**LAPORAN TUGAS BESAR**

**IF2**211Strategi Algoritma

Bot Etimo Diamonds



Dipersiapkan oleh:

Muhammad Dzaki Arta 13522149  
Muhammad Fauzan Azhim 13522153  
Pradipta Rafa Mahesa 13522162

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 4013

**Daftar Isi**

[**BAB I**](#_heading=h.k1n2hve18b0m) [**Deskripsi Tugas 3**](#_heading=h.qia21agkvhrt)

[Bab 2 Landasan Teori 7](#_heading=)

[2.1 Algoritma Greedy 7](#_heading=h.uc5h9eiyoudf)

[2.2 Diamonds dari Etimo 7](#_heading=h.ksu2kirbqj61)

[BAB 3 Aplikasi Strategi Greedy 9](#_heading=)

[3.1 Mapping Persoalan 9](#_heading=h.35hvrvz0bkiw)

[3.2 Eksplorasi 9](#_heading=h.ypxfp5vcij8c)

[BAB 4 Implementasi dan Pengujian 11](#_heading=)

[4.1 PseudoCode 11](#_heading=h.owgxoe2u7ix0)

[4.2 Hasi Uji Coba 16](#_heading=h.869a66ng9ks1)

[4.3 Analisis 16](#_heading=h.v6bvc1jqhkya)

[Bab 5 Kesimpulan dan Saran 17](#_heading=)

[5.1 Kesimpulan 17](#_heading=h.lgxst8fezcrl)

[5.2 Saran 17](#_heading=h.ufcgw45oviqm)

[Lampiran 18](#_heading=)

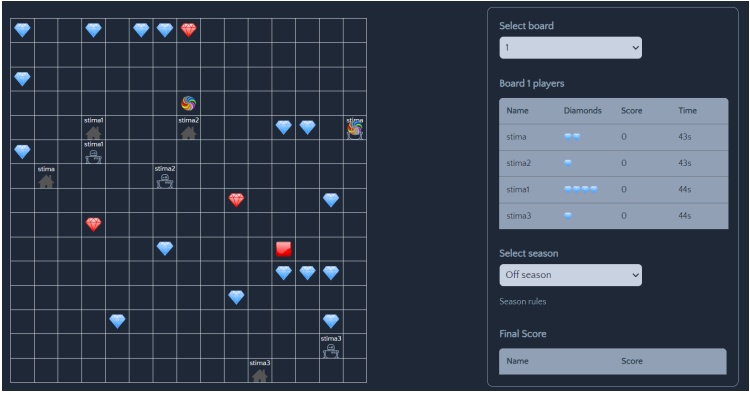
[Daftar Pustaka 19](#_heading=)

# 

# BAB I

# Deskripsi Tugas

Diamonds merupakan suatu programming challenge yang mempertandingkan bot yang anda buat dengan bot dari para pemain lainnya. Setiap pemain akan memiliki sebuah bot dimana tujuan dari bot ini adalah mengumpulkan diamond sebanyak-banyaknya. Cara mengumpulkan diamond tersebut tidak akan sesederhana itu, tentunya akan terdapat berbagai rintangan yang akan membuat permainan ini menjadi lebih seru dan kompleks. Untuk memenangkan pertandingan, setiap pemain harus mengimplementasikan strategi tertentu pada masing-masing bot-nya. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah.



Pada tugas pertama Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Tentunya mahasiswa harus menggunakan strategi greedy dalam membuat bot ini. Program permainan Diamonds terdiri atas:

1. Game engine, yang secara umum berisi:

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

b. Kode frontend permainan, yang berfungsi untuk memvisualisasikan permainan 2. Bot starter pack, yang secara umum berisi:

a. Program untuk memanggil API yang tersedia pada backend

b. Program bot logic (bagian ini yang akan kalian implementasikan dengan algoritma greedy untuk bot kelompok kalian)

c. Program utama (main) dan utilitas lainnya

Untuk mengimplementasikan algoritma pada bot tersebut, mahasiswa dapat

menggunakan game engine dan membuat bot dari bot starter pack yang telah tersedia pada pranala berikut.

