

**LAPORAN TUGAS BESAR
IF2211 STRATEGI ALGORITMA**

**Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan
“Galaxio”**



Disusun oleh :

Muhammad Rifko Favian (K1)	13521075
Moch. Sofyan Firdaus (K1)	13521083
Fazel Ginanda (K2)	13521098

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2022**

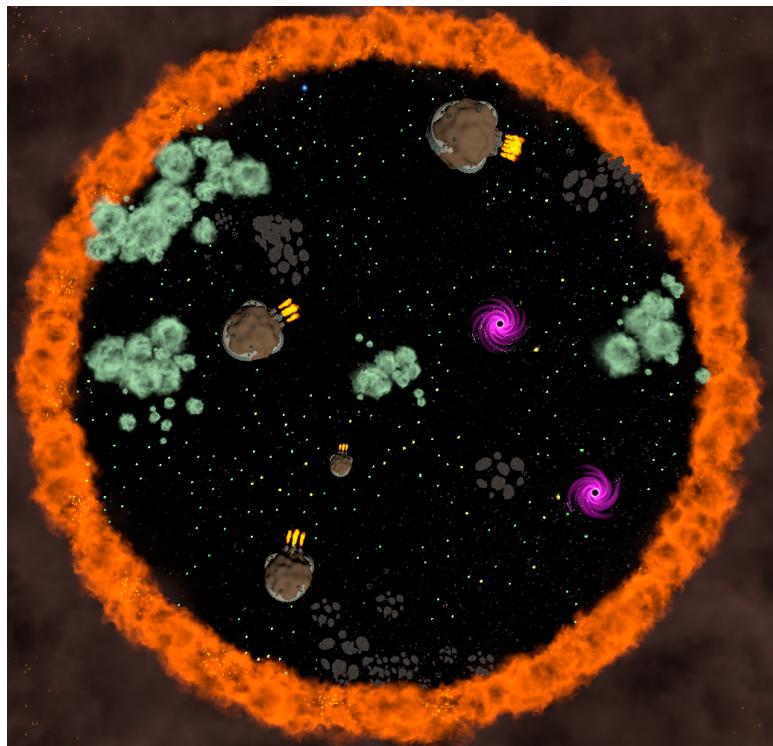
DAFTAR ISI

DAFTAR ISI.....	1
BAB I DESKRIPSI TUGAS.....	2
BAB II LANDASAN TEORI.....	5
BAB III APLIKASI STRATEGI <i>GREEDY</i>.....	10
BAB IV PENGUJIAN DAN IMPLEMENTASI.....	13
BAB V SIMPULAN DAN SARAN.....	21
GITHUB REPOSITORY.....	22
DAFTAR PUSTAKA.....	22

BAB I

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. 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 mengonsumsi 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*. Untuk daftar *commands* yang tersedia, bisa merujuk ke tautan [panduan](#) di spesifikasi tugas.
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>

BAB II

LANDASAN TEORI

2.1 Algoritma *Greedy*

Algoritma greedy merupakan algoritma yang memecahkan persoalan secara langkah per langkah sedemikian rupa sehingga pada setiap langkah diambil pilihan langkah terbaik yang dapat dilakukan saat itu tanpa perlu memikirkan konsekuensi dari langkah yang diambil. Dengan mengambil langkah yang dianggap optimum untuk setiap langkah yang diambil atau optimum lokal, diharapkan algoritma dapat berakhir dengan solusi yang merupakan optimum global.

Algoritma ini memiliki beberapa elemen umum, yaitu :

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 obyektif : memaksimumkan atau meminimumkan.

Berikut adalah skema umum dari algoritma *greedy* :

```
function greedy(C : himpunan_kandidat) → himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }

Deklarasi
x : kandidat
S : himpunan_solusi

Algoritma:
S ← {} {inisialisasi S dengan kosong}
while (not SOLUSI(S)) and (C != {}) do
    x ← SELEKSI(C) {pilih sebuah kandidat dari C}
    C ← C - {x} {buang x dari C karena sudah dipilih}
    if LAYAK(S ∪ {x}) then {x memenuhi kelayakan untuk dimasukkan ke dalam
                           himpunan solusi}
        S ← S ∪ {x} {masukkan x ke dalam himpunan solusi}
    endif
endwhile
{SOLUSI(S) or C = {}}
```

```

if SOLUSI(S) then { solusi sudah lengkap }
    return S
else
    write('tidak ada solusi')
endif

```

2.2 Starter-Pack

Pada permainan Galaxio ini sudah terdapat *game engine* yang dapat diunduh pada laman ini :
<https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2>

