LAPORAN AKHIR PROJECT ALGORITMA

Disusun oleh:

Ghora Laziola	2001020025
Muhammad Fadhly Azuhri	2001020022
Muhammad Rizky Amanullah	2001020059

Dosen Pengampu :

Tekad Matulatan, S.Sos., S.Kom., M.Inf.Tech

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNIK DAN TEKNOLOGI KEMARITIMAN

2023

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

A. Latar Belakang

Project ini mirip seperti game Hide and Seek dimana ada yang bertugas untuk mencari dan ada yang bertugas untuk menghindar. Game ini terdiri dari droid merah yang bertugas untuk mencari droid hijau, dan droid hijau bertugas untuk menghindar dari droid merah. Droid merah memiliki kemampuan untuk mengetahui secara lengkap kondisi peta tetapi tidak bisa melihat atau menembusi tembok sehingga droid merah tidak bisa mengetahui posisi droid hijau jika terhalang oleh tembok dan jika tidak ada halangan maka droid merah akan mengejar droid hijau. Droid hijau memiliki kemampuan untuk mengetahui posisi droid merah akan tetapi droid hijau rabun jauh sehingga tidak mengetahui kondisi peta. Jarak pandang droid hijau sangat pendek sehingga droid hijau baru menyadari adanya tembok setelah berhadapan dengan tembok. Droid hijau akan berusaha menghindar dari droid merah. Kegiatan ini akan berhenti ketika droid merah bersentuhan dengan droid hijau.

B. Rumusan Masalah

1. Algoritma apa yang digunakan untuk membuat project game droid?
2. Jelaskan analisis asymptotic dari algoritma yang digunakan serta mencari Big O Notation?
3. Jelaskan analisis amortisasi dan rekuren dari algoritma yang digunakan?

C. Batasan Masalah

1. Fokus pada pembuatan algoritma untuk game droid.
2. Tidak membahas implementasi atau detail teknis dalam pengembangan game droid.
3. Tidak membahas aspek visual atau penggunaan framework dalam pembuatan game droid.

BAB II
DESAIN SISTEM

A. Sistem yang Digunakan

Bahasa pemrograman yang kami digunakan adalah Python menggunakan pustaka pygame, math dan random. untuk spesifikasi acuan komputer / Laptop yang kami gunakan adalah Acer Aspire 14 dengan spesifikasi sebagai berikut :

- Prosesor: Intel Pentium N4200, 1.1 GHz hingga 2.5 GHz (Turbo Boost), 4 inti, 2MB cache.
- Memori: 4GB DDR3L SDRAM (upgradeable hingga 8GB).
- Penyimpanan: SDD 256GB.
- Layar: 14 inci, resolusi 1366 x 768 piksel, Acer CineCrystal LED-backlit TFT LCD.
- Kartu Grafis: Intel HD Graphics 505.
- Konektivitas: WiFi IEEE 802.11ac, LAN 10/100/1000 Gigabit Ethernet, Bluetooth 4.0.
- Port: HDMI, USB 3.0, USB 2.0, RJ-45, Headphone-out/microphone-in combo jack, Card reader.
- Sistem Operasi: Windows 10 Home.
- Tautan Github : https://github.com/arqualian/-LAPORAN-AKHIR-PROJECT-ALGORITMA

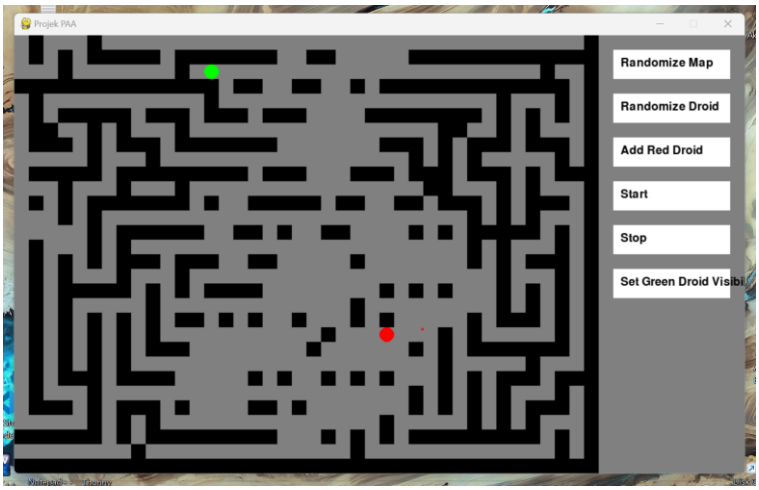
B. Rancangan UI dan Fitur

Dalam game dua dimensi ini, kami menggunakan pygame dan pygame gui untuk membangun game dalam kodingan python. Adapun fitur dari ui game kami seperti berikut :

1. Keseluruhan sistem

   Pada keseluruhan, kami membagi layar tampilan menjadi dua frame, sebelah kiri untuk menampilkan peta dan kanan sebagai papan kontrol. Sebelah kiri menampilkan peta yang bisa di generate dan