● Game engine : <https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0>

● Bot starter pack : <https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1>

Buatlah program sederhana dalam bahasa Python yang mengimplementasikan algoritma Greedy pada bot permainan Diamonds dengan tujuan memenangkan permainan. Tugas dikerjakan berkelompok dengan anggota minimal 2 orang dan maksimal 3 orang, boleh lintas kelas dan lintas kampus. Strategi greedy yang diimplementasikan setiap kelompok harus dikaitkan dengan fungsi objektif dari permainan ini, yaitu memenangkan permainan dengan memperoleh diamond sebanyak banyak nya dan jangan sampai diamond tersebut diambil oleh bot lain.

Strategi greedy yang dibuat harus dijelaskan dan ditulis secara eksplisit pada laporan, untuk diperiksa saat demo apakah strategi yang dituliskan sesuai dengan yang diimplementasikan. Tiap kelompok dapat menggunakan kreativitas yang bermacam macam dalam menyusun strategi greedy untuk memenangkan IF2211 Strategi Algoritma – Tugas Besar 1 5 permainan.

Implementasi pemain harus dapat dijalankan pada game engine yang telah disebutkan diatas serta dapat dikompetisikan dengan bot dari kelompok lain. Program harus mengandung komentar yang jelas, dan untuk setiap strategi greedy yang disebutkan, harus dilengkapi dengan kode sumber yang dibuat.

Terdapat beberapa komponen dalam permainan ini :

1. **Diamonds**

****

Untuk memenangkan permainan kita harus mengumpulkan diamond sebanyak mungkin. Untuk mendapatkan diamond bot harus melewatinya. Terdapat 2 jenis diamond yaitu diamond biru yang bernilai 1 poin dan diamond merah yang bernilai 2 poin. Diamond akan selalu muncul secara berkala dengan rasio diamond merah dan biru akan selalu berubah dalam setiap kemunculan.

1. **Red Button/ Diamond Button**

****

Ketika Red Button ini di lewati oleh bot maka seluruh diamond yang ada dalam permainan akan tereset dengan posisi yang berubah dibandingkan awalnya

1. **Teleporters**

****

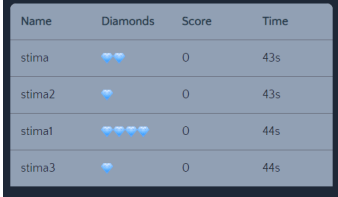
Ketika bot memasuki salah satu teleporter maka bot akan langsung berpindah ke tempat teleporter lainnya. Teleporter hanya berisi 2 dalam map

1. **Bots dan Bases**

****

Pada game ini akan digerakan bot untuk menangkap diamond sebanyak-banyaknya untuk memenangkan permainan. Bots akan selalu muncul dari bases setiap awal mulainya permainan. Bases merupakan tempat munculnya bots dan tempat menyimpan diamonds yang akan bot bawa nantinya. Diamond akan dimunculkan ke scoreboard saat bot membawa diamond ke dalam base.

1. **Inventory**

****

Bot memiliki inventory untuk menyimpan diamond sebelum dibawa ke dalam base. Inventory memiliki kapasitas maksimum sehingga bot tidak bisa selalu mengambil diamond yang ada dan harus menyimpannya ke dalam base untuk mengosongkan inventory yang ada.

Untuk mengetahui flow dari game ini, berikut ini adalah cara kerja permainan Diamonds :

1. Pertama, setiap pemain (bot) akan ditempatkan pada board secara random. Masing-masing bot akan mempunyai home base, serta memiliki score dan inventory awal bernilai nol.
2. Setiap bot diberikan waktu untuk bergerak, waktu yang diberikan semua sama untuk setiap pemain.
3. Objektif utama bot adalah mengambil diamond-diamond yang ada di peta sebanyak-banyaknya. Seperti yang sudah disebutkan di atas, diamond yang berwarna merah memiliki 2 poin dan diamond yang berwarna biru memiliki 1 poin.
4. Setiap bot juga memiliki sebuah inventory, dimana inventory berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini sewaktu-waktu bisa penuh, maka dari itu bot harus segera kembali ke home base.
5. Apabila bot menuju ke posisi home base, score bot akan bertambah senilai diamond yang tersimpan pada inventory dan inventory bot akan menjadi kosong kembali.
6. Usahakan agar bot anda tidak bertemu dengan bot lawan. Jika bot A menimpa posisi bot B, bot B akan dikirim ke home base dan semua diamond pada inventory bot B akan hilang, diambil masuk ke inventory bot A (istilahnya tackle).
7. Selain itu, terdapat beberapa fitur tambahan seperti teleporter dan red button yang dapat digunakan apabila anda menuju posisi objek tersebut.
8. Apabila waktu seluruh bot telah berakhir, maka permainan berakhir. Score masing-masing pemain akan ditampilkan pada tabel Final Score di sisi kanan layar.

