IMPLEMENTASI ALGORITMA GREEDY DALAM PEMECAHAN BOT PERMAINAN DIAMOND

Tugas Besar

Diajukan sebagai syarat menyelesaikan mata kuliah Strategi Algoritma (IF2211) Kelas RA
di Program Studi Teknik Informatika, Fakultas Teknologi Industri, Institut Teknologi Sumatera

Oleh: Kelompok Bottot

Muhammad Fajri Firdaus	123140050
Bayu Brigas Novaldi	123140072

Dosen Pengampu: Imam Ekowicaksono, S.Si., M.Si.

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA
2025

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Permainan Diamonds merupakan tantangan pemrograman yang di mana mengadu strategi antar bot buatan kelompok yang mengambil mata kuliah Strategi Algoritma. Setiap kelompok membuat sebuah bot dengan satu tujuan utama: yaitu mengumpulkan diamond sebanyak mungkin dan menyetorkannya ke base untuk mendapatkan poin. Tetapi yang menjadi tantangannya yaitu terdapat banyak rintangan seperti lawan yang bisa tackle dan mengambil diamond kita, diamond merah yang berisiko untuk tidak terambil ketika inventory penuh, serta teleporter dan tombol regenerasi diamond yang membuat game berubah secara acak.

Karena itu, setiap kelompok harus bisa merancang strategi yang baik untuk. Bot perlu bisa mengambil keputusan dengan cepat: kapan harus ambil diamond, kapan harus pulang, kapan harus menghindar, dan bahkan kapan harus memanfaatkan teleporter. Bot juga harus bisa menyesuaikan diri dengan kondisi permainan yang berubah-ubah secara acak.

Intinya, bot tidak hanya butuh kecepatan, tapi juga kecerdasan dan kemampuan beradaptasi di berbagai kondisi. Kreativitas dan pemahaman strategi algoritma juga sangat dibutuhkan agar bot bisa bertahan dan mendapat poin yang banyak di tengah persaingan yang ketat di dalam satu board.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma Greedy merupakan suatu algoritma yang memiliki metode pendekatan untuk membangun solusi optimal dengan mencari rute perjalanan terpendek, tetapi menghabiskan waktu sesingkat atau seminimum mungkin. Algoritma Greedy dapat menentukan jalur mana yang akan diambil terlebih dahulu atau dapat disebut juga dengan jalur optimum lokal, sehingga sampai seluruh jalur diambil pada akhir perjalanan dan menciptakan rute perjalanan terpendek atau disebut dengan optimum global.

Dalam konteks tugas besar ini, kami menerapkan algoritma greedy untuk mengatur aksi bot dalam permainan Diamonds. Tujuan utamanya adalah mengumpulkan poin sebanyak mungkin dengan cara yang paling efisien, berdasarkan informasi yang tersedia di saat itu. Bot tidak mencoba menghitung semua kemungkinan seperti pada algoritma pencarian menyeluruh, melainkan langsung bertindak berdasarkan situasi yang sedang dihadapi, mirip seperti kita mengambil keputusan cepat dalam kondisi mendesak.

2.2 Cara Kerja Program

Pada bagian ini, kami akan menjelaskan secara umum bagaimana program ini bekerja, termasuk dengan bagaimana bot melakukan aksi, apa algoritma diterapkan, dan bagaimana cara menjalankan bot tersebut. Berikut merupakan rincian yang dapat kami jadikan poin dalam deskripsi cara kerja program ini, antara lain:

1. **Cara Implementasi Program**
    - Deskripsi Umum: Cara kerja program ini adalah dengan mengimplementasikan algoritma tertentu dalam bot, seperti contoh pada tugas besar kali ini, kami menggunakan algoritma greedy untuk menentukan aksi yang diambil oleh bot.
    - Detail Proses: Bot dapat diatur untuk membuat keputusan berdasarkan strategi yang sudah kami buat, dengan mengambil langkah-langkah tertentu untuk mencapai tujuannya, seperti mengumpulkan diamond sampai dengan menghindari lawan. Implementasi program ini dilakukan dengan mendefinisikan logika yang

akan digunakan oleh bot, yang melibatkan pengambilan keputusan dalam game engine.