Yang perlu diunduh pada laman tersebut adalah file *starter-pack.zip*. Adapun struktur dari folder starter-pack adalah sebagai berikut :

```

building-a-bot.md
README.md
run.sh

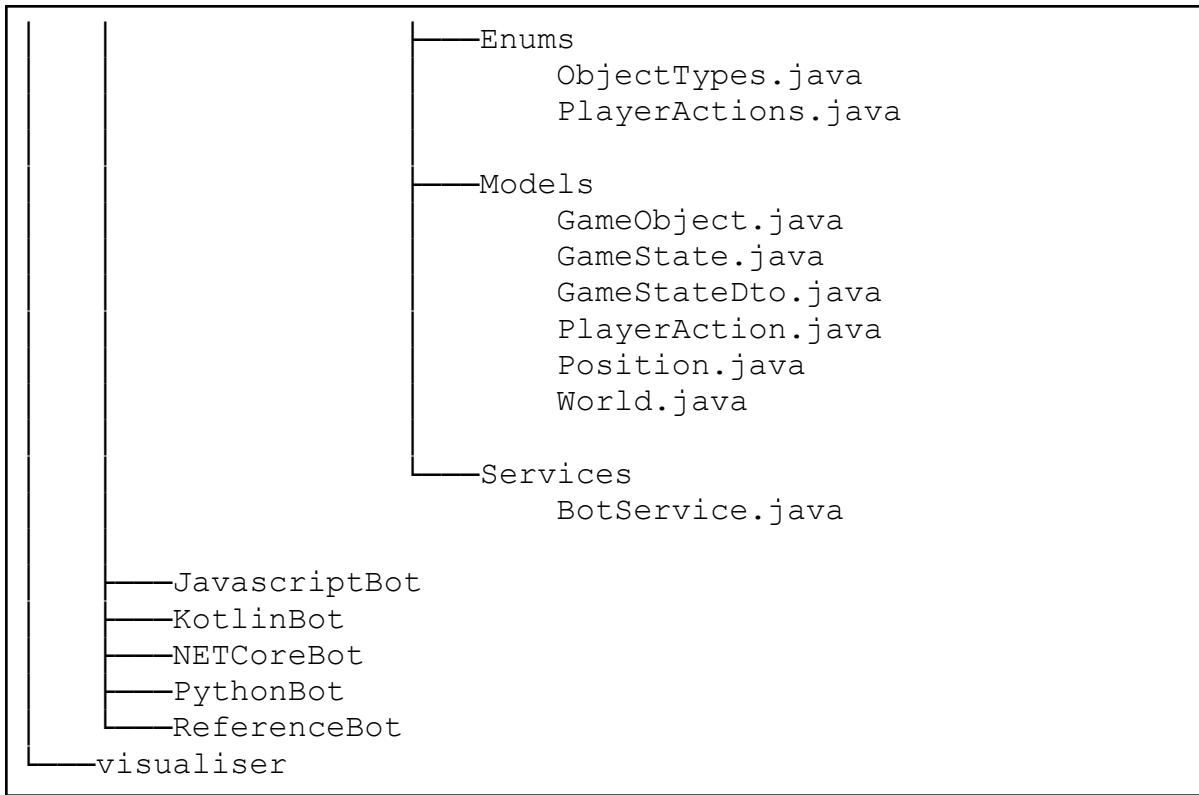
engine-publish
logger-publish
reference-bot-publish
runner-publish
starter-bots
  .gitignore
  .gitlab-ci.yml
  README.md

consoleClient
cppBot
JavaBot
  Dockerfile
  pom.xml

  .github
    workflows
      .gitkeep

src
  main
    java
      Main.java

```



Pada starter-pack, terdapat banyak bahasa yang dapat digunakan untuk memodifikasi bot, yaitu C++, Java, JavaScript, Kotlin, NET Core, dan Python. Pada tugas ini, kami menggunakan bahasa Java.

2.3 Kerja Bot Beraksi

Pada folder enums, terdapat PlayerActions.java yang berisi aksi-aksi yang dapat dilakukan bot, aksi-aksi itu adalah :

1. FORWARD = menggerakkan kapal ke arah yang ditentukan
2. STOP = berhenti menggerakkan kapal hingga perintah gerakan lain dikeluarkan
3. STARTAFTERBURNER = mengaktifkan afterburner kapal. Perhatikan, afterburner hanya akan berpengaruh saat Anda sedang bergerak. Biaya afterburner akan selalu berlaku, jika efek afterburner aktif
4. STOPAFTERBURNER = menonaktifkan afterburner
5. FIRETORPEDOES = menggunakan 1 muatan salvo dan ukuran 5 untuk mengirimkan salvo torpedo ke arah yang diberikan
6. FIRESUPERNOVA = mengirimkan supernova ke pos yang diberikan
7. DETONATESUPERNOVA = meledakkan supernova yang ada
8. FIRETELEPORTER = menghabiskan 1 biaya teleportasi dan 20 ukuran untuk mengirim teleporter ke arah yang diberikan

9. TELEPORT
10. ACTIVATESHIELD = mengaktifkan shield untuk membelokkan torpedo yang masuk, jika punya.

Jika dilihat pada Main.java, kode 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, di situlah bot bisa melakukan aksinya berdasarkan strategi yang kami buat. Selain itu, terdapat bot lain bernama ReferenceBot yang disediakan sebagai bot referensi atau sebagai lawan bot yang dimodifikasi untuk menge-test hasilnya.

2.4 Implementasi Algoritma *Greedy* pada Bot

Di sini, bahasa pemrograman yang kami gunakan adalah Java. Maka dari itu, kami mengimplementasi algoritma *greedy* pada folder bot Java, lebih tepatnya di BotService.java pada folder Service. Di dalam BotService, kami membuat strategi permainan di prosedur setUpFeedingSituation, setUpAttackingSituation, dan setUpDefendingSituation di mana bot akan menjalankan perintah-perintah yang kami buat berdasarkan situasi atau keadaan bot saat itu.