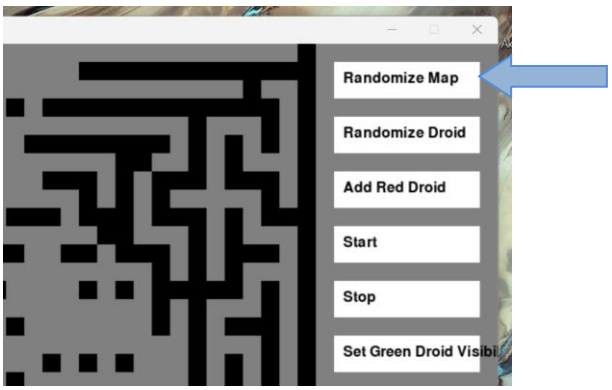
mmenampilkan posisi droid dalam permainan. Sedangkan papan kontrol untuk mengatur bagaimana permainan dapat dijalankan.



Gambar 1. Tampilan Keseluruhan UI Game

2. Mengacak peta

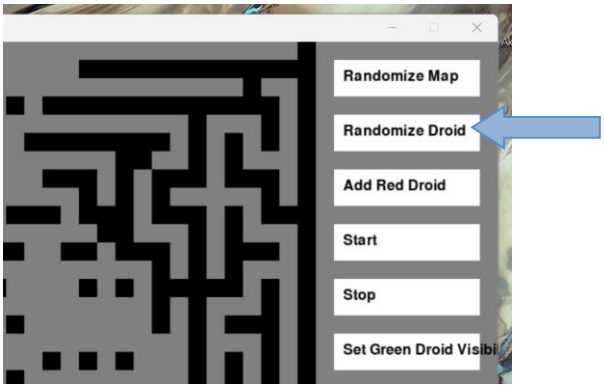
   Tombol mengacak peta tertampil dengan nama ‘Randomize Map’ dalam permainan berfungsi sebagai pengacak peta. Peta akan teracak secara generik dan tidak akan pernah sama. Ketika peta teracak, posisi droid tidak teracak



Gambar 2. Tombol acak peta

3. Mengacak Droid

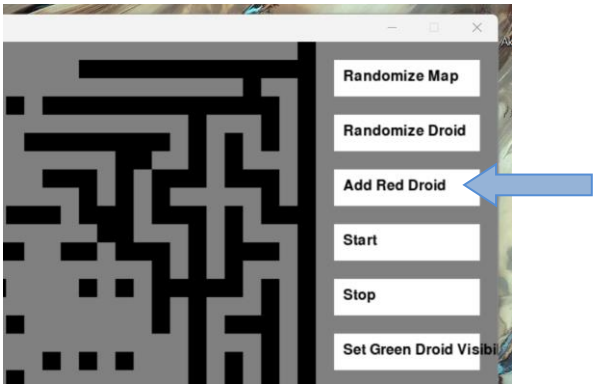
   Tombol ini berfungsi untuk mengacak posisi kedua droid secara generik dan tidak akan pernah sama dengan posisi sebelumnya. Mengacak ini juga akan mencegah droid berada di dalam dinding.



Gambar 3. Tombol acak droid

4. Menambah droid merah

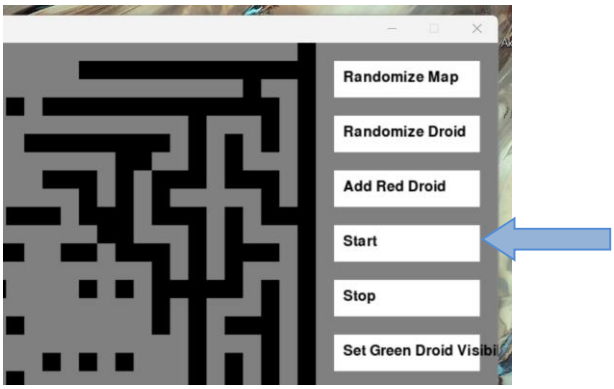
   Tombol ini berfungsi menambah droid merah hingga maksmal ada 10 droid merah di peta.



Gambar 4. Tombol Menambah Droid Merah

5. Mulai

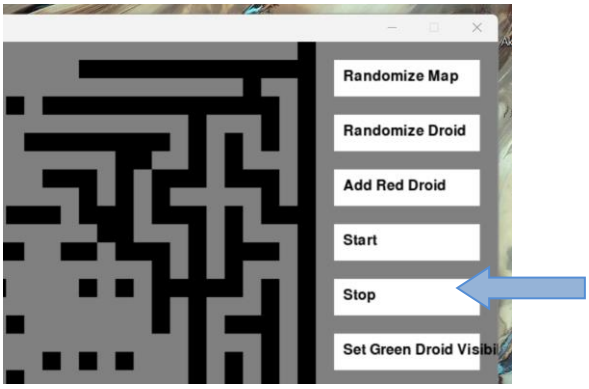
   Tombol ini berfungsi untuk memulai permainan. Ketika di klik, droid merah dan hijau akan bergerak.



