Analyzing Animatronic Movement Probabilities in Five Nights at Freddy 2 Using Markov Chain Graph Models

Muhammad Pandu Pulunggana - 13525059
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: pulungganapandu@gmail.com , 13525059@std.stei.itb.ac.id

***Abstract*— Setiap animatronik pada *Five Nights at Freddy's 2* bergerak menuju ruang penjaga melalui jalur yang berbeda-beda, dengan intensitas yang dikendalikan oleh parameter *AI Level*. Makalah ini memodelkan denah Freddy Fazbear's Pizza sebagai graf berarah berbobot, kemudian merepresentasikan pergerakan tiap animatronik sebagai rantai Markov *absorbing* dengan *Office* sebagai *absorbing state*. Bobot transisi pada tiap sisi diturunkan dari mekanisme *movement time* pada *source code* permainan, lalu dikonversi menjadi peluang berpindah dalam satu jendela waktu observasi. Matriks transisi yang terbentuk dipartisi ke dalam bentuk kanonik *QR* untuk menghitung *expected absorption time* melalui matriks fundamental *N*. Hasil perhitungan pada *AI Level* 10 menunjukkan bahwa Foxy memiliki rute terpendek dan waktu absorbsi tercepat menuju *Office*, sedangkan Mangle, yang melewati sembilan *transient state*, memiliki rute terpanjang dan waktu absorbsi terlama, Toy Bonnie berada pada posisi menengah. Pendekatan ini menunjukkan bahwa rantai Markov pada graf berarah berbobot dapat digunakan untuk mengukur tingkat ancaman setiap animatronik secara kuantitatif, berdasarkan struktur rute dan parameter kesulitan permainan.**

Keywords*— *graf berarah berbobot; rantai markov; absorbing state; expected absorption time; matriks transisi

I. PENDAHULUAN

Five Nights at Freddy's 2 adalah sebuah gim horor maskot yang dirilis oleh Scott Cawthon. Mekanisme inti dari gim ini adalah bertahan hidup dari pukul 12 malam hingga pukul 6 pagi, di mana pemain berperan sebagai seorang satpam malam yang harus mempertahankan diri dari sebelas karakter animatronik. Setiap animatronik memiliki mekanisme pergerakan yang intensitasnya dikendalikan oleh parameter utama yang dikenal sebagai *AI Level*, dengan rentang nilai 0 hingga 20. Semakin tinggi *AI Level*, semakin sering animatronik bergerak menuju lokasi berikutnya ke arah ruangan penjaga. Karena setiap animatronik mengikuti pola pergerakan yang berbeda, makalah ini menganalisis masing-masing animatronik secara individual dan memodelkannya sebagai satu graf berarah terpadu, di mana setiap sisi berarah memiliki bobotnya masing-masing untuk membedakan perilaku pergerakan antar animatronik.

Dari pemodelan tersebut muncul pertanyaan penelitian utama yang menjadi fokus makalah ini yaitu sejauh mana perilaku pergerakan animatronik di FNAF 2 dapat direpresentasikan secara formal melalui graf berarah berbobot dan informasi apa yang dapat digali melalui analisis Markov chain terkait tingkat ancaman individual maupun risiko kumulatif yang dihadapi satpam malam.

II. LANDASAN TEORI

A. Graf

1) Definition

Graf G didefinisikan sebagai pasangan G = (V, E), di mana V adalah himpunan tidak-kosong dari simpul-simpul = $\{v_1, v_2, \ldots, v_n\}$, dan E adalah himpunan sisi yang menghubungkan pasangan simpul = $\{e_1, e_2, \ldots, e_n\}$. Himpunan V tidak boleh kosong, artinya suatu graf harus memiliki setidaknya satu simpul. Namun, himpunan E boleh kosong, artinya suatu graf diperbolehkan untuk tidak memiliki sisi sama sekali [1].

2) Jenis-Jenis Graf

Berdasarkan ada tidaknya gelang dan sisi ganda, graf diklasifikasikan menjadi dua kategori. Graf sederhana adalah graf yang tidak mengandung gelang maupun sisi ganda. Sebaliknya, graf tidak sederhana adalah graf yang mengandung sisi ganda atau gelang. Graf tidak sederhana dibagi lebih lanjut menjadi *multigraf*, yang mengandung sisi ganda antara pasangan simpul yang sama, dan *pseudograf*, yang mengandung sisi gelang (sisi yang berawal dan berakhir pada simpul yang sama).