- Proses Pengaturan: Pengguna harus menyesuaikan kode dalam file bot sesuai dengan logika yang diinginkan (misalnya, logika acak, logika berbasis perhitungan, dll.).

2. Menjalankan Bot Program

- Menjalankan Bot: Untuk menjalankan bot ini, pengguna dapat memilih logika yang telah diimplementasikan pada bot dan menjalankannya dengan perintah yang telah disediakan (misalnya menggunakan Python di terminal). Setelah itu, bot akan mulai beraksi di dalam game.
- Kontrol dan Monitoring: Bot akan terus beroperasi sesuai dengan strategi yang diterapkan, dan pengguna dapat memonitor pergerakan bot melalui tampilan frontend yang telah disediakan.

3. Algoritma Bot Mengambil Keputusan

- Adapun fungsi yang paling inti dari permainan ini adalah fungsi move. Fungsi ini akan dipanggil secara berkala oleh game engine dan mengembalikan nilai gerakan berupa dx, dan dy.
- Dalam implementasian bot nya, pergerakan berdasarkan dengan posisi diamond terdekat, dan juga memperimbangkan jumlah diamond yang dibawa di inventory, waktu yang terisisa, dan apakah menggunakan teleport bisa lebih mempersingkat rute.

4. Interaksi Bot dan Engine

- Setiap langkah dari bot itu akan dikirimkan ke server melalui HTTP request ke endpoint /api/bots/{id}/move, dan status game diperbarui secara berkala melalui API. Bot hanya dapat bergerak satu langkah per tile nya, dengan arahnya juga yang terbatas, yaitu kanan, bawah, kiri, atas.

BAB III
APLIKASI STRATEGI GREEDY

3.1 Proses *Mapping*

Dalam permainan Bot Diamonds ini, algoritma greedy diterapkan utuk memilih aksi atau tindakan yang terbaik secara lokal di setiap langkah dalam permainannya, harapannya mencapai hasil global yang maksimal. Adapun proses mapping problem ke elemen algoritma greedy ini antara lain adalah:

- Himpunanan Kandidat: yaitu seluruh objek diamond yang ada di board permainan ini, baik diamond biru ataupun merah.
- Himpunana Solusi: yaitu diamond biru ataupun merah yang berhasil dikumpulkan dan disetor ke base.
- Fungsi Seleksi: yaitu diamond terdekat dari posisi sekarang, dengan preferensi terhadap diamond biru jika inventory sudah hampir penuh.
- Fungsi Kelayakan: yaitu pergerakan ke diamond yang masih bisa dijangkau dan tanpa resiko, nah resiko yang dimaksud adalah seperti menghindari diamond merah jika inventory sudah membawa empat diamond maupun lebih.
- Fungsi Objektif: yaitu memaksimalkan jumlah score yang didapat dalam waktu yang terbatas, dengan mempertimbangkan efisiensi penggunaan rute dan resiko gangguan bot lain, seperti tackle yang bisa mencuri diamond kita.

3.2 Eksplorasi Alternatif Solusi Greedy

Terdapat beberapa alternatif strategi greedy yang menjadi pertimbangan sebelum implementasi akhir, yaitu:

- Strategi Jarak Terdekat: Bot akan selalu menuju diamond terdekat tanpa mempertimbangkan jenis diamond itu merah atau biru, ataupun tanpa mempertimbangkan kondisi inventory itu full atau tidak.
- Strategi Prioritas Merah: Bot lebih memprioritaskan diamond merah karena diamond merah lah yang memiliki score paling banyak, yaitu 2 poin, walaupun sangat beresiko, karena diamond merah yang tidak banyak muncul seperti diamond biru, dan resiko lain seperti rute menjadi jauh sehingga akan sia-sia.

- Strategi Simpan Diamond Aman: Bot ini akan pulang ke base setiap kali dia menngumpulkan setidaknya 2 diamond biru, demi menghindari resiko yang dapat menyebabkan diamond tercuri seperti di tackle bot lain.

