IMPLEMENTASI ALGORITMA GREEDY DALAM PEMECAHAN BOT PERMAINAN DIAMOND

Tugas Besar

Diajukan sebagai syarat menyelesaikan mata kuliah Strategi Algoritma (IF2211) Kelas RB

di Program Studi Teknik Informatika, Fakultas Teknologi Industri, Institut Teknologi Sumatera

Oleh: Kelompok BOT

Nahli Saud Ramdani	123140049
Ray Regan Sitepu	123140047
Gian Ivander	123140040

Dosen Pengampu: Imam Eko Wicaksono, S.Si., M.Si.

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INDUSTRI

INSTITUT TEKNOLOGI SUMATERA

2025

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Merupakan suatu programing challenge yang akan mempertandingkan bot antar pemain dengan nama BOTTLE ROYALE. Setiap pemain memiliki bot dengan tujuan mengumpulkan diamond. Tugas ini bertujuan menguji kemampuan berpikir strategis dan penerapan algoritma secara nyata.

Didalam permainan ini, setiap bot ditempatkan dalam sebuah papan permainan atau board yang dipenui oleh objek-objek seperti diamond, teleporter, red button, dan bot dari tim lain sebagai lawan. Bot memiliki inventory terbatas sehingga bot perlu menyimpan diamond tersebut ke dalam markas. Selain rintangan yang harus dihadapi oleh bot, bot dapat di tackle oleh bot lawan dan apabila bot terkena tackle, diamond yang telah dikumpulkan direbut oleh bot lawan tersebut dan bot akan hidup kembali di markasnya.

Sehingga yang diuji adalah kreatifitas memetakan kondisi kompleks dalam permainan ini dan merancang logika yang cerdas tapi cepat, taktis tapi efisien, dengan menggunakan pendekatan algoritma greedy.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma greedy adalah pendekatan algoritma yang membuat keputusan berdasarkan pilihan terbaik pada saat itu, tanpa mempertimbangkan dampak jangka panjang dari keputusan tersebut. Strategi ini umumnya digunakan dalam situasi yang membutuhkan efisiensi tinggi dan waktu komputasi yang terbatas, karena implemenatasi sederhana dan kecepatan eksekusi yang relatif tinggi. Agar menghasilkan solusi optimal, terdapat dua syarat utama yaitu greedy choice property dan optimal substructure. Namun tidak semua permasalahan memenuhi kedua syarat tersebut, sehingga algoritma greedy hanya menghasilkan solusi mendekati optimal[1].

2.2 Cara Kerja Program

Dalam permainan ini yang menjadi fokus tugas besar strategi algoritma, algoritma greedy dapat diterapkan untuk memilih diamond dengan rasio terbaik antara nilai dan jarak tempuh. Selain itu, juga memutuskan kapan harus kembali ke markas untuk menyimpan poin dan kapan harus menghindari bot lawan. Mengingat adanya waktu yang terbatas dalam mengambil keputusan dan keterbatasan penyimpanan, menjadikan pendekatan greedy sangat relevan untuk menghasilkan solusi yang responsif dan adaptif.

2.2.1. Cara Implementasi Program

Dengan melakukan pemetaan terhadap papan permainan terlebih dahulu, lalu menjalankan loop pencarian keputusan menggunakan algoritma greedy. Setiap langkah, bot akan mengevaluasi seluruh kemungkinan aksi, memilih aksi yang nilainya paling optimal.

2.2.2. Menjalankan Bot Program

Bot dijalankan dengan membaca input dari permainan, seperti posisi diamond, rintangan dan bot lawan. Kemudian bot akan memproses informasi tersebut

menggunakan algoritma greedy untuk menentukan aksi selanjutnya. Aksi yang dipilih akan dikirim kembali ke permainan untuk dieksekusi.

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Dalam pengembangan bot ini, kami melakukan proses pemetaan dengan memastikan setiap aspek. Tidak hanya memperhatikan elemen-elemen permainan, tetapi juga memperhatikan dinamika dan interaksi kompleks di antara mereka. Di dalam permainan terdapat diamond merah yang memiliki poin lebih dibandingkan diamond biru, tetapi berada di lokasi tertentu dengan waktu kemunculan yang lebih lama.

Selain perbedaan poin diamond, rintangan dalam permainan menjadikannya kesulitan sendiri. Walaupun begitu rintangan sebenarnya dapat dimanfaatkan sebagai pelindung dari lawan. Untuk lawan, bisa menjadi penghalang dengan algoritma dari tim lawan sendiri, dengan begitu diperlukannya mekanisme yang akan memprediksi gerakannya dan dilakukannya antisipasi terhadap hal tersebut.