Berdasarkan arah sisi, graf dibagi menjadi dua jenis. Graf tak-berarah adalah graf yang sisi-sisinya tidak memiliki orientasi arah, artinya sisi *(u, v)* identik dengan *(v, u)*. Graf berarah (juga disebut *digraf*) adalah graf di mana setiap sisi memiliki arah tertentu, direpresentasikan sebagai pasangan terurut *(u, v)* di mana *u* adalah ekor dan *v* adalah kepala dari sisi tersebut [1].

Tabel berikut merangkum lima jenis graf utama berdasarkan kedua kriteria klasifikasi tersebut.

TABEL I. KLASIFIKASI JENIS GRAF

Jenis	Sisi	Sisi ganda dibolehkan?	Sisi gelang dibolehkan?
Graf sederhana	Tak-berarah	Tidak	Tidak
Graf ganda	Tak-berarah	Ya	Tidak
Graf semu	Tak-berarah	Ya	Ya
Graf berarah	Berarah	Tidak	Ya
Graf-ganda berarah	Berarah	Ya	Ya

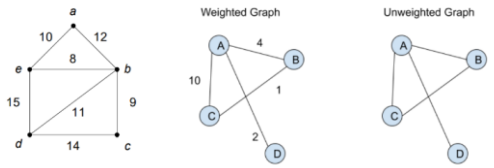
Sumber :

3) Graf Berbobot

Graf berbobot adalah graf di mana setiap sisi $e \in E$ diberi nilai numerik yang disebut bobot (atau biaya). Secara formal, graf berbobot didefinisikan sebagai $G = (V, E, w)$, di mana $w : E \rightarrow \mathbb{R}$ adalah fungsi bobot yang memetakan setiap sisi ke suatu bilangan real.

Bobot dapat merepresentasikan berbagai besaran tergantung pada konteks aplikasinya, seperti jarak antar lokasi, biaya transmisi dalam jaringan, waktu tempuh antar simpul, atau kapasitas suatu koneksi. Pada graf berarah berbobot, bobot sisi *(u, v)* dapat berbeda dari bobot sisi *(v, u)*, mencerminkan biaya yang asimetris (misalnya, jalan satu arah dengan jarak yang berbeda).

Graf tidak berbobot dapat diperlakukan sebagai kasus khusus dari graf berbobot di mana semua sisi diberi bobot yang sama, umumnya $w(e) = 1$ untuk semua $e \in E$.



Gambar 1. Graf berbobot dan tidak berbobot

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf.

B. Markov Chain

Markov chain adalah model matematis yang menggambarkan serangkaian kejadian di mana probabilitas setiap transisi hanya bergantung pada *state* saat ini, bukan pada seluruh riwayat kejadian sebelumnya [2]. Secara formal, suatu proses stokastik $\{X_t\}$ pada ruang keadaan Ω disebut *Markov chain* jika untuk setiap t berlaku persamaan (1):

$$P(X_{t+1} = y \mid X_0, X_1, \ldots, X_t) = P(X_{t+1} = y \mid X_t = x) \quad (1)$$

Markov chain dikatakan bersifat *time-homogeneous* apabila probabilitas transisi tidak bergantung pada waktu t, sehingga berlaku *transition kernel* $K(x, y) = P(X_{t+1} = y \mid X_t = x)$ untuk seluruh t [2]. Setiap langkah pada rantai juga dapat ditentukan melalui representasi *random mapping*, di mana setiap transisi didefinisikan oleh suatu fungsi deterministik dari *state* saat ini dan suatu nilai acak yang dibangkitkan [2]. Lebih lanjut, *Markov chain* dapat direpresentasikan sebagai graf berarah berbobot, di mana simpul merepresentasikan *state* dan sisi berbobot merepresentasikan probabilitas transisi antar *state* [2].

