playerAction : PlayerAction, input gameState : GameState, input bot : GameObject, input teleporter : Teleporter)

{ Mengisi value yang akan digunakan pada *list* values }

 Heading traversal [0...359]

 values.add(FORWARD, Heading)

 values.add(ACTIVATESHIELD, 0)

Iterate player from PlayerList

 values.add(FIRETORPEDOES, HeadingTo(player))

if(teleporter.count > 0 and bot.size > 45 and not teleporter.active) then

Iterate player from PlayerList

if(bot.size - 20 > player.size) then

 values.add(FIRETELEPORT, HeadingTo(player))

Repeat 20 times

 values.add(FIRETELEPORT, random)

if(teleporter.active) then

if(teleporter.position - target.position ≤ 10) then

 values.add(TELEPORT, 0)

```

else

    if(teleporter.position > target.position) then

        values.add(TELEPORT, 0)

    else

        if(teleporter.isEnemyNearTeleporter()) then

            values.add(TELEPORT, 0)

Procedure compute (input gameState : GameState, input bot : GameObject)

{ Melakukan komputasi nilai-nilai dari values List }

Iterate value from values

    value.immediateValue ← 0, value.heuristicValue ← 0

    switch(value.action)

        Case FORWARD:

            value.setNewPosition(value.heading)

            break

        Case FIRETORPEDOES:

            value.addTorpedoValue(bot, gameState)

            break

        Case FIRETELEPORT:

            value.setNewPosition()

            bot.size ← bot.size - 20

            value.addFireTeleportValue()

            break

        Case TELEPORT:

            value.setNewPosition()

            value.addHeuristicValue(-10)

            break

```

Case ACTIVATESHIELD:

```
    value.addShieldValue(bot, TorpedoList)
```

```
    break
```

```
    value.computeImmediateValueGain()
```

```
    value.computeHeuristicValueGain()
```

Function bestAction() → PlayerActionValues

{ Mengembalikan aksi dengan value tertinggi dari *list* values }

```
    values.sort()
```

```
    → values.lastElement()
```

4.1.3. Implementasi kelas *PlayerActionValues*

Class PlayerActionValues

{ Kelas yang memiliki atribut & metode untuk tiap elemen *PlayerActionValues* }

Kamus

immediateValue, heuristicValue : double

target : GameObject

Algoritma

PlayerActionValues (input bot : GameObject, input playerAction : PlayerAction

```
        input playerActions : PlayerActions, input heading : integer)
```

{ Konstruktor kelas }

setImmediateValue(0), setHeuristicValue(0),

setPlayerID(), setPlayerAction(PlayerActions),

setHeading(heading)

Procedure computePositionImmediateGain(input position : Position, input gameState :

```

GameState, input bot : GameObject, input allList : array of GameObject)
{ Melakukan perhitungan immediate value yang didapatkan untuk posisi tertentu }
iterate player from PlayerList
    addImmediateValueGainFormula(player)
iterate food from FoodList
    addImmediateValueGainFormula(food)
iterate superFood from SuperFoodList
    addImmediateValueGainFormula(superFood)
iterate poisonGas from PoisonGasList
    addImmediateValueGainFormula(poisonGas)
addImmediateValueGainFormula(outerGas)

```

Procedure computePositionHeuristicGain(input position : Position, input gameState :

```

GameState, input bot : GameObject, input allList : array of GameObject)
{ Melakukan perhitungan heuristic value yang didapatkan untuk posisi tertentu }
iterate player from PlayerList
    addHeuristicValueGainFormula(player)
iterate food from FoodList
    addHeuristicValueGainFormula(food)
iterate superFood from SuperFoodList
    addHeuristicValueGainFormula(superFood)
iterate poisonGas from PoisonGasList
    addHeuristicValueGainFormula(poisonGas)
addHeuristicValueGainFormula(outerGas)

```

Procedure addTorpedoValue(input bot : GameObject, input gameState : GameState)

```

{ Formula untuk penambahan heuristic & immediate value dari action torpedo }

```

Kamus

maxcos, p, areaLintasan, totalArea, predictedObjectInArea : double

Algoritma

```
if(not target.isShield() and bot.size > 25 and bot.salvoCount ≠ 0) then
    maxcos ← (target.size() + 10) * 20 / target.speed /
        (distanceBetween(target, bot) - bot.size - 20 - target.size)
    if(maxcos ≥ 0 and maxcos ≤ 1) then
        p ← 1 - arccos(maxcos) / 3.1415
    else
        p ← 1
    areaLintasan ← 20 * (distanceBetween(target, bot) - bot.size - target.size)
    totalArea ← 3.1415 * radius * radius
    predictedObjectInArea ← totalObjectInArea * areaLintasan / totalArea
    addHeuristicValue(20 * p * max(0, 1 - predictedObjectInArea / 4))
    addImmediateValue(-5)
```