# Bab 2

# Landasan Teori

## 2.1 Algoritma Greedy

Algoritma Greedy adalah sebuah pendekatan algoritma yang mencoba mencari solusi optimal pada setiap tahap dengan harapan menemukan solusi global optimal. Dalam kata lain, algoritma ini membuat pilihan yang tampaknya terbaik pada saat itu, dengan harapan pilihan ini akan mengarah ke solusi optimal global.

Algoritma Greedy tidak selalu menghasilkan solusi yang paling optimal, namun mereka biasanya menghasilkan solusi yang cukup dekat dengan optimal dalam waktu yang jauh lebih cepat dibandingkan algoritma lainnya. Algoritma ini mengorbankan beberapa keakuratan dan mendapatkan solusi yang sub-optimal untuk mendapatkan kecepatan perhitungan yang lebih baik.

Oleh karena itu, jika solusi terbaik mutlak tidak terlalu diperlukan, maka algoritma greedy dapat digunakan untuk menghasilkan solusi hampiran. Namun bila algoritma greedy dapat menghasilkan solusi optimal, maka keoptimalannya itu harus dapat dibuktikan secara matematis. Karena itu juga, lebih mudah memperlihatkan algoritma greedy tidak selalu optimal dengan menunjukkan counter example (contoh kasus yang menunjukkan solusi yang diperoleh tidak optimal). Misalnya pada persoalan penukaran uang dimana solusinya terkadang optimal dan tidak optimal.

Komponen-komponen dari algoritma Greedy meliputi:

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.

## 2.2 Diamonds dari Etimo

Diamonds Etimo adalah tantangan untuk programmer yang dibuat oleh organisasi bernama “*Etimo”* di github. Tantangan ini ditujukan untuk programmer mencari algoritma terbaik untuk mendapatkan diamonds terbanyak dalam rentang waktu yang ada. Ditambah ada beberapa faktor lain selain strategi algoritma mencari diamond terbanyak, yaitu adanya teleporter, diamond reset button, dan tackle. Teleporter dapat memindahkan player dari titik teleport yang satu ke titik teleport yang lain nya. Diamond reset button akan mereset semua diamond yang ada di board dan memunculkan diamond baru. Tackle dapat dilakukan oleh pemain untuk mencuri diamond yang dimiliki oleh player lain dan memindahkan player yang di tackle kembali ke base player tersebut.

Diamonds Etimo memiliki berbagai season dengan setting yang berbeda-beda. Setiap seasonnya memiliki kombinasi algoritma dari komponen board yang unik. Kombinasi unik dari setiap season ini membuat game ini menjadi sangat beragam dengan banyaknya kemungkinan strategi yang dapat dilakukan. Baik mengincar diamond dan menghindari musuh sampai mengincar tackle agar musuh tidak bisa mendapatkan diamond.

## 2.3 Bot

"BOT" atau "bot" adalah singkatan dari "robot". Bot adalah program komputer yang otomatis dan independen, yang dapat melakukan tugas-tugas tertentu. Bot biasanya diprogram untuk melakukan tugas yang berulang atau monoton. Karena *bot* dapat melakukan repetisi dan akan melakukan tugas sesuai dengan yang kita programkan, kita dapat mengimplementasikan *greedy* kepada bot dengan logic yang simple.

Salah satu caranya adalah dengan memasukkan data-data atau informasi yang kita punya. Lalu olah informasi tersebut menjadi angka desimal atau float. Setelah itu kita tinggal menentukan mau memilih yang paling maksimal atau minimal sesuai dengan hasil yang kita inginkan.

Setelah algoritma terpasang pada bot tentunya kita harus melakukan testing hasil apakah hasil yang diberikan sudah sesuai dengan yang diinginkan, lalu jika sudah sesuai lanjutkan dengan apakah hasil sudah optimal. Dilanjutkan dengan membuat handle jika ada edge-cases yang mungkin bisa terjadi. Setelah semua sudah dilakukan tahap selanjutnya adalah tahap analisis. Pada tahap ini kita dapat mengetahui apa saja kelemahan dari algoritma pada bot tersebut, kelebihannya, dan apa saja yang bisa diperbaiki.