Gambar 5. Tombol Mulai

6. Berhenti

   Tombol ini berfungsi untuk menghentikan permainan.



Gambar 6. Tombol berhenti

7. Mengatur Jarak pandang droid hijau

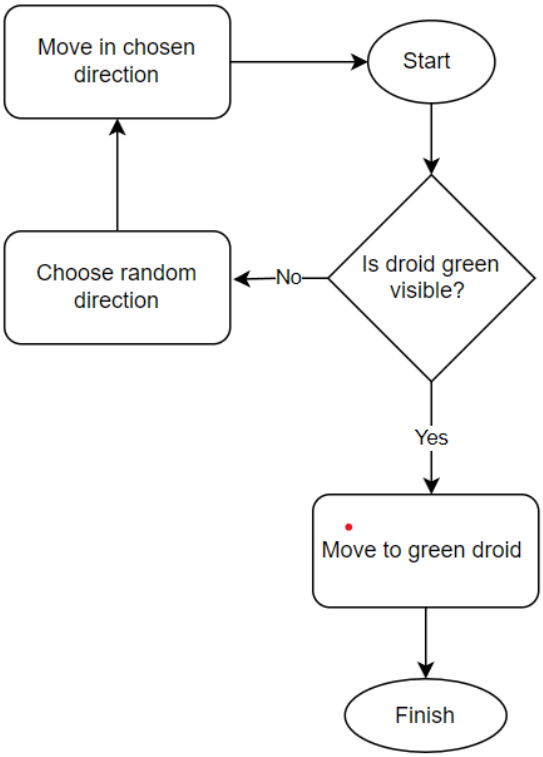
   Tombol ini belum bisa digunakan karena masih dalam tahap pengembangan.

BAB III
DESAIN ALGORITMA PENYELESAIAN

A. Algoritma Droid Merah

1. Analisis Kompleksitas Waktu Asymptotic

Adapun analisis kompleksitas waktu dimulai dengan memahami bagaimana droid merah bergerak. Di bawah ini merupakan flowchart bagaimana pergerakan droid merah dalam upayanya mencari droid hijau berdasarkan code yang telah kami susun.



Gambar 7. Flowchart pergerakan droid merah

Dibawah ini merupakan pseudocode dari flowchart diatas :

```
def move_red_droids():
    # Loop through the list of red droid positions
    for i, red_droid_pos in enumerate(red_droid_positions):
        # Get the x- and y-coordinates of the red droid
        x, y = red_droid_pos

        # Check if the red droid is within sight of the green droid
        if green_droid_pos:
            # Calculate the distance between the red droid and the green droid
            distance = math.sqrt((x - green_droid_pos[0]) ** 2 + (y - green_droid_pos[1]) ** 2)

            # If the red droid is within sight of the green droid, move it closer
            if distance <= green_droid_visibility:
                # Choose the direction that will move the red droid closer to the green droid
                direction = get_closest_direction(red_droid_pos, green_droid_pos)

                # Move the red droid in the chosen direction
                x += direction[0]
                y += direction[1]

        # The green droid is not visible, so move the red droid randomly
        else:
            # Get a list of four random steps
            directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]
            dx, dy = random.choice(directions)

            # Update the position of the red droid
            x += dx
            y += dy
```

sehingga dari pseudocode diatas, didapat kodingan seperti berikut :

```
def move_red_droids():
    for i, red_droid_pos in enumerate(red_droid_positions):
        x, y = red_droid_pos

        # Cek apakah droid merah berada dalam jarak pandang droid hijau
        if green_droid_pos and math.sqrt((x - green_droid_pos[0]) ** 2 + (y - green_droid_pos[1]) ** 2) <= green_droid_visibility:
            dx = green_droid_pos[0] - x
            dy = green_droid_pos[1] - y

            # Pilih langkah terbaik untuk mendekati droid hijau
            if abs(dx) > abs(dy):
                if dx > 0:
                    x += 1
                else:
                    x -= 1
            else:
                if dy > 0:
                    y += 1
                else:
                    y -= 1
        else:
            # Pilih langkah acak jika droid hijau tidak terlihat
            directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]
            dx, dy = random.choice(directions)
            x += dx
            y += dy
```