Dari semua alternatif solusi Greedy yang kami pertimbangkan, semua pendekatan ini memiliki kekurangannya, seperti kurang efisien dan juga terlalu menjaga atau defensif, sehingga lupa dengan tujuan permainan ini.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Nah, kami akan menganalisis semua alternatif yang kami ingin kembangkan sebelumnya, tetapi masih penuh akan pertimbangan:

- Strategi Jarak Terdekat: Ini sebenarnya dapat bekerja dengan cepat, tetapi akan sangat beresiko karena bot nya tidak mempertimbangkan atau mengabaikan inventory dan juga waktu.
- Strategi Prioritas Merah: ini juga sangat berpotensi untuk memberikan score yang tinggi, karena pada ketentuannya, diamond merah lah yang bisa memberikan kita point yang banyak, yaitu 2 poin, tetapi sangat tidak aman bila inventory kita penuh, rute yang dilalui akan terlalu jauh, dan juga bisa mendapat resiko terkena tackle oleh bot lain.
- Strategi Simpan Diamond Aman: Kami berfikir bot ini terlalu defensif, bot akan bermain aman karena terlalu sering kembali ke base, dan akan kehilangan potensi untuk mengumpulkan poin yang lebih tinggi dalam sekali jalan.

Efektifitas paling tinggi akan terlihat pada strategi yang telah dikombinasikan, dengan mempertimbangkan seluruh hal mulai dari posisi diamond, inventory, waktu dan juga penggunaan teleporter untuk menghemat langkah.

3.4 Strategi Greedy yang Dipilih

Dari alternatif strategi tadi, kami telah mendapatkan beberapa strategi Greedy yang telah kami pilih dan kami implementasikan dengan menggabungkan beberapa strategi dan yang telah kami sesuaikan, adapaun strategi Greedy yang kami pilih dan kami gabungkan antara lain:

- Pergerakan bot akan dilakukan dengan pola zig-zag, bergantian antara sumbu X, dan Y untuk menghindari tackle dari bot lain.
- Bot akan menghindari untuk mengambil diamond merah jika inventory sudah memiliki diamond sebanyakn 4 ataupun lebih.
- Bot akan memilih untuk bergerak menuju diamond terdekat ataupun pulang ke base, nah ini akan bergantung dari rute mana yang terdekat, misalnya, base lebih dekat untuk dikunjungi daripada diamond yang terdekat pada saat ini, maka bot akan pulang ke base untuk menyetor diamond nya terlebih dahulu. Ini juga bisa untuk menghindari resiko yang tidak diinginkan seperti menghindari tackle dari bot lain.
- Jika terdapat teleporter menuju ke diamond ataupun base, maka bot nya akan memanfaatkan teleporter tersebut untuk menghemat rute ataupun jarak sehingga bisa mendapatkan waktu yang maksimal.