# BAB 3

# Aplikasi Strategi Greedy

## 3.1 Mapping Persoalan

Elemen-elemen algoritma Greedy dapat dijelaskan sebagai berikut: Himpunan kandidat dalam konteks ini adalah diamond merah, diamond biru, teleporter, diamond reset plate, dan rumah. Hal-hal tersebut adalah pilihan yang tersedia untuk dipilih oleh algoritma. Himpunan solusi adalah diamond yang diperoleh. Himpunan solusi adalah hasil dari pilihan yang dibuat oleh algoritma. Fungsi solusi, dalam hal ini, adalah memeriksa apakah inventory sudah penuh. Ini adalah cara algoritma menentukan apakah solusi telah ditemukan atau apakah lebih banyak pilihan harus dibuat. Fungsi seleksi adalah memilih diamond yang nilai per waktu untuk mencapainya maksimal. Ini adalah metode yang digunakan oleh algoritma untuk memilih pilihan berikutnya dari himpunan kandidat. Fungsi kelayakan adalah memeriksa apakah nilai diamond yang dituju melebihi inventory. Ini adalah cara algoritma memeriksa apakah pilihan yang dipilih adalah pilihan yang valid. Akhirnya, fungsi objektif adalah nilai yang diperoleh maksimal. Ini adalah tujuan akhir dari algoritma, yaitu untuk memaksimalkan nilai yang diperoleh.

## 3.2 Eksplorasi

Dari eksplorasi, kami menganalisis beberapa algoritma, antara lain :

1. **Algoritma Shortest Distance**

Algoritma ini membandingkan pilihan diamond dengan mengambil jarak diamond yang paling dekat. Algoritma ini menggunakan rumus pythagoras untuk menghitung jarak antara bot dengan diamond.

1. **Algoritma Score per Distance**

Algoritma ini menyerupai algoritma pertama tetapi menambahkan nilai diamondnya ke dalam perhitungannya. Algoritma tersebut menghitung nilai diamond per jarak yang ditempuh lalu memilih yang nilainya paling tinggi.

1. **Algoritma Biggest Diamond**

Algoritma ini mengutamakan diamond merah dalam perhitunganya.

1. **Algoritma Cluster Diamond**

Algoritma ini mencari jumlah diamond yang paling banyak dalam area 3x3.

1. **Algoritma Closest to Home**

Algoritma ini mengutamakan diamond-diamond yang berjarak dekat dari rumah.

1. **Algoritma Beset**

Algoritma ini berfokus pada membunuh bot lain untuk mendapatkan diamond bot tersebut dan mencegah bot tersebut untuk meng-*score.*

Dari hasil diskusi dan analisis, kami memilih untuk menggunakan Algoritma **Score per Distance**, hal ini karena sebagai berikut :

1. **Algoritma Shortest Distance** : Mirip seperti algoritma score per distance tetapi algoritma ini tidak mementingkan diamond yang poinnya lebih besar jika jaraknya sama. Algoritma ini juga tidak memperhitungkan banyaknya point yang didapatkan sehingga terkadang kurang cepat dalam pengambilan diamond dibandingkan **Score Per Distance**.
2. **Algoritma Biggest Diamond** : Algoritma ini terlalu memprioritaskan diamond merah yang mempunyai kemungkinan mempunyai jarak yang jauh. Jarak yang jauh ini membuat bot tidak optimal karena perbedaan point diamond merah dan diamond biru hanya satu point.
3. **Algoritma Cluster Diamond** : Pencarian ini meningkatkan pencarian diamond dengan jarak sebanyak 95 . Karena mahalnya komputasi cluster diamond saat ini, kemungkinan untuk waktu komputasi dari algoritma ini sendiri dapat memakan waktu lebih dari satu detik dan bot akan berjalan lebih lama dari bot lainnya.
4. **Algoritma Closest to Home** : Jarak diamond dari rumah terkadang relatif jauh terhadap posisi saat ini. Karena jarak dari rumah ke diamond bisa digambarkan seperti lingkaran dengan jarak sebagai jari-jari, bisa saja kita membuat pergerakan jauh yang berada di sisi lain dari rumah.
5. **Algoritma Beset** : Bot lain bisa memiliki algoritma untuk menghindar dari serangan. Tackle juga sangat tidak efektif bila pergerakan player kecepatannya sama, satu -satunya cara hanyalah dengan memotong pergerakan lawan. Namun untuk memprediksi pergerakan lawan juga termasuk komputasi yang berat dan hasil yang didapatkan belum tentu benar.