Procedure addShieldValue(input bot : GameObject, input TorpedoList : array of GameObject)

{ Formula untuk penambahan heuristic & immendiate value dari action shield }

Kamus

flag : boolean
distance, relativeHeading, deviance : double

Algoritma

```
flag ← false
iterate torpedo from TorpedoList
    distance ← distanceBetween(bot, torpedo)
    relativeHeading ← headingBetween(bot, torpedo) - torpedo.heading
    deviance ← arcsin((bot.size + torpedo.size) / distance)
    if(relativeHeading < deviance) then
```

```

if(relativeHeading = 0) then
    relativeHeading = 0.001
    if(4 ≤ distance / torpedo.speed and distance / torpedo.speed ≤ 5) then
        flag ← true
        if(flag) then
            addHeuristicValue(1000)

```

4.2. Struktur Data

Sebagian besar struktur data pada program ini menggunakan konsep objek berdasarkan paradigma OOP (*Object Oriented Programming*).

4.2.1. Modul Models

Nama	Deskripsi
GameObject	Kelas yang memiliki data <i>id</i> , <i>size</i> , <i>speed</i> , <i>currentHeading</i> , <i>position</i> , <i>gameObjectType</i> , <i>effects</i> , <i>torpedoSalvoCount</i> , <i>supernovaAvailable</i> , <i>teleporterCount</i> , dan <i>shieldCount</i> untuk setiap objek yang diinstansiasi, seperti <i>food</i> , <i>player</i> , <i>teleporter</i> , dll.
GameState	Kelas yang memiliki data <i>world</i> , <i>gameObjects</i> , dan <i>playerGameObjects</i> untuk setiap objek yang diinstansiasi. Kelas ini menjelaskan mengenai keadaan dari permainan, berisikan keadaan peta, seluruh objek dalam permainan, dan seluruh pemain dalam permainan.
PlayerAction	Kelas yang memiliki data <i>playerId</i> , <i>action</i> , dan <i>heading</i> yang digunakan untuk mengetahui <i>action</i> dan <i>heading</i> yang dilakukan oleh <i>playerId</i> .
PlayerActionValues	Kelas yang merupakan turunan dari kelas <i>PlayerAction</i> dengan tambahan data <i>immediateValue</i> , <i>heuristicValue</i> , <i>dead</i> , dan

	<p><i>target</i> yang digunakan untuk menentukan berapa <i>value</i> dari suatu <i>action</i>. Kelas ini juga terdapat <i>methods</i> yang digunakan untuk menambah & mengurangi <i>value</i> dari suatu <i>action</i>.</p>
PlayerActionValuesList	Kelas yang memiliki data <i>list</i> of <i>PlayerActionValues</i> yang memiliki <i>methods</i> untuk mengisi <i>list</i> dengan <i>PlayerActionValues</i> yang <i>valid</i> , dan melakukan komputasi <i>value</i> yang diperoleh dari setiap elemennya. Kelas ini juga memiliki <i>method</i> <i>bestAction()</i> yang mengembalikan <i>action</i> terbaik dari <i>list</i> tersebut, yakni <i>action</i> dengan <i>value</i> terbesar.
Position	Kelas yang memiliki data <i>x</i> dan <i>y</i> yang merepresentasikan posisi. Kelas ini juga memiliki <i>method</i> tambahan <i>generateRandomPosition()</i> yang akan memberikan posisi <i>random</i> secara <i>uniform</i> .
Teleporter	Kelas yang memiliki data <i>id</i> , <i>isTeleporterActive</i> , <i>heading</i> , <i>targetPos</i> , <i>cntNotFound</i> , dan <i>cntOutput</i> . Kelas ini berfungsi untuk melakukan manipulasi terhadap teleporter yang dimiliki.
World	Kelas yang memiliki data <i>centerPoint</i> , <i>radius</i> , <i>currentTick</i> , dan <i>deltaRadius</i> . Kelas ini sebagian besar digunakan untuk merepresentasikan peta pada permainan.

4.2.2. Modul Enums

Nama	Deskripsi
Effects	Kelas yang digunakan untuk mencatat efek-efek yang <i>valid</i> dalam permainan. Kelas ini juga memiliki <i>method</i> <i>getEffects()</i> untuk memberikan informasi mengenai <i>effects</i> apa saja yang sedang dimiliki oleh <i>player</i> .
ObjectTypes	Kelas yang digunakan untuk mencatat objek-objek yang <i>valid</i> dalam permainan.