Strategi di atas kami pilih karena kami pikir itu adalah strategi Greedy yang fleksibel, adaptif terhadap kondisi lingkungan yang berubah secara acakm dan juga telah terbukti efektif dalam simulasi bot yang telah kami lakukan berkali-kali.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Pseudocode

```
# Muhammad Fajri Firdaus (123140050 )
# Bayu Brigas Novaldi (123140072)
# Kelompok : BOTTOT

from __future__ import annotations

import random
from typing import List, Optional, Tuple

from game.logic.base import BaseLogic
from game.models import Board, GameObject, Position
from ..util import get_direction


class GreedyLogic(BaseLogic):
    # Bot greedy yang sudah ditingkatkan dengan fitur:
    # - mengenali teleporter,
    # - gerakan zig-zag,
    # - menghindari diamond merah saat inventory tinggi (>= 4),
    # - pulang ke base jika sudah cukup diamond (>= 5),
    # - berada dekat base
    # - dan kesadaran waktu (pulang ke base jika waktu hampir habis).

    def __init__(self) -> None:
        # Urutan arah: Kanan, Bawah, Kiri, Atas
        self.directions: List[Tuple[int, int]] = [(1, 0), (0, 1), (-1, 0), (0,
-1)]
        self.goal_position: Optional[Position] = None
        self.axis_toggle: bool = False  # Untuk zig-zag: False = prioritas X,
True = prioritas Y

    #  Fungsi Utilitas
    @staticmethod
    def _manhattan(a: Position, b: Position) -> int:
        # Menghitung jarak Manhattan (tanpa diagonal)
        # jalan zig-zag tanpa perpindahan diagonal
        return abs(a.x - b.x) + abs(a.y - b.y)

    @staticmethod
    def _is_red_diamond(obj: GameObject) -> bool:
```

```
        # Mengecek apakah objek adalah diamond merah (poin = 2 → dikurangi
saat disetor)
        return obj.type == "DiamondGameObject" and getattr(obj.properties,
"points", 1) == 2

    # fungsi untuk mendeteksi bahaya
    # jika inventory >= 4, hindari diamond merah
    def _is_dangerous(self, board: Board, pos: Position) -> bool:
        # Menentukan apakah posisi tersebut mengandung diamond merah
        for obj in board.game_objects:
            if obj.position == pos and self._is_red_diamond(obj):
                return True
        return False

    @staticmethod
    def _other_teleporter(entry: GameObject, teleporters: List[GameObject]) ->
Optional[GameObject]:
        # Mendapatkan teleporter pasangannya (bukan diri sendiri)
        for tp in teleporters:
            if tp.id != entry.id:
                return tp
        return None

    def _best_teleporter_entry(
        self,
        current: Position,
        dest: Position,
        teleporters: List[GameObject],
    ) -> Tuple[Optional[Position], int]:

        # Menentukan pintu masuk teleporter terbaik untuk mendekati tujuan,
        # dibandingkan dengan jarak langsung. Jika tidak lebih baik, tidak
menggunakan teleporter.
        direct_distance = self._manhattan(current, dest)
        if len(teleporters) < 2:
            return None, direct_distance

        best_entry: Optional[Position] = None
        best_total: int = direct_distance

        for entry in teleporters:
            exit_ = self._other_teleporter(entry, teleporters)
            if not exit_ or entry.position == current:
                continue
            total = self._manhattan(current, entry.position) +
self._manhattan(exit_.position, dest)
            if total < best_total:
                best_total = total
```

```
                best_entry = entry.position

        return best_entry, best_total

    # Logika Utama Pergerakan
    def next_move(self, bot: GameObject, board: Board) -> Tuple[int, int]:
        cur = bot.position
        props = bot.properties
        diamond_count: int = props.diamonds or 0
        base: Optional[Position] = props.base

        # Konversi waktu tersisa dari milidetik ke detik
        milliseconds_left: int = props.milliseconds_left or 0
        seconds_left: int = milliseconds_left // 1000

        # Hitung jarak ke base (digunakan untuk pulang atau perbandingan)
        d_base: int = self._manhattan(cur, base) if base else float("inf")

        # Menentukan Tujuan
        # Ambil diamond terdekat, hindari diamond merah jika inventory >= 4
        diamonds: List[GameObject] = [
            obj
            for obj in board.game_objects
            if obj.type == "DiamondGameObject" and not (diamond_count >= 4 and
self._is_red_diamond(obj))
        ]
        diamonds.sort(key=lambda d: self._manhattan(cur, d.position))
        nearest_diamond: Optional[GameObject] = diamonds[0] if diamonds else
None
        d_diamond: int = (
            self._manhattan(cur, nearest_diamond.position) if nearest_diamond
else float("inf")
        )

        # Aturan utama penentuan tujuan:
        if diamond_count >= 5:
            dest: Optional[Position] = base  # Sudah cukup diamond, pulang
        elif diamond_count == 0:
            dest = nearest_diamond.position if nearest_diamond else base  #
Belum punya diamond, cari
        else:
            dest = nearest_diamond.position if nearest_diamond and d_diamond <
d_base else base  # Pilih tergantung jarak

        # Aturan waktu kritis
        # Jika waktu tersisa tidak cukup untuk pulang, langsung pulang
        if base and seconds_left <= d_base + 1:
            dest = base
```

```
        # Jika tidak ada tujuan (misalnya tidak ada base), gerak acak aman
        if dest is None:
            return self._random_safe_move(board, cur, diamond_count)

        # Optimasi dengan Teleporter
        teleporters: List[GameObject] = [o for o in board.game_objects if
o.type == "TeleportGameObject"]
        tele_entry, _ = self._best_teleporter_entry(cur, dest, teleporters)
        self.goal_position = tele_entry if tele_entry else dest

        # Zig-zag Menuju Tujuan
        dx_total: int = self.goal_position.x - cur.x
        dy_total: int = self.goal_position.y - cur.y
        step_x: int = 0 if dx_total == 0 else (1 if dx_total > 0 else -1)
        step_y: int = 0 if dy_total == 0 else (1 if dy_total > 0 else -1)

        candidate_moves: List[Tuple[int, int]] = []
        if self.axis_toggle:  # Y dulu baru X
            if step_y:
                candidate_moves.append((0, step_y))
            if step_x:
                candidate_moves.append((step_x, 0))
        else:  # X dulu baru Y
            if step_x:
                candidate_moves.append((step_x, 0))
            if step_y:
                candidate_moves.append((0, step_y))

        # Coba jalankan langkah zig-zag yang valid dan aman
        for dx, dy in candidate_moves:
            new_pos = Position(cur.x + dx, cur.y + dy)
            if board.is_valid_move(cur, dx, dy) and not (
                diamond_count >= 4 and self._is_dangerous(board, new_pos)
            ):
                self.axis_toggle = not self.axis_toggle
                return dx, dy

        # Alternatif fallback: arah langsung ke tujuan
        dx, dy = get_direction(cur.x, cur.y, self.goal_position.x,
self.goal_position.y)
        new_pos = Position(cur.x + dx, cur.y + dy)
        if board.is_valid_move(cur, dx, dy) and not (
            diamond_count >= 4 and self._is_dangerous(board, new_pos)
        ):
            self.axis_toggle = not self.axis_toggle
            return dx, dy
```

```
        # Jika sedang di base, setor diamond
        if dx == 0 and dy == 0 and base and cur == base and diamond_count > 0:
            props.diamonds = 0
            return 0, 0

        # Langkah terakhir: gerakan aman acak
        return self._random_safe_move(board, cur, diamond_count)

    # Gerakan acak aman
    def _random_safe_move(self, board: Board, pos: Position, diamond_count:
int) -> Tuple[int, int]:
        valid: List[Tuple[int, int]] = []
        for dx, dy in self.directions:
            new_pos = Position(pos.x + dx, pos.y + dy)
            if board.is_valid_move(pos, dx, dy):
                if diamond_count >= 4 and self._is_dangerous(board, new_pos):
                    continue  # Hindari diamond merah saat inventory tinggi
                valid.append((dx, dy))
        return random.choice(valid) if valid else (0, 0)
```