3.2 Eksplorasi Alternatif Solusi Greedy

Selama pengembangan kami menemukan beberapa strategi algoritma greedy yang bisa digunakan untuk pengembangan bot ini. Masing-masing strategi memiliki kelebihan dan kekurangannya, diperlukannya penentuan untuk memilih strategi yang akan digunakan kepada bot yang sedang dikembangkan tersebut. Strategi algoritma greedy yang ditemukan adalah sebagai berikut:

3.2.1. Greedy Berdasarkan Jarak Terdekat Dengan Item

Bot memilih jalan menuju item yang memiliki jarak terdekat dari posisi bot. Strategi ini memiliki kemiripan dengan algoritma BFS atau Manhattan Distance sebagai estimator jarak bot dengan item.

3.2.2. Greedy Berdasarkan Poin Tertinggi

Strategi ini tidak hanya mempertimbangkan jarak bot dengan item, tetapi mempertimbangkan poin yang didapatkan dari item atau menargetkan diamond merah. Sehingga, apabila terdapat satu diamond merah dan satu diamond biru yang berada di sekitar bot, maka bot memilih diamond merah yang memiliki poin lebih tinggi.

3.2.3. Greedy Dengan Penalty Resiko

Selain item dan jarak, keberadaan lawan juga perlu diperhitungkan. Ketika melakukan tackle terhadap lawan, kita mendapatkan diamond yang telah dikumpulkannya. Strategi ini juga dapat memperhitungkan resiko kedekatan dengan lawan, sehingga bot dapat menghindari area berbahaya.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Pada pengembangan bot ini telah ditemukannya strategi yang bisa digunakan, masing-masing strategi memiliki kelebihan dan kekurangannya. Kami telah melakukan percobaan dengan strategi yang ada dalam pengembangan, berikut penilaian strateginya:

3.3.1. Berdasarkan Jarak Terdekat

Memiliki waktu eksekusi yang cepat dikarenakan hanya memperhitungkan jarak bot dengan item, tetapi bot cenderung berada di posisi berbahaya. Hal ini disebabkan, bot yang tidak mempertimbangkan bahaya disekitar item sehingga poin yang didapatkan sedikit.

3.3.2. Berdasarkan Poin Tertinggi

Strategi ini memiliki kemungkinan tinggi menghasilkan skor yang tinggi, karena memilih poin dari item terlebih dahulu. Tetapi bot terkadang terlalu agresif sehingga mengambil resiko dengan begitu bot tetap bisa kalah.

3.3.3. Berdasarkan Penalty Resiko

Strategi ini merupakan strategi yang paling seimbang dibandingkan strategi sebelumnya. Meskipun strategi ini membutuhkan waktu yang sedikit lebih lama yang disebabkan memperhitungkan resiko keberadaan lawan. Sehingga hasil yang didapatkan dari strategi ini adalah yang paling aman mengumpulkan item dan skor yang didapatkan tetap tinggi.

3.4 Strategi Greedy Yang Dipilih

Strategi yang kami gunakan terhadap pengembangan bot dengan aplikasi algoritma greedy adalah gabungan dari jarak terdekat dan penalty resiko. Bot yang telah kami kembangkan akan memilih langkah yang optimal dengan jarak diamond terdekat, jika sudah mengumpulkan 5 diamond bot akan langsung kembali ke markas tanpa mengambil diamond lain. Kami juga menggabungkan strategi penalty resiko yang apabila lawan berada disekitar bot, bot akan mengutamakan menghindar dengan menghitung arah berlawanan dari lawan.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