Perintah-perintah tersebut berada di Command.java di dalam folder Enums. Perintah-perintah tersebut juga diberikan atribut-atribut yang nilainya nanti disesuaikan di dalam fungsi setUpFeedingSituation, setUpAttackingSituation, dan setUpDefendingSituation. Atribut-atribut tersebutlah yang menjadi penentu strategi *greedy* yang diambil dengan cara memaksimalkan atau meminimalkan suatu nilai berdasarkan atribut-atribut tersebut.

2.5 Menjalankan game engine

Sebelum menjalankan game engine kita harus mempunyai .NET versi 3 dan versi 5. Sebelum menjalankan game, pastikan inisiasi dulu berapa bot yang mau dijalankan di game nya nanti dengan cara mengubah BotCount yang ada di appsettings.json yang ada di folder engine-publish dan runner-publish. Cara menjalankan game Galaxio ini adalah pertama dengan menjalankan GameRunner.dll yang ada di folder runner-publish terlebih dahulu, kemudian menjalankan Engine.dll yang ada di folder engine-publish, setelah itu menjalankan Logger.dll yang ada di folder logger-publish. Terakhir kita tinggal menjalankan bot yang ingin kita jalankan sebanyak yang sudah kita inisiasi sebelumnya.

Berikut adalah contoh cara menjalankan game nya dengan menggunakan shell file :

```
$ run.sh    X  
$ run.sh  
1 cd ./runner-publish && dotnet GameRunner.dll &  
2 cd ./engine-publish && sleep 1 && dotnet Engine.dll &  
3 cd ./logger-publish && sleep 1 && dotnet Logger.dll &  
4  
5 cd ./reference-bot-publish && sleep 1 && dotnet ReferenceBot.dll &  
6 cd ./reference-bot-publish && sleep 1 && dotnet ReferenceBot.dll &  
7 cd ./starter-bots/Tubes1_Bottan/target && sleep 1 && java -jar JavaBot.jar  
8  
9 wait
```

Setelah semuanya dijalankan, tunggu sampai semua selesai, setelah itu akan muncul file logger dalam folder logger-publish bernama “GameStateLog_[tanggal].json”. File logger ini dapat dijalankan pada visualizer agar bisa melihat ilustrasi pertandingan penuhnya.

BAB III

APLIKASI STRATEGI *GREEDY*

3.1 Pemetaan Persoalan Galaxio

Untuk menyelesaikan persoalan permainan Galaxio, diperlukan elemen-elemen algoritma *greedy*. Berikut adalah cara kami dalam memetakan persoalan permainan ini:

3.1.1 Himpunan Kandidat

Himpunan kandidat dalam strategi *greedy* ini adalah daftar perintah yang ada dalam enum Command di file Command.java. Perintah-perintah tersebut adalah sebagai berikut.

1. EAT_NEAREST_FOOD

Merupakan perintah agar bot maju menuju objek makanan (Food atau Superfood) terdekat untuk memakannya tetapi letaknya tidak terlalu dekat dengan batas peta.

2. ATTACK_NEAREST_OPPONENT

Merupakan perintah agar bot maju menyerang musuh terdekat yang berukuran lebih kecil darinya.

3. ESCAPE_FROM_ATTACKER

Merupakan perintah agar bot lari dari musuh yang berukuran lebih besar darinya dengan memakan objek makanan yang berlawanan arah dengan musuh.

4. FIRE_TORPEDO

Merupakan perintah agar bot menembakkan objek Torpedo Salvo ke arah musuh terdekat.

5. ACTIVATE_SHIELD

Merupakan perintah agar bot mengaktifkan efek Shield untuk melindungi dirinya dari objek Torpedo Salvo yang ditembakkan oleh musuh ke arah dirinya.

6. NOTHING

Merupakan perintah agar bot bergerak namun dengan tidak terarah (tidak berusaha menyerang/melindungi diri dan tidak berusaha untuk makan)

7. AVOID_GAS_CLOUD

Merupakan perintah agar bot menghindari masuk gas cloud dengan cara berjalan mengitari gas cloud tersebut.

8. AVOID_TORPEDOES

Merupakan perintah agar bot menghindari tembakan torpedo yang mengarah ke dirinya dengan cara membelokkan diri 15 derajat.

3.1.2 Himpunan Solusi

Himpunan solusi untuk strategi kami adalah perintah-perintah yang telah dipilih berdasarkan fungsi seleksi.

3.1.3 Fungsi Solusi

Memeriksa apakah bot dapat menang melawan bot referensi yang telah disediakan oleh pembuat permainan Galaxio

3.1.4 Fungsi Seleksi

Setiap perintah di atas memiliki atribut profit dan dangerLevel. Nilai dari atribut ini disesuaikan dengan situasi dan keadaan lingkungan di sekitar bot pada saat itu. Strategi *greedy* yang kami buat adalah strategi *greedy* berdasarkan maksimum densitas dengan densitas adalah rasio dari profit dan dangerLevel.

Nilai atribut profit dan dangerLevel disesuaikan di dalam kelas BotService. Metode yang kami gunakan untuk menentukan nilai kedua atribut tersebut adalah dengan cara heuristik. Untuk setiap perintah, atribut profit ditentukan dengan suatu rumus matematis yang kami tentukan dengan membebankan poin-poin tertentu yang dinilai penting untuk eksekusi masing-masing perintah. Untuk atribut dangerLevel, kami menentukan dengan rumus matematis dan juga dengan *hardcode*.