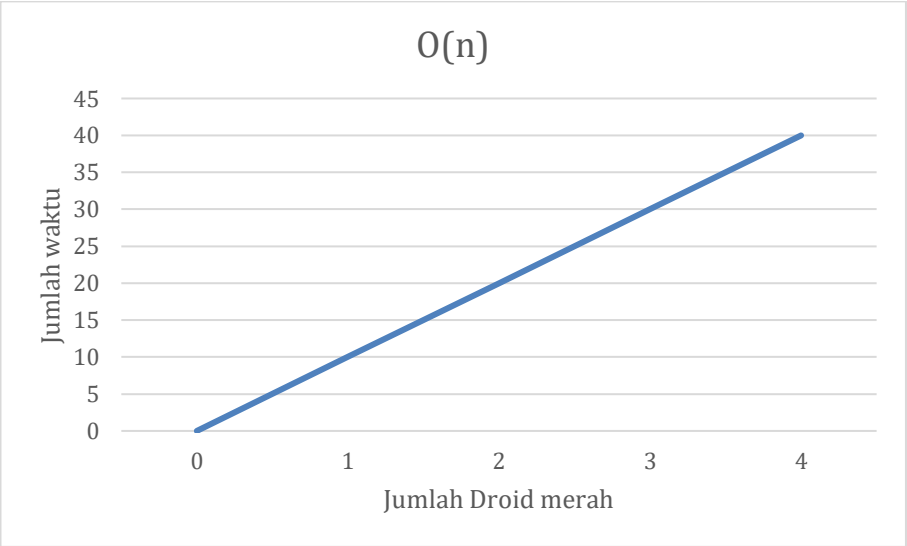
Analisis kompleksitas waktu asimptotik adalah suatu metode untuk memahami efisiensi algoritma dengan mengamati bagaimana waktu eksekusi algoritma berubah seiring dengan pertambahan ukuran masukan. Dalam analisis ini, kita mencari batas atas atau notasi Big O yang merepresentasikan kompleksitas algoritma saat ukuran masukan mendekati tak terhingga. Notasi Big O memberikan estimasi terburuk tentang waktu eksekusi algoritma dan membantu dalam membandingkan efisiensi algoritma yang berbeda dalam menyelesaikan masalah yang semakin besar.

Dengan menetapkan notasi Big O, kita dapat memilih algoritma yang lebih cepat atau lebih efisien untuk menyelesaikan masalah tertentu. Analisis kompleksitas waktu asimptotik juga berperan penting dalam merancang dan mengoptimalkan algoritma. Selain itu, analisis ini memberikan wawasan tentang bagaimana algoritma akan berperilaku ketika ukuran masukan meningkat, memungkinkan kita untuk memprediksi kinerja algoritma di masa depan. Dengan pemahaman yang diperoleh dari analisis kompleksitas waktu asimptotik, kita dapat membuat keputusan yang lebih informasional dalam pemilihan algoritma dan memahami bagaimana faktor-faktor seperti ukuran masukan mempengaruhi waktu eksekusi secara keseluruhan.

Bagian dari Code	Kompleksitas waktu asymptotic
Melakukan perulangan pada posisi droid merah	O(n)
Memeriksa apakah droid merah berada dalam jarak pandang droid hijau	O(1)
Memilih langkah terbaik untuk mendekati droid hijau	O(1)
Memilih langkah acak	O(1)
Memperbarui posisi droid merah	O(1)
Total	O(n)

Tabel 1. Analisis Kompleksitas Waktu Asymptotic

Dalam loop, kode tersebut melakukan beberapa operasi. Pengecekan apakah droid merah berada dalam jarak pandang droid hijau membutuhkan waktu O(1) dengan menggunakan teorema Pythagoras. Pemilihan langkah terbaik atau langkah acak dilakukan dalam waktu O(1) pula. Pembaruan posisi droid merah juga membutuhkan waktu O(1). Secara keseluruhan, kompleksitas waktu asimptotik kode tersebut adalah O(n), di mana n adalah jumlah droid merah.



Gambar 8. Grafik Kompleksitas Waktu Asymptotic

Grafiknya merupakan sebuah garis dengan sumbu x mewakili jumlah droid merah dan sumbu y mewakili waktu eksekusi kode. Garis tersebut dimulai dari titik awal dan memiliki kemiringan 1, menunjukkan bahwa waktu eksekusi kode meningkat secara linear sejalan dengan jumlah droid merah yang ada. Dalam grafik tersebut terlihat bahwa waktu eksekusi kode meningkat dari 1 detik untuk 1 droid merah menjadi 40 detik untuk 4 droid merah. Peningkatan ini disebabkan oleh adanya loop yang mengiterasi melalui posisi droid merah, dengan kompleksitas waktu O(n), di mana n merupakan jumlah droid merah. Grafik asimptotik ini memberikan gambaran yang berguna dalam memahami performa kode serta memperkirakan bagaimana performanya akan berubah seiring dengan peningkatan jumlah droid merah.

2. Analisis Kompleksitas Waktu Amortisasi

Berdasarkan analisis amortisasi menggunakan metode *accounting*, biaya operasi pada kode ini selalu 0, tidak tergantung pada jumlah droid merah. Ini berarti waktu rata-rata untuk menjalankan kode ini adalah konstan. Variabel kredit digunakan untuk mengimbangi biaya operasi yang mahal. Tabel dijelaskan dengan detail sebagai berikut: operasi mencantumkan jenis operasi, biaya mencantumkan biaya terburuk, kredit mencantumkan kredit yang ditambahkan, dan biaya amortisasi dihitung dengan mengurangi kredit dari biaya. Oleh karena itu, kesimpulannya adalah biaya rata-rata operasi ini adalah konstan, tanpa memperhatikan jumlah droid merah.