4.1.1. Pseudocode

```
from typing import Optional
import math
from game.logic.base import BaseLogic
from game.models import GameObject, Board, Position
from ..util import get_direction


class GreedLogic(BaseLogic):
    def __init__(self):
        self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
        self.goal_position: Optional[Position] = None
        self.current_direction = 0

    def get_nearest_diamond(self, pos: Position, diamonds:
list[Position]) -> Optional[Position]:
        """Mencari diamond terdekat dari posisi bot"""
        if not diamonds:
            return None

        min_dist = float('inf')
        nearest = None

        for diamond in diamonds:
            dist = math.sqrt((pos.x - diamond.x)**2 + (pos.y -
diamond.y)**2)
            if dist < min_dist:
                min_dist = dist
                nearest = diamond

        return nearest

    def get_nearest_enemy(self, pos: Position, enemies: list[Position])
-> Optional[Position]:
        """Mencari lawan terdekat dari posisi bot"""
```

```
        if not enemies:
            return None

        min_dist = float('inf')
        nearest = None

        for enemy in enemies:
            dist = math.sqrt((pos.x - enemy.x)**2 + (pos.y -
enemy.y)**2)
            if dist < min_dist:
                min_dist = dist
                nearest = enemy

        return nearest

    def get_safe_direction(self, pos: Position, enemy_pos: Position) ->
tuple[int, int]:
        """Menentukan arah aman untuk menghindari lawan"""
        dx = pos.x - enemy_pos.x
        dy = pos.y - enemy_pos.y

        if abs(dx) > abs(dy):
            return (-1 if dx < 0 else 1, 0)
        else:
            return (0, -1 if dy < 0 else 1)

    def next_move(self, board_bot: GameObject, board: Board):
        props = board_bot.properties
        current_position = board_bot.position

        # Jika sudah 5 diamond, kembali ke markas
        if props.diamonds >= 5:
            self.goal_position = props.base
        else:
            # Cek lawan terdekat
            nearest_enemy = self.get_nearest_enemy(current_position,
[bot.position for bot in board.bots if bot.id != board_bot.id])
            if nearest_enemy:
                enemy_dist = math.sqrt((current_position.x -
nearest_enemy.x)**2 + (current_position.y - nearest_enemy.y)**2)
                # Jika lawan terlalu dekat (dalam radius 3), hindari
                if enemy_dist < 3:
                    delta_x, delta_y =
```

```
self.get_safe_direction(current_position, nearest_enemy)
                    if board.is_valid_move(current_position, delta_x,
delta_y):
                        return delta_x, delta_y

            # Cari diamond terdekat
            nearest_diamond =
self.get_nearest_diamond(current_position, [d.position for d in
board.diamonds])
            if nearest_diamond:
                self.goal_position = nearest_diamond
            else:
                # Jika tidak ada diamond, kembali ke markas
                self.goal_position = props.base

        delta_x, delta_y = (0, 0) # Default ke tidak bergerak
        if self.goal_position:
            # Menghitung arah menuju goal
            delta_x, delta_y = get_direction(
                current_position.x,
                current_position.y,
                self.goal_position.x,
                self.goal_position.y,
            )

        # Periksa apakah langkah menuju goal valid, jika tidak, cari
arah lain
        if not board.is_valid_move(current_position, delta_x, delta_y):
            # Coba arah lain
            for dx, dy in self.directions:
                if board.is_valid_move(current_position, dx, dy):
                    return dx, dy # Temukan arah valid pertama dan
gunakan
            # Jika tidak ada arah yang valid (terjebak), tetap di
tempat
            return (0, 0)
        else:
            # Langkah menuju goal valid
            return delta_x, delta_y
```

4.1.2. Penjelasan Alur Program

Program akan mengecek jumlah diamond yang dibawa bot terlebih dahulu, jika sudah mencapai 5 diamond, tujuan bot akan langsung berubah untuk kembali ke markas. Jika belum, program akan mencari diamond terdekat dengan memperhitungkan jarak. Prioritas utama adalah mengumpulkan diamond secepat mungkin tanpa mempertimbangkan poin dari diamond yang didapatkan.

Sebelum bergerak menuju target, program akan memeriksa keberadaan lawan dalam radius 3 unit. Jika lawan terdeteksi terlalu dekat, program akan mengabaikan sementara waktu tujuan mengumpulkan diamond dan beralih ke mode penghindaran. Bot akan bergerak menjauh secara horizontal atau vertikal tergantung posisi relatif terhadap lawan. Mekanisme ini bersifat reaktif dan hanya aktif saat lawan sudah sangat dekat.

Setelah menentukan target dan memastikan area aman, bot akan mencoba bergerak menuju tujuan. Jika jalur terhalang, bot akan mencoba mencari arah alternatif . Ketika tidak tidak ada diamond tersisa, bot akan berusaha kembali ke markas. Namun, program tidak memiliki strategi khusus untuk menangani situasi terjebak.

4.2 Struktur Data yang Digunakan

Struktur Data	Fungsi	Contoh Penggunaan
Position (Class)	Menyimpan Koordinat (x, y)	Board_bot.position, diamond.position
List [GameObject]	Daftar diamond dan bot di board	Board,.diamonds, board.bots
Tuple[int, int]	Arah gerakan delta x, delta y)	(1, 0) = kanan, (0, -1) bawah
Optional [Position]	Target bisa “None” jika tidak ada diamond	self.goal_position

4.3 Pengujian Program

Program diuji melalui berbagai skenario kritis untuk memverifikasi algoritma pergerakan dan pengambilan keputusannya. Pengujian difokuskan pada tiga aspek utama: efisiensi pengumpulan diamond, respons terhadap ancaman lawan, dan penanganan kondisi terjebak.

4.3.1. Skenario Pengujian

Program diuji melalui lima skenario kritis untuk mengevaluasi efektivitas algoritmanya. Pertama, skenario Diamond Terdekat menguji kemampuan bot memilih target optimal dari beberapa diamond yang tersedia. Kedua, skenario Kembali ke Base memverifikasi logika penyelesaian misi saat bot telah mengumpulkan 5 diamond. Ketiga, skenario Menghindari Musuh mengamati respons bot terhadap ancaman dalam radius 3 unit. Keempat, Diamond Terhalang menguji adaptabilitas navigasi saat jalur terblokir. Terakhir, skenario Tidak Ada Diamond mengevaluasi perilaku bot saat tidak ada target yang bisa dicapai.