3.1.5 Fungsi Kelayakan

Kelayakan dari perintah yang telah dipilih ditentukan berdasarkan atribut dangerLevel. DangerLevel merupakan enum yang terdiri dari LOW, MODERATE, HIGH, VERY_HIGH, dan EXTREME. Setiap enum memiliki nilai representasi integer masing-masing, yaitu 1 sampai 5 secara berurutan. Perintah yang dipilih dikatakan layak jika atribut dangerLevel-nya tidak bernilai EXTREME. Jika perintah yang terpilih ternyata memiliki dangerLevel EXTREME, maka perintah dengan densitas terbesar selanjutnya akan dipilih.

3.1.6 Fungsi Obyektif

Tujuan dari strategi ini adalah agar bot yang kami buat dapat memenangkan pertandingan *battle royal* melawan bot-bot yang lain sehingga diharapkan bot yang kami buat bertahan sampai akhir permainan. Secara kuantitas, tujuan ini dicapai dengan memaksimumkan profit yang didapat sekaligus meminimumkan dangerLevel-nya.

3.2 Alternatif Solusi *Greedy*

3.2.1 Greedy by Nearest Food (Alternatif 1)

Pada solusi ini, bot akan memiliki prioritas untuk berusaha mengambil makanan terus menerus agar ia dapat menjadi yang paling besar terlebih dahulu, setelah makanan sudah habis, bot baru akan memakan musuh-musuh yang ada sampai menang. Dengan adanya prioritas ini, maka apabila terdapat musuh di makanan yang terdekat, ia akan tetap berusaha mengambil makanan tersebut, walaupun akan berakhir dimakan atau malah memakan musuh tersebut.

3.2.2 Greedy by Maximum Profit (Alternatif 2)

Pada solusi ini, bot akan menghitung profit terlebih dahulu berdasarkan situasi dan keadaan lingkungan di sekitarnya. Setelah dihitung, bot akan menjalankan aksi yang di mana profit nya yang paling tinggi. Profit yang dimaksud adalah seperti menyerang musuh yang cukup besar namun tidak melebihi dirinya, memakan makanan terdekat, dan lain-lain.

3.2.3 Greedy by Minimum Danger (Alternatif 3)

Pada solusi ini, bot akan menghitung bahaya yang ada berdasarkan situasi dan keadaan lingkungan di sekitarnya. Melalui perhitungan tersebut, bot akan melakukan serangkaian aksi yang memiliki bahaya paling rendah. Bahaya yang dimaksud adalah musuh dengan ukuran yang lebih besar dari bot, *gas cloud*, dan serangan torpedo musuh.

3.2.4 Greedy by Density (Alternatif 4, Pilihan Utama)

Solusi ini adalah solusi yang dipilih oleh penulis untuk digunakan dalam program ini. Dengan solusi ini, dilakukan perhitungan terhadap profit dan bahaya dari keadaan di sekitar bot secara simultan. Setelah itu, dilakukan perhitungan densitas yang didefinisikan sebagai nilai profit dibagi nilai danger. Dengan mempertimbangkan profit dan bahaya secara sekaligus dan menghitung densitasnya, penulis berpendapat akan tercapai pilihan yang optimal dalam mendukung keberlanjutan hidup bot. Selain itu, profit dan bahaya tersebut juga merupakan penyederhanaan model lingkungan game *Galaxio*. Berbagai hal yang mempengaruhi game dan kemenangan dan juga kekalahan dari bot-bot dalam game ditentukan oleh profit dan bahaya ini. Oleh karena itu, alternatif solusi ini adalah solusi yang menyeluruh dan mempertimbangkan segala keuntungan dan kerugian.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Program dalam Game Engine