PlayerActions	Kelas yang digunakan untuk mencatat <i>actions</i> yang <i>valid</i> dalam permainan.
---------------	---

4.2.3. Modul Services

Nama	Deskripsi
Botservices	Kelas yang memiliki data <i>bot</i> , <i>playerAction</i> , <i>gameState</i> , <i>currentTick</i> , <i>prevRadius</i> , dan <i>teleporter</i> . <i>Method</i> utamanya ialah <i>computeNextPlayerAction()</i> yang akan mengembalikan <i>action</i> terbaik yang akan dilakukan oleh player selanjutnya.

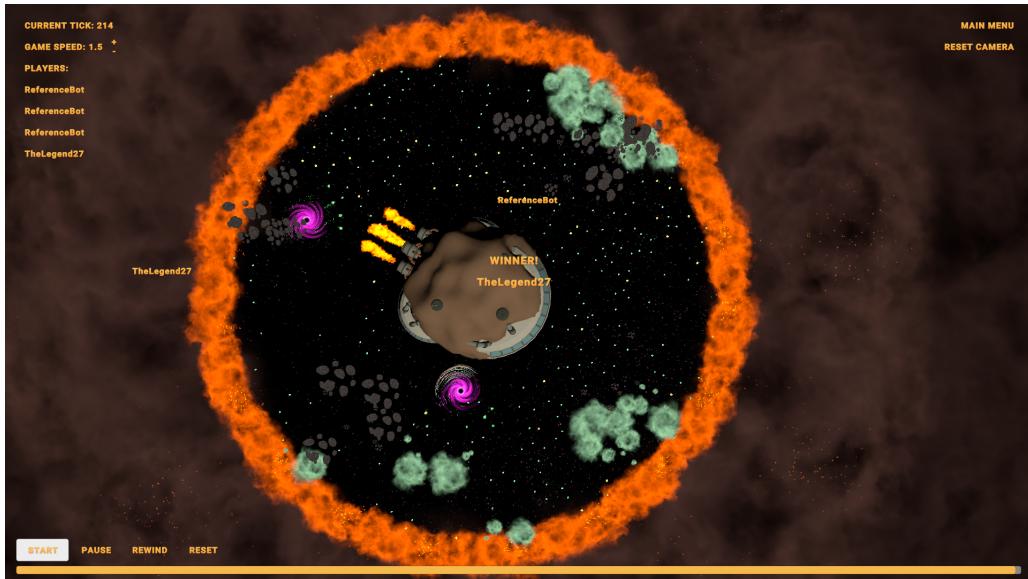
4.2.4. Kelas Main

Kelas yang digunakan untuk menginstansiasi *bot*, menghubungkan ke *runner*, menerima *game state* terkini, dan juga mengirimkan *action* player.

4.3. Analisis Pengujian

Pengujian dilakukan dengan 2 cara yakni dengan mempertandingkan bot kelompok kami dengan reference Bot dan mempertandingkan bot kelompok kami dengan bot dari kelompok lain yang dirasa cukup kuat untuk melawan bot kami.