2. **Penjelasan Alur Program**

Dari program Bottot yang telah kami buat di atas kami menggunakan algoritma greedy yang di tingkatkan dengan fitur tambahan

- **Mengenali teleporter**

  Teleporter akan di cek, apakah dengan menggunakan teleporter akan mempersingkat jalan ke target tujuan, jika mempersingkat jalan maka bot bottot akan melewati teleporter.

  Sebaliknya jika malah lebih jauh lewat teleporter maka bottot tidak akan menggunakan teleporter teleport

- **Gerakan zig-zag**

  Bottot akan melakukan perjalanan dengan zigzag menuju target tanpa memperpanjang jarak, juga akan di cek dengan fungsi _manhattan dimana bot tidak diperbolehkan melakukan perjalanan secara diagonal. Dimana pada logika zigzag ini bisa sumbu-Y dulu baru sumbu-X atau sebaliknya

- **Menghindari Diamond Merah dengan point 2**

  Dimana ketika inventory telah 4 biasanya bot saat mengambil diamond merah maka akan terjadi stuck dikarenakan diamond merah memiliki point 2 sedangkan inventory max nya itu 5. Sehingga disini kami bentuk logika dimana jika point

diamond yang ada di inventory sudah 4 atau lebih maka akan mengabaikan diamond merah bahkan akan menghindar

- **Pulang jika inventory sudah penuh**
  logika ini sebenernya logika(fitur) dasar yang ada di game ini karena jika tidak balik ke base maka akan sia sia karena inventory sudah full tidak akan bisa mengambil diamond lagi dan juga score kita juga tidak akan bertambah
- **Berada dekat base**
  Bottot melakukan pengecekan apakah lebih dekat ke rumah daripada diamond maka bottot akan pulang terlebih dahulu dengan syarat bottot memiliki setidaknya 1 diamond di inventory, jika bottot tidak memiliki diamond sama sekali di inventory maka bottot tidak diperbolehkan pulang, akan di suruh untuk mencari diamond terlebih dahulu
- **Menyisakan waktu untuk pulang**
  Dimana 1 langkah itu akan memakan waktu 1 detik (delay 1 detik) maka kami menambahkan fitur pada bottot dimana ada pengecekan waktu ketika waktu untuk pulang = waktu sekarang maka bottot diharuskan balik ke base, dikarenakan untuk apa jika waktu sudah mau habis tetapi di inventory bottot ada 1-4 maka lebih baik pulang dan dapet point daripada mencari diamond ke 5 tetapi tidak bisa pulang dengan tepat waktu