4.3.2. Hasil Pengujian dan Analisis

Pada skenario Diamond Terdekat, bot berhasil memilih diamond dengan jarak terpendek secara konsisten, membuktikan efektivitas pendekatan greedy. Dalam Kembali ke Base, bot langsung mengarah ke homebase setelah mencapai 5 diamond tanpa terganggu diamond lain. Namun di skenario Menghindari Musuh, bot hanya bereaksi saat musuh sangat dekat (<3 unit) tanpa strategi antisipasi jangka panjang. Ketika menghadapi Diamond Terhalang, bot mampu mencari alternatif rute sederhana, tetapi sering terjebak dalam pola gerakan berulang. Yang paling kritis adalah hasil skenario Tidak Ada Diamond, dimana bot menjadi pasif dan tidak memiliki mekanisme penyelamatan diri saat tidak ada target yang terjangkau.

Temuan utama menunjukkan algoritma unggul dalam situasi sederhana namun rentan dalam kondisi kompleks. Keberhasilan di skenario dasar (pemilihan diamond dan kembali ke base) membuktikan logika inti yang solid. Namun, kegagalan dalam menghadapi halangan kompleks dan ketiadaan target mengungkap kebutuhan mendesak untuk pengembangan mekanisme pathfinding yang lebih cerdas dan sistem pengambilan keputusan berbasis multi-faktor. Hasil ini menyoroti pentingnya menyeimbangkan efisiensi lokal dengan pertimbangan strategis global.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Program ini memiliki beberapa kelebihan utama, terutama dalam kesederhanaan dan kecepatan pengambilan keputusan. Dengan menggunakan pendekatan greedy berbasis jarak terdekat, program mampu merespons target secara cepat dan efisien pada peta sederhana. Mekanisme penghindaran lawan yang reaktif juga memberikan perlindungan dasar saat ancaman berada dalam jarak sangat dekat (<3 unit). Selain itu, prioritas untuk segera kembali ke markas setelah mengumpulkan 5 diamond menunjukkan logika penyelesaian misi yang tepat.

Namun, program ini memiliki beberapa keterbatasan signifikan. Strategi greedy murni tidak mempertimbangkan optimasi jangka panjang, seperti mengumpulkan diamond bernilai tinggi atau merencanakan rute terbaik menuju markas. Penghindaran lawan yang bersifat reaktif dan hanya aktif pada jarak sangat dekat membuat bot rentan terhadap lawan yang bergerak strategis. Masalah utama muncul ketika bot terjebak dalam situasi tanpa diamond yang dapat dijangkau atau markas yang terblokir, karena program tidak memiliki mekanisme recovery yang memadai. Selain itu, semua diamond diperlakukan sama tanpa mempertimbangkan perbedaan nilai (merah vs biru), yang mengurangi efisiensi pengumpulan poin.

5.2 Saran

Berdasarkan hasil pengujian, program membutuhkan penyempurnaan strategi navigasi dan pengambilan keputusan. Pertama, implementasi algoritma pathfinding seperti A* atau Dijkstra akan membantu bot merencanakan rute optimal dengan mempertimbangkan jarak, nilai diamond, dan lokasi base sekaligus. Kedua, sistem penghindaran musuh perlu ditingkatkan dengan menambahkan zona bahaya (danger zone) yang lebih luas dan kemampuan prediksi pergerakan musuh. Ketiga, penting untuk mengembangkan mekanisme penanganan deadlock, seperti pola gerakan acak terarah atau prioritas penghancuran halangan ketika terjebak.

Selain itu, program akan jauh lebih efektif dengan menerapkan sistem penilaian diamond berbobot yang memprioritaskan diamond merah (3 poin) meskipun jaraknya lebih jauh. Integrasi sistem pemantauan sumber daya (resource monitoring) juga diperlukan untuk menghindari situasi tanpa diamond yang terjangkau. Terakhir, penambahan logika "pulang paksa" ketika tidak ada diamond dalam radius tertentu akan mencegah bot berdiam diri terlalu lama. Dengan penyempurnaan ini, program tidak hanya akan lebih tangguh dalam berbagai skenario, tetapi juga mampu mencapai skor lebih tinggi secara konsisten.

LAMPIRAN

A. Repository Github

B. Video Penjelasan

https://youtu.be/uV_kzADOO8w

DAFTAR PUSTAKA

[1]

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to algorithms. Cambridge, Massachusett: The Mit Press, 2022.