Algoritma Score per Distance cukup mudah untuk dimengerti, dan tidak ada kelemahan yang besar. Maka dari itu kami memilih Algoritma Score per Distance.

# BAB 4

# Implementasi dan Pengujian

## 4.1 PseudoCode

| Program lowest\_time\_to\_diamond  {Algoritma mencari diamond tujuan berdasarkan nilai diamond dibagi dengan jarak yang ditempuh dari posisi bot} |
| --- |
| Kamus :  target : array of Position  current\_position : Position  visited\_diamonds : array of int  next\_diamond : GameObject  direction : array of Position  min\_paths : array of Position |
| Algoritma :  # Custom exception class  class TimeoutException(Exception)  pass  endclass  # Fungsi menghitung jarak ke diamond  function calculate\_path\_to\_diamond(board\_bot, board, diamonds):  target ← []  current\_position ← board\_bot.position  visited\_diamonds ← set()  # Loop selama inventory belum penuh  while len(target) < 5 - diamonds:  next\_diamond ← None  # Mencari diamond terdekat untuk semua diamond  # dilimit untuk mengurangi delay  start\_timer(min\_delay\_seconds)  try:  for diamond in board.diamonds:  if diamond.points + diamonds > 5 or diamond.id in visited\_diamonds:  continue  if next\_diamond is None or distance\_to(diamond, current\_position) < distance\_to(next\_diamond, current\_position):  next\_diamond ← diamond  if visited\_diamonds is None or next\_diamond is None:  break  visited\_diamonds.add(next\_diamond.id)  target.append(next\_diamond.position)  catch TimeoutException:  → target # Timeout break  finally:  stop\_timer()  → target  # Main logic class  class Main(BaseLogic):  public procedure \_\_init\_\_(self):  directions ← [(1, 0), (0, 1), (-1, 0), (0, -1)]  goal\_position ← None  current\_direction ← 0  endprocedure  # Kalkulasi gerakan selanjutnya  public function next\_move(board\_bot, board):  props ← board\_bot.properties    # Kalau inventory penuh , set tujuan ke base  if props.diamonds == 5:  base ← board\_bot.properties.base  goal\_position ← base  current\_position ← board\_bot.position  delta\_x, delta\_y ← get\_direction(current\_position, goal\_position)  # Kalau Inventory belum penuh  else:  # Mencari 5 - diamond di inventory  min\_paths ← calculate\_path\_to\_diamond(board\_bot, board, props.diamonds)  current\_position ← board\_bot.position  delta\_x, delta\_y ← get\_direction(current\_position, min\_paths[0])  → delta\_x, delta\_y  endfunction  # fungsi untuk mengecek apakah sisa waktu untuk ke base sudah minimum  function last\_deposit(time\_left, delay, bb):  base ← bb.properties.base  curr\_pos ← bb.position  distance\_to\_base ← abs(base.x - curr\_pos.x) + abs(base.y - curr\_pos.y)  time\_to\_base ← delay \* distance\_to\_base \* 10  if time\_left - time\_to\_base <= 1000:  → True  else:  → False  #fungsi untuk memprioritaskan tombol merah jika tidak ada diamond yang dekat  function red\_button\_priority(board, bb, x, y):  curr\_pos ← bb.position  dd ← [dd for dd in board.game\_objects if dd.type == "DiamondButtonGameObject"]  rb\_pos ← dd[0].position  distance\_to\_rb ← abs(curr\_pos.x - rb\_pos.x) + abs(curr\_pos.y - rb\_pos.y)  distance\_to\_dia ← abs(x - curr\_pos.x) + abs(y - curr\_pos.y)  if distance\_to\_rb < distance\_to\_dia:  → rb\_pos  → 0  # Fungsi mengecek apakah lebih cepat menggunakan teleporter untuk mencapai tujuan  function isTeleporterAlternatif(current\_x, current\_y, dest\_x, dest\_y, teleport\_position):  dest\_dist ← abs(dest\_x - current\_x) + abs(dest\_y - current\_y)  tele1\_dist ← abs(teleport\_position[0].position.x - current\_x) + abs(teleport\_position[0].position.y - current\_y) + abs(dest\_x - teleport\_position[1].position.x) + abs(dest\_y - teleport\_position[1].position.y)  tele2\_dist ← abs(teleport\_position[1].position.x - current\_x) + abs(teleport\_position[1].position.y - current\_y) + abs(dest\_x - teleport\_position[0].position.x) + abs(dest\_y - teleport\_position[0].position.y)  if dest\_dist < tele1\_dist and dest\_dist < tele2\_dist:  → False, 0  elif tele1\_dist < tele2\_dist:  → True, 1  else:  → True, 2  # Fungsi mengecek apakah ada teleporter di jalan yang ingin ditempuh  function isTeleporterInWay(current\_x, current\_y, dest\_x, dest\_y, teleport\_position):  if dest\_x > current\_x:  for i in range(current\_x, dest\_x + 1):  if (teleport\_position[0].position.x == i and teleport\_position[0].position.y == current\_y) or (teleport\_position[1].position.x == i and teleport\_position[1].position.y == current\_y):  → True  for i in range(current\_y, dest\_y + 1):  if (teleport\_position[0].position.y == i and teleport\_position[0].position.x == dest\_x) or (teleport\_position[1].position.y == i and teleport\_position[1].position.x == dest\_x):  → True  else:  for i in range(dest\_x, current\_x + 1):  if (teleport\_position[0].position.x == i and teleport\_position[0].position.y == current\_y) or (teleport\_position[1].position.x == i and teleport\_position[1].position.y == current\_y):  → True  for i in range(dest\_y, current\_y + 1):  if (teleport\_position[0].position.y == i and teleport\_position[0].position.x == dest\_x) or (teleport\_position[1].position.y == i and teleport\_position[1].position.x == dest\_x):  → True  # Fungsi mendapatkan arah dengan memfaktorkan teleporter  function get\_direction(current\_x, current\_y, dest\_x, dest\_y, board):  teleport\_position ← get\_teleport\_position(board)  isTeleport, teleport\_number ← isTeleporterAlternatif(current\_x, current\_y, dest\_x, dest\_y, teleport\_position)  delta\_x ← clamp(dest\_x - current\_x, -1, 1)  delta\_y ← clamp(dest\_y - current\_y, -1, 1)  if isTeleport:  if teleport\_number == 1:  delta\_x ← clamp(teleport\_position[0].position.x - current\_x, -1, 1)  delta\_y ← clamp(teleport\_position[0].position.y - current\_y, -1, 1)    else:  delta\_x ← clamp(teleport\_position[1].position.x - current\_x, -1, 1)  delta\_y ← clamp(teleport\_position[1].position.y - current\_y, -1, 1)  if delta\_x != 0:  delta\_y ← 0  elif delta\_x == 0 and delta\_y == 0:  rand\_choice ← random.choice([0, 1])  if rand\_choice == 0:  delta\_y ← random.choice([-1, 1])  else:  delta\_x ← random.choice([-1, 1])  → (delta\_x, delta\_y)  if not isTeleporterInWay(current\_x, current\_y, dest\_x, dest\_y, teleport\_position):  if delta\_x != 0:  delta\_y ← 0  elif delta\_x == 0 and delta\_y == 0:  rand\_choice ← random.choice([0, 1])  if rand\_choice == 0:  delta\_y ← random.choice([-1, 1])  else:  delta\_x ← random.choice([-1, 1])  → (delta\_x, delta\_y)  else:  rand\_choice ← random.randint(0, 2)  if rand\_choice == 0 and delta\_x != 0:  delta\_y ← 0  elif rand\_choice == 1 and delta\_y != 0:  delta\_x ← 0  else:  rand\_choice ← random.choice([0, 1])  if rand\_choice == 0:  delta\_y ← random.choice([-1, 1])  delta\_x ← 0  else:  delta\_x ← random.choice([-1, 1])  delta\_y ← 0  future\_positionX, future\_positionY ← current\_x + delta\_x, current\_y + delta\_y  while future\_positionX == teleport\_position[0].position.x and future\_positionY == teleport\_position[0].position.y or future\_positionX == teleport\_position[1].position.x and future\_positionY == teleport\_position[1].position.y:  if delta\_x == 0:  delta\_x ← random.choice([-1, 1])  delta\_y ← 0  else:  delta\_y ← random.choice([-1, 1])  delta\_x ← 0  future\_positionX, future\_positionY ← current\_x + delta\_x, current\_y + delta\_y  → (delta\_x, delta\_y)  # Fungsi mendapatkan posisi teleporter  function get\_teleport\_position(board):  → [d for d in board.game\_objects if d.type == "TeleportGameObject"] |