4.2 Struktur Data yang Digunakan

Adapun struktur data yang kami gunakan dalam projek kali ini adalah:

- Position: Digunakan untuk merepresentasikan koordinat (x, y) bot dan objek lainnya di board.
- List[GameObject]: Digunakan untuk menyimpan semua objek permainan (diamond, teleporter, dan semua objek game yang dibutuhkan).
- Tuple[int, int]: Digunakan untuk merepresentasikan arah pergerakan (dx, dy)

4.3 Pengujian Program

1. **Skenario Pengujian**
    - Bot akan ditempatkan di dekat diamond merah dan biru dengan inventory kosong.
    - Bot memiliki inventory sebanyak 4 poin dan terdapat diamond merah di dekatnya.
    - Waktu permainan yang tersisa sedikit dan kondisi posisi base yang sangat jauh dari posisi saat ini.
    - Dua teleporter yang akan dimanfaatkan untuk memotong rute.
    - Home lebih dekat dari pada diamond dan setidaknya memiliki 1 diamond

2. **Hasil Pengujian dan Analisis**
    - Bot akan bisa mengambil diamond dan menyetor diamond secara optimal dengan strategi greedy ini.
    - Bot akan menghindari/mengabaikan diamond merah yang berada di dekatnya saat inventory sudah mencapai 4 point ataupun lebih.
    - Bot akan pulang ke base jika waktu tersisa sedikit lagi.
    - Bot memanfaatkan teleporter yang ada di board, jika teleporter itu lebih efisien daripada rute langsung.
    - Bot akan pulang ke base jika base lebih dekat daripada diamond, dan bot memiliki setidaknya 1 diamond, jika bot belum memiliki diamond, maka bot tetap mencari diamond di posisi terdekatnya.

Pengujian ini dilakukan dengan menambahkan log print dan kami monitoring melalui visualisasi frontend yang sudah disediakan, nah ini menunjukkan bahwa seluruh keputusan bot sudah sesuai dengan strategi greedy yang kami rancang.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dalam tugas besar ini, kami berhasil untuk mengimplementasikan strategi greedy pada bot Permainan Diamonds dengan mempertimbangkan berbagai kondisi permainan secara dinamis, seperti jumlah diamond, jarak ke base, waktu yang tersisa, dan posisi musuh. Strategi yang sudah kami pilih itu terbukti efektif dalam menghindari resiko yang akan mungkin terjadi, memaksimalkan score, dan mampu untuk beradaptasi ketika situasi permainan berubah. Penggunaan zig-zag, dan pemanfaatan teleporter yang tersedia juga memberikan keunggulan dalam efisiensi pergerakan yang sia-sia. Secara keseluruhan, bot kami sudah mampu untuk bersaing dengan baik dalam simulasi yang telah dilakukan.

5.2 Saran

Untuk pengembangan ke depannya, strategi greedy ini masih dapat dikembangkan dan ditingkatkan dengan pendekatan lainnya, seperti mengombinasikan algoritma greedy dengan algoritma lainnya, seperti A*, terutama untuk mempertimbangkan gerakan lawan. Selain itu, pengelolaan resiko tackle dari bot lain juga masih bisa dapat dikembangkan dengan deteksi posisi lawan menggunakan probabilitas keberhasilan.

LAMPIRAN

A. Repository Github (Link Repository Github)

B. Video Penjelasan (Link GDrive)

DAFTAR PUSTAKA

A. Yohanes, E. N. Hayati and. *Pencarian Rute Terpendek Menggunakan Algoritma Greedy*. 2014.

R. Toyib, Y. Darnita and. *Penerapan Algoritma Greedy Dalam Pencarian Jalur Terpendek Pada Instansi-Instansi Penting Di Kota Argamakmur Kabupaten Bengkulu Utara*. vol. vol. 15, 2019. no. 2 vols.

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

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