C. Matriks Transisi

Matriks transisi adalah representasi matriks dari seluruh probabilitas transisi antar *state* pada suatu *Markov chain* berhingga. Untuk ruang keadaan $\Omega = \{1, 2, \ldots, n\}$, matriks transisi P berukuran $n \times n$ dengan entri yang didefinisikan sebagai persamaan (2) [2]:

$$P_{ij} = P(X_{t+1} = j \mid X_t = i), \forall i, j \in \Omega \quad (2)$$

Setiap baris matriks transisi membentuk distribusi probabilitas yang valid, memenuhi $\Sigma_j P_{ij} = 1$ dan $P_{ij} \geq 0$. Apabila v_i merupakan vektor distribusi probabilitas pada langkah ke-i, maka distribusi pada langkah berikutnya diperoleh melalui persamaan (3) [3]:

$$v_{i+1} = v_i \, . P \quad (3)$$

Distribusi *state* setelah t langkah dari distribusi awal μ dinyatakan sebagai μP^t, di mana P^t adalah hasil perkalian matriks P sebanyak t kali [2]. Levin dan Peres lebih lanjut menunjukkan bahwa representasi spektral dari matriks transisi yang *reversible* memungkinkan analisis konvergensi menuju distribusi stasioner secara sistematis melalui nilai eigen dari P [2].

D. Absorbing State & Expected Absorption Time

1) Absorbing State

Suatu *state* s_i pada *Markov chain* disebut *absorbing state* apabila sekali rantai memasuki *state* tersebut, ia tidak dapat meninggalkannya, yaitu jika $p_{ii} = 1$ [3]. Suatu *Markov chain* disebut *absorbing Markov chain* apabila memenuhi dua syarat: (1) memiliki setidaknya satu *absorbing state*, dan (2) dari setiap *state* yang bukan *absorbing state* terdapat kemungkinan untuk akhirnya mencapai suatu *absorbing state* [3]. *State* yang bukan *absorbing state* disebut *transient state*. Begitu rantai mencapai *absorbing state*, proses berakhir dan tidak ada transisi lebih lanjut yang terjadi [3].

2) Expected Absorption Time

Expected absorption time adalah nilai harapan dari banyaknya langkah yang diperlukan oleh rantai untuk pertama kali mencapai *absorbing state*, dimulai dari suatu *transient state* tertentu. Untuk *absorbing Markov chain* dengan *transient state* {1, …, t} dan *absorbing state* {t+1, …, n}, matriks transisi dapat dipartisi ke dalam bentuk kanonik berikut, ditunjukkan pada persamaan (4) [2][3]:

$$P = \begin{pmatrix} Q & R \\ 0 & I \end{pmatrix} \quad (4)$$

di mana Q menyatakan transisi antar *transient state*, R menyatakan transisi dari *transient state* ke *absorbing state*, dan I adalah matriks identitas. Matriks fundamental N kemudian didefinisikan sebagai persamaan (5) [2][3]:

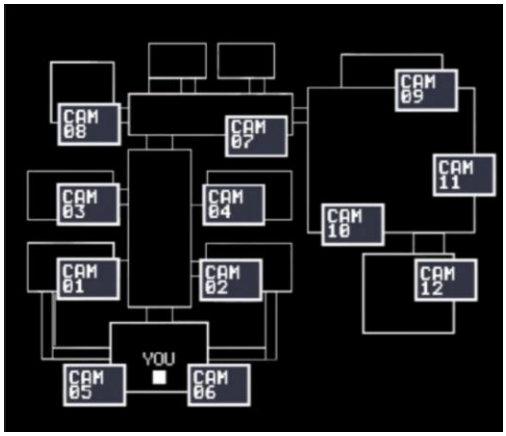
$$N = (I - Q)^{-1} = \sum_{k=0}^{\infty} Q^k \tag{5}$$

Entri N_{ij} dari matriks fundamental menyatakan ekspektasi banyaknya kali rantai mengunjungi *transient state* j sebelum terserap, apabila dimulai dari *transient state* i. Dengan demikian, ekspektasi waktu absorbsi dari *transient state* i dinyatakan sebagai persamaan (6) [3]:

$$t_i = \sum_j N_{ij}\,, atau\ dalam\ bentuk\ vektor: \boldsymbol{t} = N\mathbf{1} \tag{6}$$

di mana 1 adalah vektor kolom yang seluruh entrinya bernilai 1. Hajihashemi menunjukkan bahwa melalui metode *fundamental matrix* ini, besaran seperti *fixation probability* dan *fixation time* dapat dihitung secara analitik, dengan hasil yang terbukti ekuivalen dengan hasil simulasi [3].