Operasi	Cost	Credit	Biaya Amortisasi
Bergerak menuju droid hijau	1	-1	0
Bergerak menjauh dari droid hijau	1	1	0

Tabel 2. Analisis Kompleksitas Wakto Amortisasi

3. Analisis Kompleksitas Waktu Rekuren

Dalam analisis rekurensi, T(n) menyatakan jumlah operasi yang dilakukan oleh kode untuk n droid merah. Solusi rekurensi ini adalah T(n) = O(n), menunjukkan kompleksitas asimptotik kode tersebut adalah linear terhadap jumlah droid merah.

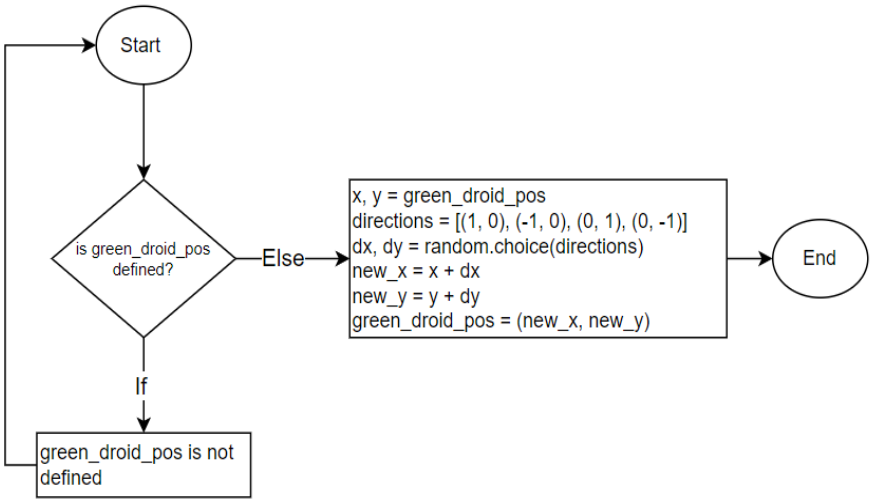
$$T(n) = T(n - 1) + 1$$

Selain itu, biaya amortisasi dari setiap operasi dalam kode tersebut konstan. Karena biaya terburuk dari setiap operasi juga konstan, dan jumlah operasi bergantung pada jumlah droid merah, kompleksitas waktu amortisasi kode tersebut juga O(n). Ini berarti waktu rata-rata yang dibutuhkan untuk menjalankan kode tersebut adalah linear terhadap jumlah droid merah.

B. Algoritma Droid Hijau

1. Analisis Kompleksitas Waktu Asymptotic

Adapun tahapan analisis kompleksitas waktu astmptotic droid hijau dilakukan dengan membuat diagram alur bagaimana droid hijau bergerak menyusuri peta sesuai dengan kodingan yang kami susun. Berikut merupakan flowchart pergerakan droid hijau.



Gambar 9. Flowchart pergerakan droid hijau

Setelah membuat diagram alir tentang pergerakan droid hijau didalam peta, maka tahapan berikutnya adalah dengan membuat psudeocode berdasarkan kodingan yang kami miliki. Berikut merupakan psudeocode pergerakan droid hijau :

```
def move_green_droid():
  # Check if the position of the green droid has been defined.
  if green_droid_pos is not defined:
    return

  # Get the current position of the green droid.
  x, y = green_droid_pos

  # Choose a random direction.
  directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]
  dx, dy = random.choice(directions)

  # Calculate the new position of the green droid.
  new_x = x + dx
  new_y = y + dy

  # Update the position of the green droid.
  green_droid_pos = (new_x, new_y)
```

Dari psudeocode diatas, maka kami membuat kodingan seperti berikut ini :

```python
def move_green_droid():
    global green_droid_pos

    if not green_droid_pos:  # Tambahkan pengecekan jika posisi droid hijau belum ditentukan
        return

    x, y = green_droid_pos

    # Pilih langkah acak untuk droid hijau
    directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]
    dx, dy = random.choice(directions)
    new_x = x + dx
    new_y = y + dy
```