Algoritma *greedy* diimplementasikan pada program ini dalam file BotService.java yang berada pada folder Services. Di dalam file tersebut, terdapat sebuah prosedur utama computeNextPlayerAction yang menentukan aksi yang dilakukan oleh bot. Prosedur itu menggunakan tiga prosedur lain dalam menjalankan aksinya. Penulis mengimplementasikan algoritma *greedy* dalam tiga prosedur tersebut. Baris-baris yang diawali dengan ‘//’ adalah komentar untuk memperjelas *pseudocode*.

```
procedure setUpFeedingSituation(input gameObjects)
// menyesuaikan nilai profit dan dangerLevel untuk command-command bersifat feeding
if isEmpty(gameObjects) then
    return

setProfit("EAT_NEAREST_FOOD", 0)
setDangerLevel("EAT_NEAREST_FOOD", "LOW")

nearestFood ← getNearestFood(gameObjects)
if nearestFood ≠ then
    setProfit("EAT_NEAREST_FOOD", getBotSpeed())
    if isEffectActive("SUPERFOOD") then
        setProfit(getBotSpeed() * 2)

    // menyesuaikan dangerLevel berdasarkan jarak makanan dari center, semakin jauh
    // maka semakin berbahaya
    foodDst ← getDstToCenter(nearestFood)

    // normalisasi nilai berdasarkan range 0 sampai radius map
    normalizedValue ← normalize(foodDst, 0, getWorldRadius())

    // membatasi dangerLevel di range 1-5
    dangerLevel ← min(5, max(1, normalizedValue))
    setDangerLevel("EAT_NEAREST_FOOD", Danger(dangerLevel))

procedure setUpDefendingSitution(input gameObjects)
// menyesuaikan nilai profit dan dangerLevel untuk command-command bersifat defending
if isEmpty(gameObjects) then
    return

// torpedo salvo yang mengarah ke bot
torpedoesCount ← getAttackingTorpedoes(gameObjects)
```

```

// menyesuaikan dangerLevel berdasarkan akar dari ukuran bot, semakin kecil semakin
// berbahaya
botSize ← sqrt(getBotSize())
// normalisasi botSize berdasarkan range 7-20
normalizedValue ← normalize(botSize, 7, 20)
dangerLevel ← max(1, min(5 - normalizedValue * 5, 5))
setDangerLevel("ACTIVATE_SHIELD", Danger(dangerLevel))
// menyesuaikan profit berdasarkan efek shield dan torpedoesCount
if isEffectActive("SHIELD") and getShieldCount() > 0 then
    profit ← 0
else
    profit ← torpedoesCount * getDangerLevel("ATTACK_NEAREST OPPONENT") * 7
setProfit("ACIVATE_SHIELD", profit)

// menyesuaikan profit berdasarkan efek shield, torpedoesCount, dan botSpeed
if isEffectActive("SHIELD") then
    profit ← 0
else
    profit ← torpedoesCount * getBotSize()
setProfit("AVOID_TORPEDOES", profit)

// menyesuaikan profit berdasarkan bot speed, bot size, dan efek GAS_CLOUD
if isEffectActive("SHIELD") then
    profit ← 0
else
    profit ← getBotSpeed() * getBotSize() / 2
setProfit("AVOID_TORPEDOES", profit)

procedure setUpAttackingSituation(input gameObjects)
// menyesuaikan nilai profit dan dangerLevel untuk command-command bersifat feeding
if isEmpty(gameObjects) then
    return

opponents ← getOpponents(gameObjects)
// musuh terdekat
enemy1 ← getNearestOpponent(opponents)
// musuh terdekat ke enemy1, null jika tidak ada
enemy2 ← getNearestOpponentFrom(opponents, enemy1)

// jarak bot dari enemy1
x ← getDistanceFrom(enemy1)
normalizedX ← normalize(x, 0, getWorldRadius() / 4)
if getTorpedoSalvoCount() > 0 then
    profit ← getSize(enemy1) / normalizedX
else
    profit ← 0
setProfit("FIRE_TORPEDO", profit)

normalizeBotSize ← normalize(sqrt(getBotSize()), 5, 15)

```

```

dangerLevel ← max(1, min(5, 5 - dangerLevel * 5))
setDangerLevel("FIRE_TORPEDO", Danger(dangerLevel))

setProfit("ATTACK_NEAREST_OPPONENT", getSize(enemy1) / 2)

if (getSize(enemy1) > getBotSize() - 6) then
    setDangerLevel("ATTACK_NEAREST_OPPONENT", "EXTREME")
    setProfit("ESCAPE_FROM_ATTACKER", getSize(enemy1) - x)
else
    setDangerLevel("ATTACK_NEAREST_OPPONENT", "MODERATE")
    if enemy2 ≠ null then
        if getDistanceBetween(enemy1, enemy2) < x then
            setDangerLevel("ATTACK_NEAREST_OPPONENT", "VERY_HIGH")
        else
            setDangerLevel("ATTACK_NEAREST_OPPONENT", "HIGH")

procedure determineCommand(input commands, output command)
// fungsi seleksi algoritma greedy
sortCommandByDensity(commands)
// mengecek kelayakan command
index ← 0
repeat
    command ← getElement(commands, index)
    index ← index + 1
until getDangerLevel(command) ≠ DangerValue("EXTREME")
    or index ≥ length(commands)

```

4.2 Penjelasan Struktur Data

4.2.1 Command

Command adalah kelas bertipe enum yang memiliki atribut dan method yang dibutuhkan oleh bot untuk menjalankan semua aksi. Terdapat delapan method yang menjadi anggota kelas ini, yaitu NOTHING, EAT_NEAREST_FOOD, ATTACK_NEAREST_OPPONENT, ESCAPE_FROM_ATTACKER, FIRE_TORPEDO, ACTIVATE_SHIELD, AVOID_GAS_CLOUD, AVOID_TORPEDOES. Selain itu, hal penting dari kelas Command adalah atribut profit dan dangerLevel. Profit bertipe integer yang menyatakan keuntungan yang didapatkan bot ketika menjalankan suatu aksi. Sementara itu, dangerLevel bertipe DangerLevel yang didefinisikan dalam kelas terpisah.

4.2.2 DangerLevel

DangerLevel merupakan kelas yang mendefinisikan tingkat bahaya yang dihadapi oleh bot ketika menjalankan suatu aksi. Kelas ini memiliki atribut value yang bertipe integer yang merupakan representasi numerik tingkat bahaya. Kemudian, skala angka tersebut dinyatakan dalam bentuk kata, mulai dari LOW untuk value bernilai 1 hingga EXTREME untuk value bernilai 5.

4.2.3 Effects

Effects berfungsi untuk mendefinisikan efek-efek yang ada dalam game *Galaxio*. Effects direpresentasikan dalam array dengan elemennya adalah effects. Selain itu, terdapat juga method untuk mengembalikan efek-efek yang aktif dalam game.

4.2.4 ObjectTypes

ObjectTypes adalah enum yang mendefinisikan objek-objek yang ada dalam permainan *Galaxio*. Terdapat sebelas objek yang direpresentasikan sebagai integer.