III. Metode Penelitian



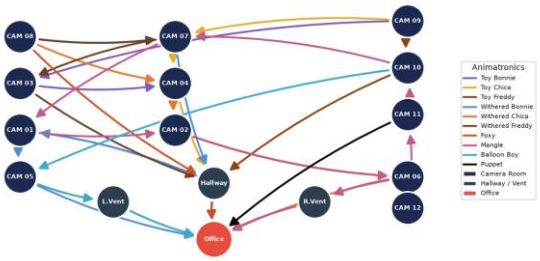
Gambar 2. Denah lokasi Freddy Fazbear's Pizza di FNAF 2

Sumber : https://freddy-fazbears-pizza.fandom.com/wiki/Map:Freddy_Fazbear%27s_Pizza_(FNaF2)

Terdapat daftar ruangan dan titik pengawasan pada *Freddy Fazbear's Pizza* beserta kode kameranya. Informasi ini digunakan untuk mengelompokkan setiap ruangan berdasarkan fungsinya dalam denah, memudahkan penandaan pada visualisasi graf, dan membantu dalam analisis pergerakan animatronik.

1. CAM01 – Party Room 1
2. CAM02 – Party Room 2
3. CAM03 – Party Room 3
4. CAM04 – Party Room 4
5. CAM05 – Left Air Vent
6. CAM06 – Right Air Vent
7. CAM07 – Main Hall
8. CAM08 – Parts/Service
9. CAM09 – Show Stage
10. CAM10 – Game Area
11. CAM11 – Prize Corner
12. CAM12 – Kid's Cove
13. Hall – titik lorong utama menuju Office
14. RVent – titik akhir Right Air Vent menuju Office
15. LVent – titik akhir Left Air Vent menuju Office
16. Office – ruang kendali satpam malam (simpul tujuan)

Berdasarkan daftar ruangan dan titik pengawasan tersebut, setiap lokasi direpresentasikan sebagai simpul (*vertex*) dalam graf berarah, sedangkan hubungan antarlokasi direpresentasikan sebagai sisi berarah (*directed edge*) yang menunjukkan arah kemungkinan perpindahan animatronik sesuai mekanisme permainan. Ruangan yang memiliki kamera pengawasan direpresentasikan oleh simpul CAM01 hingga CAM12, sementara *Hallway*, *R.Vent*, dan *L.Vent* berfungsi sebagai titik transisi yang menghubungkan jalur pergerakan animatronik menuju *Office*, yaitu simpul tujuan akhir tempat penjaga malam berada. Representasi ini memungkinkan struktur denah *Freddy Fazbear's Pizza* divisualisasikan dalam bentuk graf, sehingga hubungan antarruangan serta jalur perpindahan setiap animatronik dapat diamati dengan lebih jelas.



Gambar 3. Graf Berarah Pengerakan Animatronic

Penampilan Graf Berarah dari pengerakan animatronic dibuat dengan program tersebut.

```python
import networkx as nx
import matplotlib.pyplot as plt

MOVEMENTS = {
    "Toy Bonnie": [(9,3),(3,4),(4,2),(2,6),(6,102),(102,104)],
    "Toy Chica": [(9,7),(7,4),(4,101),(101,1),(1,5),(5,103),(103,104)],
    "Toy Freddy": [(9,10),(10,101),(101,104)],
    "Withered Bonnie": [(8,7),(7,101),(101,1),(1,5),(5,104)],
    "Withered Chica": [(8,4),(4,2),(2,6),(6,104)],
    "Withered Freddy": [(8,7),(7,3),(3,101),(101,104)],
    "Foxy": [(8,101),(101,104)],
    "Mangle": [(12,11),(11,10),(10,7),(7,1),(1,2),(2,6),(6,102),(102,104)],
    "Balloon Boy": [(10,5),(5,103),(103,104)],
    "Puppet": [(11,104)]
}

G = nx.MultiDiGraph()

for edges in MOVEMENTS.values():
    G.add_edges_from(edges)

POS = {
    8:(0,4.5), 7:(2,4.5),
    9:(5,5),
    3:(0,3), 4:(2,3),
    10:(5,3.5),
    1:(0,1.5), 2:(2,1.5),
    11:(5,2),
    5:(0,0), 6:(5,0),
    101:(2.5,-0.2),
    103:(1.2,-0.8),
    102:(3.8,-0.8),
    104:(2.5,-2),
    12:(5,-1)
}

nx.draw_networkx(
    G,
    pos=POS,
    with_labels=True,
    arrows=True
)

plt.show()
```