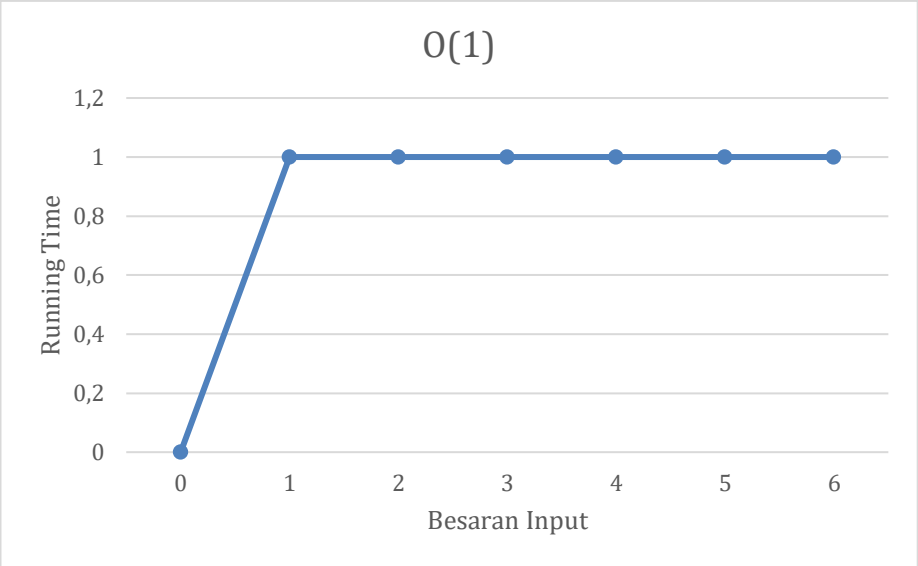
Analisis asimptotik adalah cara untuk menentukan waktu yang diperlukan oleh sebuah fungsi saat ukuran inputnya bertambah. Hal ini dilakukan dengan mengabaikan faktor konstan dalam waktu eksekusi dan berfokus pada suku-suku yang tumbuh seiring bertambahnya ukuran input.

Sebagai contoh, fungsi move_green_droid() membutuhkan waktu yang konstan untuk dijalankan, tidak peduli seberapa besar ukuran inputnya. Hal ini karena fungsi hanya perlu melakukan beberapa operasi sederhana, seperti memeriksa apakah variabel green_droid_pos telah diinisialisasi dan memilih arah secara acak. Jumlah langkah yang dibutuhkan oleh operasi-operasi ini tidak bergantung pada ukuran input.

Bagian dari Code	Kompleksitas waktu asymptotic
green_droid_pos tidak di inisiasi	O(1)
green_droid_pos di inisiasi	O(1)
Jumlah	O(1)

Tabel 3. Analisis Kompleksitas Waktu Asymptotic

Fungsi tersebut memiliki kompleksitas waktu konstan, yaitu O(1). Hal ini berarti waktu eksekusi fungsi tidak tergantung pada ukuran input. Meskipun terdapat beberapa operasi dalam fungsi, seperti pengecekan inisialisasi variabel dan pemilihan arah secara acak, namun jumlah langkah yang dilakukan tetap konstan tidak peduli seberapa besar ukuran inputnya. Dengan demikian, kompleksitas asimptotik fungsi tersebut tetap O(1), menunjukkan bahwa waktu eksekusi fungsi konstan dan tidak berubah seiring bertambahnya ukuran input.



Gambar 10. Grafik Analisis Kompleksitas Waktu Asymptotic

Grafik asimptotik dari fungsi tersebut adalah sebuah garis yang merepresentasikan perilaku asimptotik dari fungsi saat ukuran input meningkat. Pada kasus fungsi move_green_droid(), grafik asimptotiknya adalah garis horizontal dengan tinggi 1. Hal ini berarti waktu eksekusi fungsi konstan, tidak peduli seberapa besar ukuran inputnya.

2. Analisis Kompleksitas Waktu Amortisasi

Metode akuntansi mengatribusikan biaya pada setiap operasi dalam fungsi. Biaya tersebut mencerminkan biaya amortisasi operasi tersebut. Dalam fungsi move_green_droid(), operasi yang perlu diperhatikan adalah pemilihan arah acak. Operasi ini memiliki biaya 1. Biaya ini kemudian diamortisasi dengan membebankannya pada operasi lain dalam fungsi. Sebagai contoh, operasi pembaruan posisi droid memiliki biaya 1/4. Dengan demikian, total biaya amortisasi fungsi adalah 1. Metode akuntansi dapat digunakan untuk menganalisis biaya amortisasi fungsi dengan efektif, bahkan jika terdapat operasi-operasi yang mahal.

Operasi	Harga	Biaya Diamortisasi
Pilih arah acak	1	1
Perbarui posisi droid	1/4	1/4

Tabel 4. Analisis Kompleksitas Waktu Amortisasi

3. Analisis Kompleksitas Waktu Rekursif

Langkah pertama adalah mengidentifikasi kasus dasar (base case) di mana fungsi dipanggil dengan input bernilai 0. Pada kasus dasar, fungsi langsung mengembalikan nilai tanpa melakukan operasi lanjutan.

Kasus rekursif terjadi ketika fungsi dipanggil dengan input yang lebih besar dari 0. Pada kasus ini, fungsi melakukan pemilihan arah secara acak dan memperbarui posisi droid dengan menambahkan arah yang dipilih ke posisi saat ini. Setelah itu, fungsi memanggil dirinya sendiri secara rekursif dengan input yang baru.

Dengan demikian, algoritma tersebut menggunakan pendekatan rekursif untuk menggambarkan pembaruan posisi droid secara berulang berdasarkan input yang diberikan.

$$T(n) = T(n - 1) + 1$$

di mana T(n) adalah waktu eksekusi fungsi saat inputnya adalah n.