4.2.5 PlayerActions

PlayerActions adalah enum yang mendefinisikan apa saja aksi-aksi yang dapat dilakukan oleh sebuah player.

4.2.6 GameObject

GameObject adalah kerangka umum objek-objek di dalam game.

4.2.7 GameState

GameState adalah kerangka umum *state-state* di dalam game.

4.2.8 GameStateDto

GameStateDto adalah kerangka umum struktur state di dalam game yang nantinya akan digunakan oleh hub untuk mengonversi data yang dikirimkannya ke dalam objek di dalam java

4.2.9 PlayerAction

PlayerAction adalah kerangka umum aksi-aksi player yang akan dikirim ke hub game.

4.2.10 Position

Position adalah kerangka umum posisi untuk semua objek di dalam game.

4.2.11 World

World adalah state yang berkaitan dengan peta di dalam game dan keberlangsungan game.

4.2.12 CommandLogic

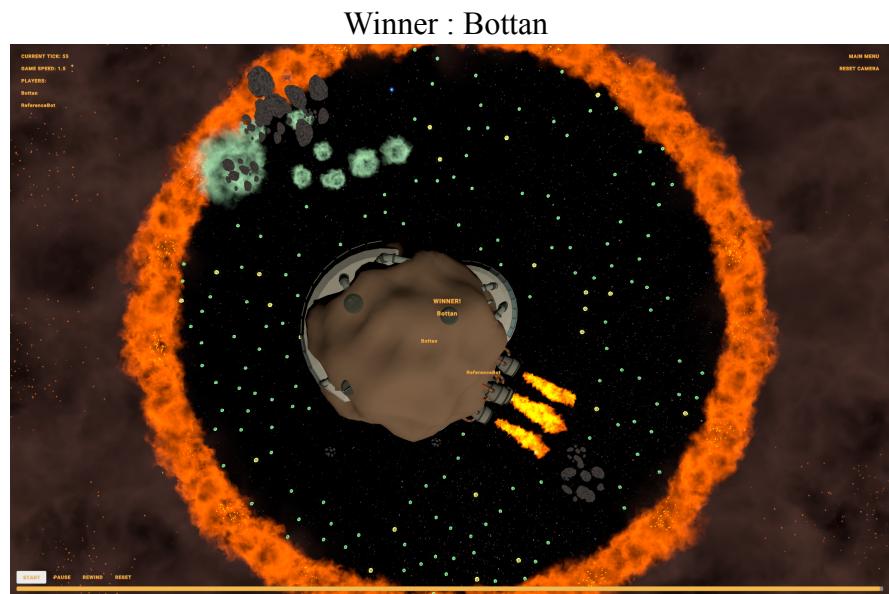
CommandLogic adalah interface untuk logic yang akan dieksekusi di setiap Command.

4.2.13 Util

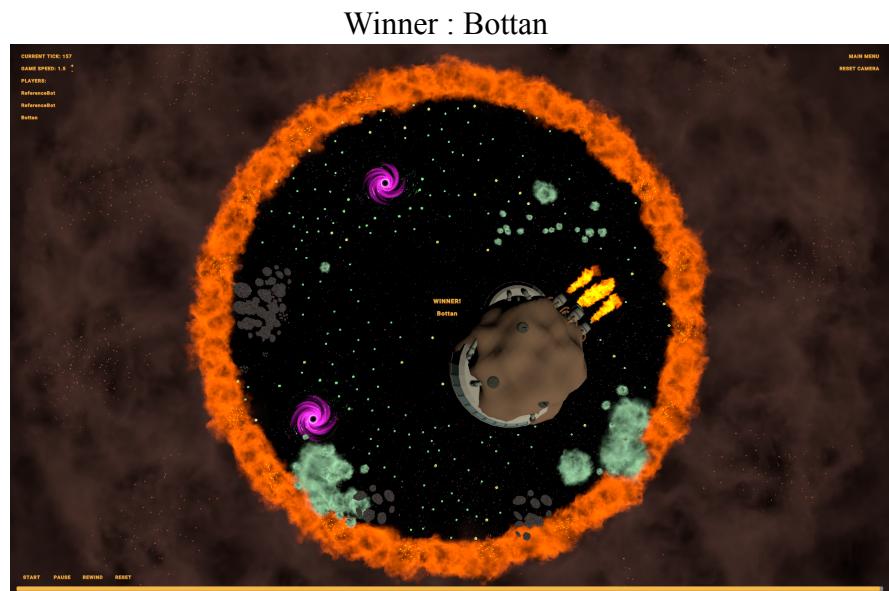
Util adalah seperangkat atribut dan method yang diperlukan untuk menjalankan game meliputi penentuan posisi dan penghitungan jarak.