4.3.1. Pengujian ke-1

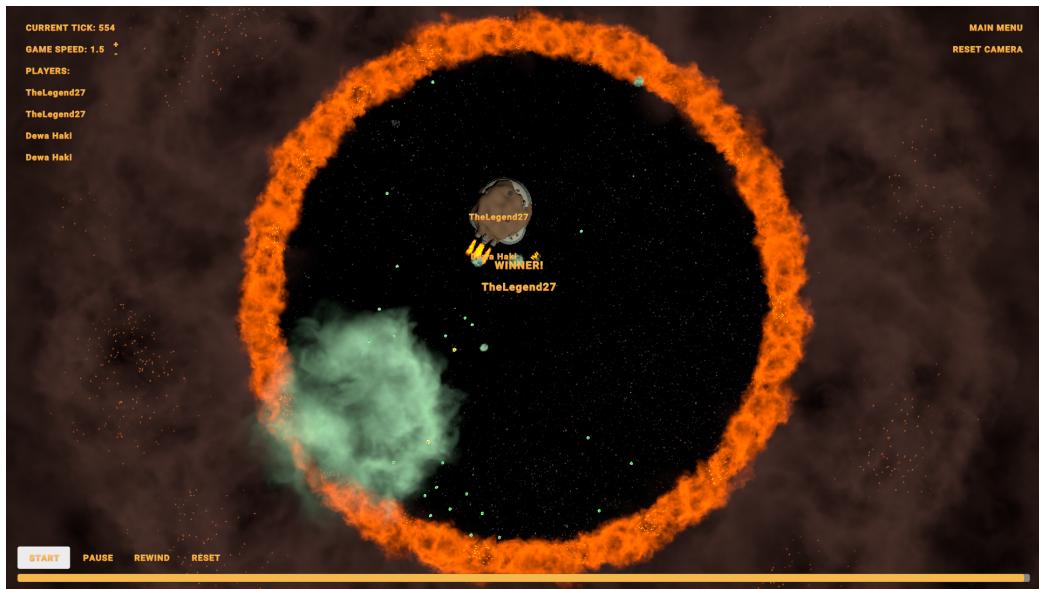


Gambar 4.1. Kemenangan bot TheLegend27 melawan reference bot

(sumber: dokumen pribadi)

Pada pengujian ke-1, bot kami mengungguli permainan melawan 3 buah reference bot. Teleport dan torpedo digunakan oleh bot kami secara tepat sasaran. Hanya sekitar 3 buah torpedo yang meleset. Salah satu hal yang membuat bot kami menang ialah aksi yang dapat dilakukan oleh reference bot masih sangatlah terbatas. Sehingga, bot kami pun telah *win streak* melawan reference bot sebanyak 27 kali.

4.3.2. Pengujian ke-2

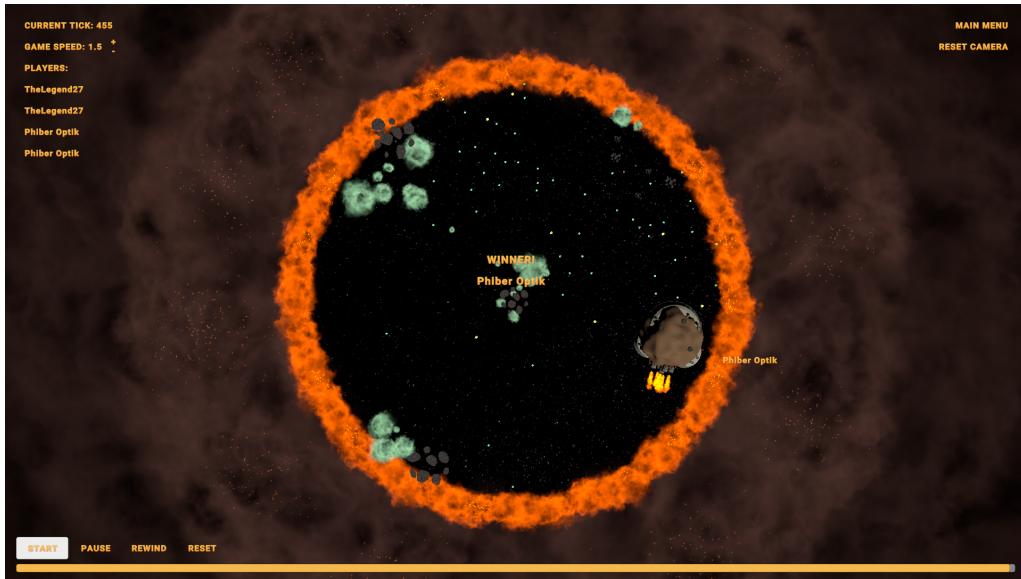


Gambar 4.2. Kemenangan bot TheLegend27 melawan bot DewaHaki

(sumber: dokumen pribadi)

Pada pengujian ke-2, dilakukan pertandingan dengan memberikan 2 bot lawan melawan 2 bot kami. Pertandingan berlangsung lumayan sengit dengan bot lawan melemparkan supernova di awal pertandingan yang membuat lawan menguasai permainan. Namun, semua berubah ketika bot kami mulai melempar-lempar torpedo yang menyebabkan bot lawan ketar ketir dan mencium. Hingga, bot kami pun dapat mengakhiri permainan dengan melakukan teleport ke posisi lawan.