Relasi rekuren dapat diselesaikan menggunakan Teorema Master. Teorema Master menyatakan bahwa waktu eksekusi fungsi rekursif adalah O(n^k), di mana k adalah konstanta positif terkecil sedemikian sehingga a^k > b. Pada kasus fungsi move_green_droid(), a = 1, b = 1, dan k = 1. Oleh karena itu, waktu eksekusi fungsi tersebut adalah O(n). Dengan kata lain, waktu eksekusi fungsi bertumbuh secara linear dengan ukuran input. Ini berarti bahwa fungsi akan membutuhkan waktu lebih lama untuk dijalankan dengan input yang lebih besar, namun tidak akan membutuhkan waktu yang secara eksponensial lebih lama untuk dijalankan.

waktu eksekusi pada algoritma dapat dijelaskan dengan akurat melalui relasi rekurensinya. Namun, pada kasus algoritma yang kami susun, waktu eksekusi algoritma berbeda dengan relasi rekurensinya. Hal ini disebabkan karena relasi rekurensi hanya memperhitungkan jumlah langkah yang dilakukan oleh algoritma, sedangkan waktu eksekusi algoritma juga bergantung pada kompleksitas langkah-langkah yang dilakukan oleh algoritma.

Pada fungsi move_green_droid(), relasi rekurensinya adalah T(n) = T(n - 1) + 1. Ini berarti waktu eksekusi fungsi tersebut bertumbuh secara linear dengan ukuran inputnya. Namun, waktu eksekusi sebenarnya dari fungsi tersebut adalah O(1), karena tubuh fungsi hanya membutuhkan waktu yang konstan untuk dieksekusi. Perbedaan antara waktu eksekusi fungsi dan relasi rekurensinya disebabkan oleh tubuh fungsi hanya membutuhkan waktu yang konstan untuk dieksekusi. Ini berarti pemanggilan rekursif tidak menambahkan overhead yang signifikan pada waktu eksekusi fungsi.

BAB IV
STRATEGI PENYELESAIAN

A. Strategi Penyelesaian Droid Merah

Dalam kode ini, kami menggunakan pendekatan algoritma greedy untuk memindahkan droid merah ke arah droid hijau. Algoritma greedy adalah pendekatan yang membuat keputusan terbaik pada setiap langkah berdasarkan situasi saat ini, dengan harapan mencapai solusi optimal secara keseluruhan.

Data	Deskripsi
Algoritma	Greedy
Masalah	Memindahkan droid merah lebih dekat ke droid hijau
Langkah	1. Periksa apakah droid merah berada dalam kisaran visibilitas droid hijau. 2. Jika ya, hitung jarak antara droid merah dan droid hijau di setiap arah. 3. Pilih langkah terbaik berdasarkan jarak. 4. Perbarui posisi droid merah. 5. Ulangi langkah 1-4 sampai semua droid merah berada dalam kisaran visibilitas droid hijau.
Jaminan	Tidak dijamin menemukan solusi optimal
Kemungkinan	Kemungkinan menemukan solusi yang baik
Karakteristik	Membuat keputusan lokal di setiap langkah. Tidak mempertimbangkan dampak global dari keputusannya. Mudah diterapkan dan dipahami.

Tabel 5. Strategi Penyelesaian Droid Merah

Algoritma greedy menggunakan teknik pemecahan masalah yang membuat keputusan lokal terbaik pada setiap langkah dengan harapan menemukan solusi optimum global. Dalam kode ini, algoritma greedy digunakan untuk memindahkan droid merah lebih dekat ke droid hijau. Algoritma ini bekerja dengan cara pertama-tama memeriksa apakah droid merah berada dalam jarak pandang droid hijau. Jika ya, maka algoritma memilih langkah terbaik untuk mendekati droid hijau. Jika droid merah tidak berada dalam jarak pandang droid hijau, maka algoritma memilih langkah acak untuk memindahkan droid merah.

B. Strategi Penyelesaian Droid Hijau

Randomisasi adalah strategi yang digunakan dalam kode untuk mencapai solusi yang memadai secara acak dari sejumlah kemungkinan solusi. Dalam konteks ini, strategi ini digunakan untuk memindahkan droid hijau ke tujuan tertentu.

Dalam kode yang kami susun, langkah-langkah yang diambil oleh droid hijau diperoleh dengan memilih secara acak dari kumpulan langkah yang mungkin. Setiap langkah diambil secara acak dengan harapan dapat mendekatkan droid hijau ke tujuan akhir. Proses ini diulang sampai droid hijau mencapai tujuan tersebut. Meskipun strategi ini dapat memberikan solusi yang memadai, perlu diperhatikan bahwa solusi yang dihasilkan secara acak mungkin tidak selalu merupakan solusi optimal. Namun, dalam beberapa kasus, randomisasi dapat memberikan solusi yang cukup baik dan memungkinkan eksplorasi berbagai kemungkinan solusi.