4.3 Analisis Solusi dari Pengujian

Pengujian 1

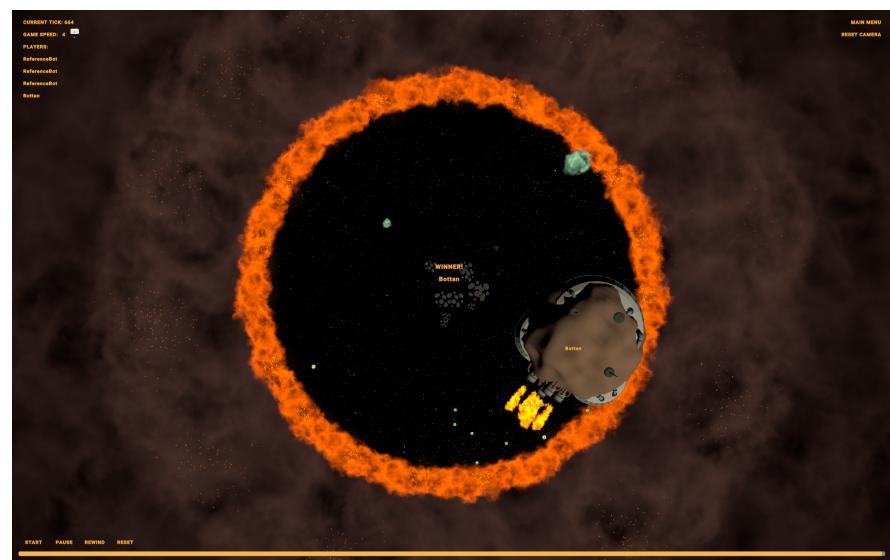


Pengujian 2

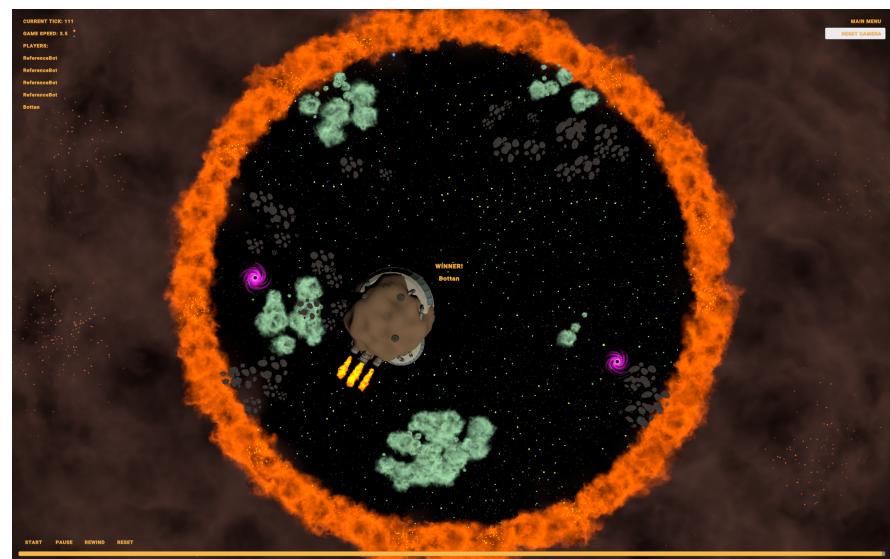


Pengujian 3

Winner : Bottan

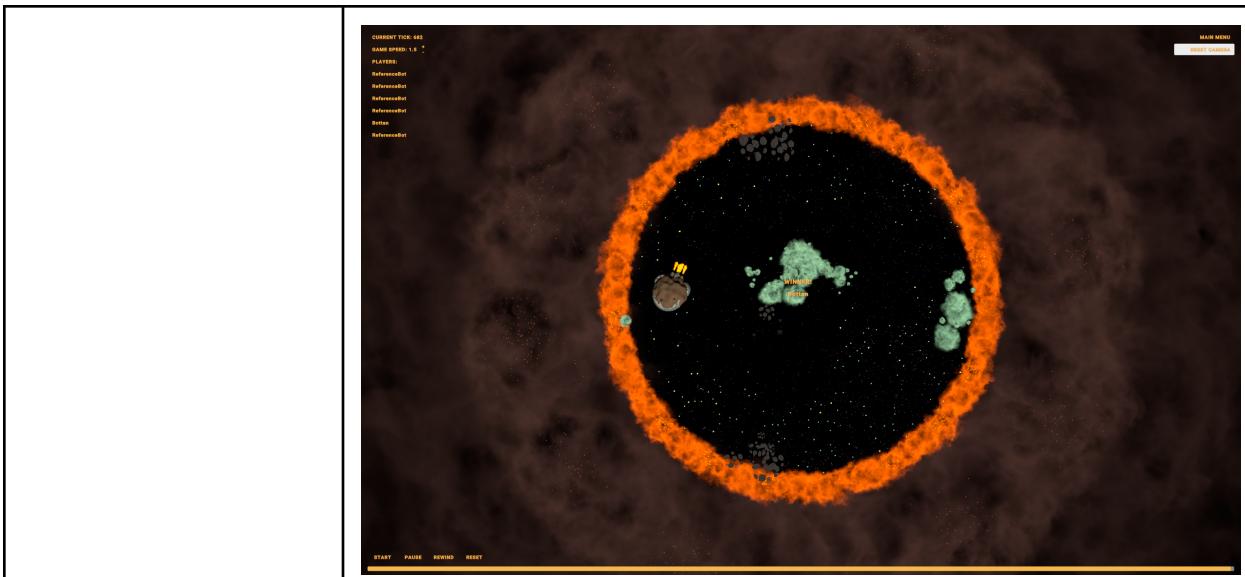


Pengujian 4



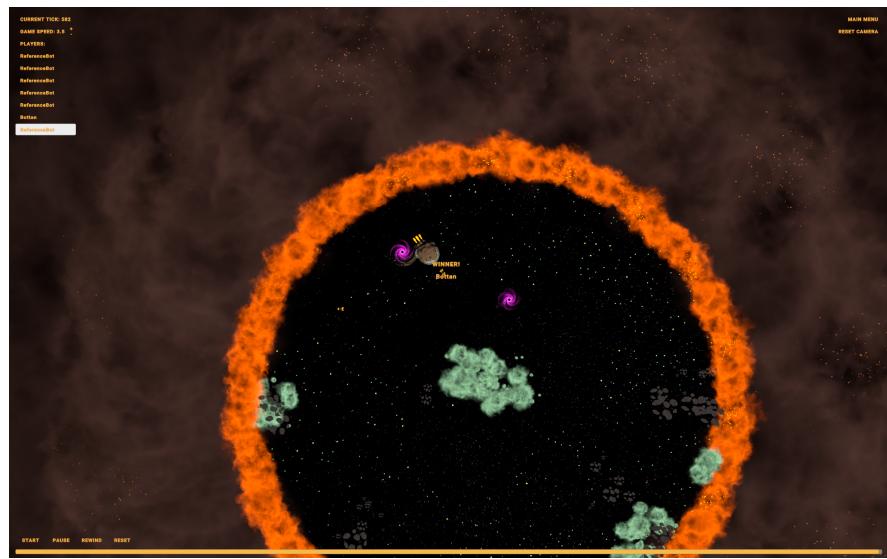
Pengujian 5

Winner : Bottan



Pengujian 6

Winner : Bottan



Pengujian kami lakukan dengan mencoba melawan bot dari 1 bot sampai 6. Berdasarkan hasil 6 pengujian tersebut di atas, bot yang kami buat dengan implementasi algoritma *greedy* dapat memenangkan seluruh pertandingan yang diuji. Oleh karena itu, dapat dinyatakan bahwa persentase kemenangan bot adalah 100%.

Meskipun persentasenya 100%, bot ini masih jauh dari kata sempurna dalam implementasinya di algoritma *greedy*, karena kami tidak menggunakan `playerAction` dari `AfterBurner`, `FireSupernova`, dan `FireTeleporter`. Selain itu, kami tidak membuat bot melihat Asteroid sebagai obstacle, jadi apabila bot berjalan menuju arah asteroid, bot akan tetap terus berjalan walaupun menabrak asteroid tersebut, karena bagi bot tersebut asteroid itu tidak ada.

Dan juga, pengujian yang kami lakukan ini adalah pengujian terhadap reference bot yang di mana bot tersebut masih memiliki strategi permainan yang mungkin bisa dikatakan sangat sedikit. Jadi, pengujian yang kami lakukan tidak bisa menjadi pegangan untuk bisa mengatakan “bot kami pasti bisa menang” karena masih ada kemungkinan besar bot yang kami buat bisa kalah dengan bot buatan orang lain. Selain itu, bot yang kami buat ini masih hanya mengimplementasikan algoritma *greedy* yang seperti namanya “*greedy*” di mana hanya fokus menghadapi apa yang sedang terjadi.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada tugas besar pertama mata kuliah IF2211 Strategi Algoritma ini, penggunaan algoritma *greedy* untuk bot kapal selam dalam permainan *Galaxio* telah berhasil diimplementasikan. Melalui implementasi ini, penulis merumuskan solusi untuk membuat bot kapal selam berdasarkan teori algoritma *greedy* yang diajarkan di kelas dan juga pengamatan terhadap alur permainan *Galaxio*.