## 4.2 Hasi Uji Coba

Untuk uji coba akan dibandingkan algoritma **Score Per Distance** dengan **Shortest Distance.** Bot 1 adalah **Shortest Distance** dan bot 2 adalah **Score Per Distance.** Uji coba akan dilakukan sebanyak sepuluh kali. Lima kali dengan kecepatan sebesar 0.1*,* dan lima kali lagi akan sebesar 1 detik per langkah. Pada kolom deskripsi akan ditampilkan apa saja hal unik yang terjadi saat uji coba bot.

| No | Bot1 | Bot2 | Difference | sleep | Deskripsi |
| --- | --- | --- | --- | --- | --- |
| 1 | 90 | 101 | 11 | 0.1 | - |
| 2 | 85 | 93 | 8 | 0.1 | - |
| 3 | 80 | 105 | 25 | 0.1 | - |
| 4 | 75 | 121 | 46 | 0.1 | Bot SpD melakukan tackle berkali-kali |
| 5 | 93 | 116 | 23 | 0.1 | - |
| 6 | 5 | 15 | 10 | 1 | - |
| 7 | 10 | 5 | 5 | 1 | - |
| 8 | 10 | 10 | 0 | 1 | - |
| 9 | 5 | 14 | 9 | 1 | - |
| 10 | 5 | 11 | 6 | 1 | - |
| Total | 458 | 591 | 133 |  | |

## 4.3 Analisis

Dari hasil pengujian terdapat perbedaan skor yang sangat besar untuk algoritma **Score Per Distance (SpD)** dibandingkan **Shortest Distance (SD)**. Untuk kecepatan input 0.1 detik, SpD mendominasi dengan perbandingan skor yang jauh dan berhasil memenangkan semua percobaan. Bahkan ada sekali percobaan, Bot SpD melakukan tackle berulang kali pada bot lawannya dan berhasill mendapatkan keuntungan skor sebesar 46. Untuk kecepatan input 1 detik, perbedaan skor sudah semakin berkurang dan ada sekali bot SD berhasil mengimbangi bot SpD.

Pada pengujian terdapat beberapa faktor yang membuat skor semakin menjauh. Salah satunya adalah keberuntungan akan munculnya komponen pada board. Siapa yang paling banyak diamond disekitar rumahnya atau yang memiliki teleporter yang strategis untuk pulang pergi sangat lah mengubah kemungkinan menang suatu bot. Masing-masing algoritma tidak bekerja dengan baik jika diamond yang muncul jauh dari rumah atau basenya.

# Bab 5

# Kesimpulan dan Saran

## 5.1 Kesimpulan

Dari pembuatan algoritma greedy Score per Distance, penulis dapat menarik kesimpulan bahwa :

1. Algoritma Score per Distance (SpD) adalah algoritma yang memanfaatkan nilai diamond dan lokasi diamond untuk menemukan nilai per jarak dari bot sebagai logika utama untuk memilih tujuan selanjutnya.
2. Dari hasil percobaan, algoritma ini berhasil mengambil diamond dan membawanya ke base dengan cukup efektif dan efisien. Saat diuji berlawan dengan algoritma lain (Shortest Distance), algoritma SpD berhasil secara konsisten meraih nilai diamond yang lebih banyak.
3. Algoritma ini masih belum optimal dalam pemanfaatan teleporter dan red button.

## 5.2 Saran

Setelah pembuatan algoritma greedy Score per Distance, masih terdapat beberapa kekurangan. Adapun saran-saran sebagai berikut:

1. Diperlukan handler untuk menghindari musuh
2. Diperlukan handler untuk menimpa musuh apabila ada musuh yang dapat ditimpa

# Lampiran

Repository : [Github](https://github.com/Rapa285/Tubes1_gatau_hehe.git)

Gambar Tabel percobaan sebelum diberikan handler:

# 

# Daftar Pustaka

Munir, Rinaldi. “Algoritma Greedy.” *Informatika*, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf. Accessed 8 March 2024.

unila. “Apa itu Bot.” *Himatro Unila*, https://himatro.ee.unila.ac.id/apa-itu-bot/. Accessed 9 March 2024.