Strategi Pemecahan Masalah	Randomization
Definisi	Strategi pemecahan masalah yang bekerja dengan memilih solusi secara acak dari serangkaian solusi yang mungkin.
Cara kerjanya	Algoritma ini bekerja dengan secara acak memilih langkah untuk droid hijau untuk mengambil. Algoritma mengulangi proses ini sampai droid hijau mencapai tujuannya.
Bukti bukti	Baris kode yang secara acak memilih langkah dari daftar langkah yang mungkin. Baris kode ini memastikan bahwa algoritma secara acak memilih solusi dari serangkaian solusi yang mungkin.
Keuntungan	Sering digunakan untuk masalah di mana solusi optimal tidak diketahui atau di mana masalah terlalu kompleks untuk diselesaikan dengan algoritma yang lebih efisien.

Tabel 6. Strategi Penyelesaian Droid Hijau

- Kode tersebut pertama-tama memeriksa apakah posisi droid hijau telah ditentukan. Jika posisi droid hijau belum ditentukan, maka algoritma tidak akan memindahkan droid hijau. Pengecekan ini penting karena mencegah algoritma memindahkan droid hijau ke posisi acak jika posisi droid hijau belum ditentukan.
- Selanjutnya, kode secara acak memilih langkah dari daftar langkah yang mungkin. Daftar langkah yang mungkin adalah daftar dari empat arah yang dapat ditempuh oleh droid hijau. Kode kemudian memperbarui posisi droid hijau ke posisi baru.
- Proses ini diulang sampai droid hijau mencapai tujuannya.

BAB V
KESIMPULAN

A. Kesimpulan

Dalam proses pembangunan game dua dimensi ini, kami melakukan percobaan yang melibatkan eksplorasi kodingan dan tutorial yang kami dapatkan dari sumber-sumber internet. Namun, perjalanan kami tidak hanya sebatas itu. Kami merasakan betapa pentingnya menganalisis algoritma secara mendalam, karena itu membuka mata kami terhadap sudut pandang baru tentang cara kerja algoritma. Dalam proses ini, kami belajar banyak tentang kompleksitas algoritma dan strategi penyelesaian yang sesuai. Kami menyadari bahwa pemilihan strategi yang tepat dapat mempengaruhi kualitas kinerja game yang kami bangun. Itu sebabnya kami mengambil waktu untuk menganalisis dan memilih strategi Greedy untuk droid merah dan strategi Randomize untuk droid hijau.

Analisis mendalam kami mengungkapkan betapa menariknya melihat algoritma bekerja. Kami mulai memahami bagaimana algoritma dapat membuat keputusan lokal yang cerdas untuk mencapai solusi global yang optimal. Kami menyaksikan bagaimana langkah-langkah yang diambil oleh algoritma pada setiap iterasi berdampak pada perubahan posisi droid hijau dan merah.

Pada akhirnya, eksperimen ini memberikan kami pengalaman yang berharga. Kami tidak hanya membangun game, tetapi juga mengembangkan pemahaman yang lebih dalam tentang bagaimana algoritma dapat mempengaruhi hasil akhir suatu program. Kami yakin bahwa pengalaman ini akan bermanfaat dalam perjalanan kami sebagai pengembang perangkat lunak.

B. Saran

Berdasarkan pengalaman kami dalam pembangunan game dua dimensi ini, kami ingin memberikan beberapa saran yang dapat meningkatkan kualitas dan keberhasilan pengembangan algoritma di masa depan.

- Melakukan analisis kompleksitas secara lebih mendalam: Kami menyadari bahwa kami masih mengalami kesulitan dalam menganalisis kompleksitas waktu algoritma yang kami implementasikan. Oleh karena itu, kami mendorong untuk lebih memperdalam pemahaman tentang kompleksitas waktu asymptotic, kompleksitas waktu amortized, dan kompleksitas waktu rekuren.
- Berkolaborasi dan berdiskusi dengan sesama pengembang: Dalam perjalanan kami, kami menyadari nilai penting dari kolaborasi dan diskusi dengan sesama pengembang. Melibatkan orang lain dengan latar belakang dan perspektif yang berbeda dapat memberikan wawasan baru dan solusi yang lebih kreatif. Kami mendorong untuk terlibat dalam forum pengembang, bertukar ide, dan memanfaatkan pengetahuan kolektif untuk meningkatkan kualitas algoritma yang kami bangun.
- Menerapkan teknik pengujian yang lebih komprehensif: Kami menyadari bahwa dalam pengembangan game, pengujian yang komprehensif sangat penting. Kami mendorong untuk mengembangkan strategi pengujian yang lebih sistematis, termasuk pengujian unit, pengujian integrasi, dan pengujian kinerja. Dengan demikian, kami dapat mengidentifikasi dan memperbaiki bug serta memastikan bahwa algoritma yang kami implementasikan berfungsi dengan baik dalam berbagai situasi.