4.3.3. Pengujian ke-3

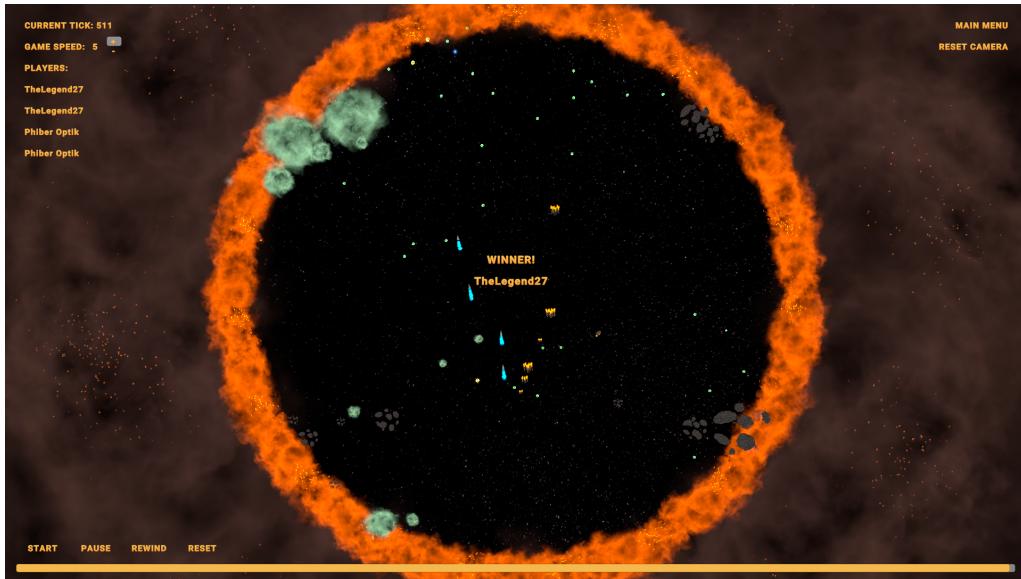


Gambar 4.3. Kekalahan bot TheLegend27 melawan bot PhiberOptik

(sumber: dokumen pribadi)

Pada pengujian ke-3, dilakukan lagi pertandingan melawan dua bot lawan yang berbeda dari sebelumnya. Pada pertandingan ini, bot kami mengalami kekalahan. Awalnya, bot kami telah mengungguli lawan dengan menghilangkan 1 bot-nya. Namun, ketika di pertengahan permainan, kedua bot kami saling lawan satu sama lain sehingga bot lawan dibiarkan *farming* terus menerus. Yang pada akhirnya, bot kami saling mengecil dan terjebak di antara *gas cloud*.

4.3.4. Pengujian ke-4



Gambar 4.4. Kemenangan bot TheLegend27 melawan bot PhiberOptik

(sumber: dokumen pribadi)

Pada pengujian ke-4, dilakukan *rematch* melawan bot sebelumnya. Pada pertandingan ini, bot kami sangat mendominasi permainan dengan melakukan *wiped out* kedua bot lawan menggunakan torpedo & teleporter sehingga menyisakan dua bot kami dalam permainan. Namun, ketika mendekati akhir permainan, salah satu bot kami melemparkan 4 buah teleporter yang merupakan aksi yang sia-sia. *Bug* ini menyebabkan ukuran bot kami mengecil 4x lipat.

BAB 5

Kesimpulan dan Saran

5.1. Kesimpulan

Algoritma yang diimplementasikan sebagai penyelesaian dari persoalan permainan Galaxio berupa strategi *greedy* dengan pendekatan nilai heuristik untuk setiap aksi. Pendekatan nilai heuristik memudahkan dalam memodifikasi strategi-strategi *greedy* sehingga menambah fleksibilitas dalam proses percobaan. Meskipun demikian, pendekatan nilai heuristik menambah kompleksitas dalam proses perhitungan dan *tuning* untuk prioritas yang relatif setara. Uji coba yang dilakukan juga membuahkan hasil yang diharapkan dengan 3x menang serta 1x kalah dalam 4x percobaan.

5.2. Saran

Pada tugas besar ini, kelompok kami masih belum mengimplementasi beberapa aksi seperti, afterburner, supernova, dll. Sehingga, kelompok kami menyarankan agar dapat mengimplementasikan aksi-aksi tersebut dan memenangkan pertandingan internasional Galaxio.

Repository Github

Berikut disertakan tautan untuk mengakses *source code* yang tersimpan pada github:

https://github.com/Michaelu670/Tubes1_TheLegend27

Daftar Pustaka

Munir, R. (2020). Algoritma Greedy Bagian 1[PDF]. Institut Teknologi Bandung. Diakses pada 14 Februari 2023 pukul 19.24 WIB melalui

[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)

Weisstein, Eric W. (n.d.). Circle-Circle Tangents. MathWorld. Diakses pada 16 Februari 2023 pukul 13.07 WIB melalui

<https://mathworld.wolfram.com/Circle-CircleTangents.html>

Gupta, Ishwar. (2022). Length of the transverse common tangent between the two non intersecting circles. GFG. Diakses pada 16 Februari 2023 pukul 13.32 WIB melalui

<https://www.geeksforgeeks.org/length-of-the-transverse-common-tangent-between-the-two-non-intersecting-circles/>