Gambar 4. Implementasi Program Visualisasi Graf Berarah

Kode pada Gambar 4 melanjutkan data yang telah didefinisikan sebelumnya, yaitu *NODE_LABELS*, *MOVEMENTS*, *COLORS*, dan *POS*, menjadi graf yang siap divisualisasikan. Graf dibangun menggunakan *nx.MultiDiGraph()* karena beberapa animatronik dapat melewati simpul maupun jalur yang sama sehingga setiap lintasan perlu disimpan sebagai sisi yang terpisah. Proses pembentukan graf dilakukan dengan melakukan perulangan pada seluruh data *MOVEMENTS*, kemudian setiap pasangan lokasi asal dan tujuan ditambahkan sebagai sisi beserta atribut *character* yang menunjukkan animatronik pemilik lintasan tersebut. Selanjutnya, warna simpul ditentukan berdasarkan fungsi lokasinya. Simpul *Office* diberi warna merah sebagai tujuan akhir, *Hallway*, *R.Vent*, dan *L.Vent* diberi warna abu gelap sebagai titik transisi, sedangkan seluruh ruang kamera *CAM01* hingga *CAM12* diberi warna biru tua. Posisi setiap simpul mengikuti koordinat pada *POS* yang telah ditentukan secara manual agar sesuai dengan tata letak denah permainan.

Pada tahap visualisasi, sisi graf digambar melalui perulangan untuk setiap animatronik agar masing-masing jalur dapat ditampilkan dengan warna berbeda sesuai data pada *COLORS*, dengan panah sedikit melengkung (*connectionstyle="arc3,rad=0.08"*) agar sisi yang searah tidak saling menutupi, serta legenda manual yang ditempatkan di luar area graf agar tidak mengganggu visualisasi. Hasil akhirnya disimpan sebagai gambar beresolusi tinggi sebagai luaran analisis pergerakan animatronik.

Langkah berikutnya adalah menganalisis graf tersebut secara kuantitatif dengan memberi bobot probabilitas pada setiap sisi berdasarkan AI Level, sehingga graf dapat dimodelkan sebagai rantai Markov dan disusun ke dalam matriks transisi untuk menghitung *expected absorption time*, yaitu ekspektasi waktu yang dibutuhkan setiap animatronik untuk mencapai Office.

```python
T0 = 50_000
K_FACTOR = 157
MAX_AI_LEVEL = 20


def movement_time(ai_level: int) -> float:
    if not (0 <= ai_level <= MAX_AI_LEVEL):
        raise ValueError(f"AI Level harus 0..{MAX_AI_LEVEL}, didapat {ai_level}")
    return max(T0 - ai_level * K_FACTOR, 1.0)


def move_threshold_time(ai_level: int) -> float:
    if ai_level == 0:
        return float("inf")
    return movement_time(ai_level) / ai_level


def transition_weight(ai_level: int, dt: float = 1000.0) -> float:
    t_wait = move_threshold_time(ai_level)
    if t_wait == float("inf"):
        return 0.0
    return min(1.0, dt / t_wait)


def transition_weight_foxy(ai_level: int, dt: float = 1000.0) -> float:
    if ai_level == 0:
        return 0.0
    t_wait = movement_time(ai_level) / ((ai_level + 1) * 1.2)
    return min(1.0, dt / t_wait)


def transition_weight_balloon_boy_first(ai_level: int, dt: float = 1000.0) -> float:
    if ai_level == 0:
        return 0.0
    t_wait = (movement_time(ai_level) + 96_000) / ai_level
    return min(1.0, dt / t_wait)
```