Penggunaan algoritma *greedy* dalam program diimplementasikan pada file BotService.java yang berada di folder Services. Setelah implementasi *greedy* berhasil, dilakukan pengujian dalam permainan. Dalam enam permainan, bot berhasil memenangkan keenam pertandingan tersebut. Oleh karena itu, dapat dinyatakan bahwa persentase kemenangan bot adalah 100%.

Dengan demikian, dapat disimpulkan bahwa melalui tugas besar pertama mata kuliah IF2211 Strategi Algoritma ini, permainan game *Galaxio* dapat dimenangkan melalui pendekatan algoritma *greedy* yang diimplementasikan dalam bentuk bot kapal selam.

5.2 Saran

Pengerjaan tugas besar ini memberikan banyak pelajaran bagi penulis. Hal ini dilatarbelakangi oleh spesifikasi tugas maupun kondisi perkuliahan saat tugas ini diberikan. Pelajaran itu didapat ketika menemukan berbagai kendala maupun hal-hal menarik dalam pengerjaan tugas besar ini. Oleh karena itu, penulis memberikan saran bagi pihak-pihak yang ingin mengerjakan hal yang sama ataupun ingin melakukan pendalaman lebih lanjut sebagai berikut.

1. Diperlukan diskusi dan kajian yang mendalam sebelum memulai penulisan kode program. Perancang program harus mengetahui persis bagaimana langkah-langkah implementasi strategi *greedy* dalam permainan *galaxio*.
2. Diperlukan pengujian secara terus-menerus untuk mengetahui aspek apa yang perlu ditingkatkan dari pergerakan bot
3. Diperlukan koordinasi dan kerja sama yang baik dalam pengerjaan tugas secara berkelompok. Dengan adanya komunikasi serta kerja sama yang baik, maka ide-ide dapat diaktualisasikan dengan tepat.

GITHUB REPOSITORY

Link Github Repository : https://github.com/Mifkiyan/Tubes1_Bottan

DAFTAR PUSTAKA

[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)