Gambar 5. Implementasi Program Bobot Transisi

Pada bagian ini, rumus bobot transisi dibangun berdasarkan mekanisme waktu gerak yang ditemukan langsung pada animatronic_base.py di source code game. Konstanta T0, K_FACTOR, dan MAX_AI_LEVEL digunakan oleh fungsi movement_time() untuk menghitung waktu gerak dasar sesuai AI Level, kemudian move_threshold_time() menentukan waktu tunggu sebelum animatronik berpindah simpul, dan transition_weight() mengonversi waktu tunggu tersebut menjadi peluang berpindah dalam satu jendela observasi dt. Dua fungsi tambahan, transition_weight_foxy() dan transition_weight_balloon_boy_first(), juga disertakan karena kedua animatronik tersebut memiliki rumus waktu gerak yang berbeda dari animatronik lainnya pada source code aslinya.

```python
from importlib import import_module

bobot = import_module("1_bobot")

OFFICE_STATE = 104

ROUTES: dict[str, list[int]] = {
    "Toy Bonnie":       [9, 3, 4, 2, 6, 102, 104],
    "Toy Chica":        [9, 7, 4, 101, 1, 5, 103, 104],
    "Toy Freddy":       [9, 10, 101, 104],
    "Withered Bonnie":  [8, 7, 101, 5, 104],
    "Withered Chica":   [8, 4, 2, 6, 104],
    "Withered Freddy":  [8, 7, 3, 101, 104],
    "Foxy":             [8, 101, 104],
    "Mangle":           [12, 11, 10, 7, 1, 2, 6, 101, 102, 104],
    "Balloon Boy":      [10, 5, 103, 104],
    "Puppet":           [11, 104],
}


def build_markov_chain(route: list[int], ai_level: int, dt: float = 1000.0
                       ) -> dict[int, dict[int, float]]:
    w = bobot.transition_weight(ai_level, dt)
    chain: dict[int, dict[int, float]] = {}

    for idx, state in enumerate(route):
        if state == OFFICE_STATE:
            chain[state] = {OFFICE_STATE: 1.0}
            continue

        next_state = route[idx + 1]
        chain[state] = {
            next_state: w,
            state: 1.0 - w,
        }

    return chain
```

Gambar 6. Implementasi Program Rantai Markov

Selanjutnya, rute pergerakan setiap animatronik yang tersimpan pada ROUTES diubah menjadi rantai Markov melalui fungsi build_markov_chain(). Fungsi ini mengambil rute satu animatronik berdasarkan namanya, memeriksa apakah ada bobot khusus pada SPECIAL_WEIGHTS untuk state tertentu, lalu membangun struktur transisi di mana setiap state non-Office memiliki dua kemungkinan: berpindah ke state berikutnya dengan peluang w, atau tetap berada di state yang sama dengan peluang 1-w. Office (104) sendiri ditetapkan sebagai *absorbing state* yang selalu kembali ke dirinya sendiri.

```python
import numpy as np
from importlib import import_module

markov = import_module("2_markov")

def build_transition_matrix(route: list[int], ai_level: int, dt: float = 1000.0
                            ) -> tuple[np.ndarray, list[int]]:
    chain = markov.build_markov_chain(route, ai_level, dt)

    states = list(dict.fromkeys(route))
    n = len(states)
    index = {s: i for i, s in enumerate(states)}

    P = np.zeros((n, n))
    for state, transitions in chain.items():
        i = index[state]
        for target, prob in transitions.items():
            j = index[target]
            P[i, j] = prob

    return P, states


def partition_QR(P: np.ndarray, states: list[int], absorbing_state: int = 104
                 ) -> tuple[np.ndarray, np.ndarray, list[int]]:
    absorb_idx = states.index(absorbing_state)
    transient_idx = [i for i in range(len(states)) if i != absorb_idx]

    Q = P[np.ix_(transient_idx, transient_idx)]
    R = P[np.ix_(transient_idx, [absorb_idx])]
    transient_states = [states[i] for i in transient_idx]

    return Q, R, transient_states
```

Gambar 7. Implementasi Program Matriks Transisi

Setelah rantai Markov terbentuk, struktur tersebut dikonversi menjadi representasi matriks agar perhitungan probabilitasnya dapat dilakukan secara sistematis. Fungsi build_transition_matrix() menyusun seluruh peluang transisi ke dalam matriks P berukuran n×n sesuai jumlah state unik pada rute, sementara partition_QR() memecah matriks tersebut menjadi submatriks Q (transisi antar-state transient) dan R (transisi menuju Office), sesuai bentuk kanonik matriks pada rantai Markov absorbing.

```python
import numpy as np
from importlib import import_module

matrix_mod = import_module("3_matrix")


def fundamental_matrix(Q: np.ndarray) -> np.ndarray:
    n = Q.shape[0]
    I = np.eye(n)
    return np.linalg.inv(I - Q)


def expected_steps(N: np.ndarray) -> np.ndarray:
    ones = np.ones((N.shape[0], 1))
    return N @ ones


def expected_absorption_time(route: list[int], ai_level: int, dt: float = 1000.0
                             ) -> dict[int, float]:
    P, states = matrix_mod.build_transition_matrix(route, ai_level, dt)
    Q, R, transient_states = matrix_mod.partition_QR(P, states)

    N = fundamental_matrix(Q)
    t_steps = expected_steps(N)
    t_time = t_steps.flatten() * dt

    return dict(zip(transient_states, t_time))
```

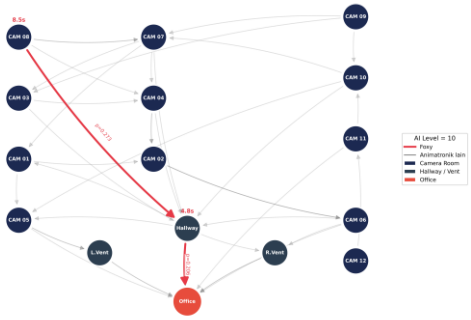
Gambar 8. Implementasi Program Expected Absorption Time

Pada tahap terakhir, submatriks Q yang telah diperoleh digunakan untuk menghitung *expected absorption time*. Fungsi fundamental_matrix() menghitung matriks $N = (I-Q)^{-1}$, kemudian expected_steps() mengalikan N dengan vektor satu untuk memperoleh ekspektasi banyaknya langkah sebelum diserap, dan terakhir expected_absorption_time() mengonversi hasil tersebut dari satuan langkah menjadi satuan waktu nyata dalam milidetik, sehingga diperoleh ekspektasi waktu yang dibutuhkan setiap animatronik untuk mencapai Office.

IV. Hasil dan Pembahasan

Perhitungan expected absorption time dilakukan pada AI Level = 10 untuk tiga animatronik yang merepresentasikan tiga karakteristik rute berbeda: Foxy dengan rute terpendek, Toy Bonnie dengan rute menengah, dan Mangle dengan rute terpanjang. Pemilihan ini dimaksudkan untuk melihat bagaimana panjang rute serta jumlah transient state memengaruhi waktu yang dibutuhkan suatu animatronik untuk mencapai Office.

A. Foxy



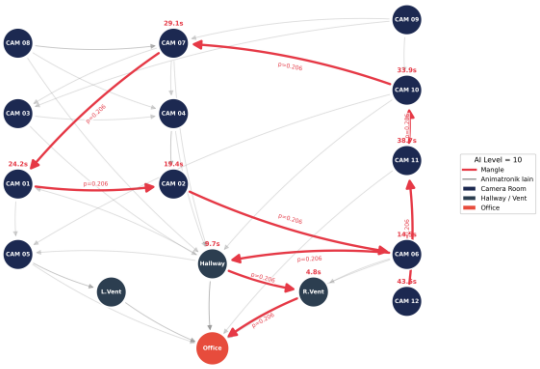
Gambar 9. Hasil Expected Absorption Time - Foxy

	8	101	104
8	0.727	0.273	0.000
101	0.000	0.794	0.206
104	0.000	0.000	1.000

Gambar 10. Matriks Transisi – Foxy

Foxy hanya melewati dua transient state, yaitu CAM08 dan Hallway, sebelum mencapai Office. Karena Foxy menggunakan fungsi transition_weight_foxy() yang berbeda dari animatronik lain, peluang berpindahnya dari CAM08 sebesar 0.273 per jendela observasi, lebih tinggi dibandingkan peluang 0.206 yang digunakan animatronik dengan rumus standar. Matriks transisi pada Gambar 10 menunjukkan struktur bidiagonal khas absorbing Markov chain dengan rute linear, di mana entri tak nol hanya muncul pada diagonal utama (probabilitas tetap pada state yang sama) dan diagonal di atasnya (probabilitas berpindah ke state berikutnya). Akibat rute yang pendek dan peluang berpindah yang relatif besar, Foxy memiliki expected absorption time tercepat di antara ketiga animatronik yang dianalisis, menjadikannya salah satu animatronik dengan tingkat ancaman tertinggi karena waktu tempuhnya menuju Office paling singkat.

B. Mangle



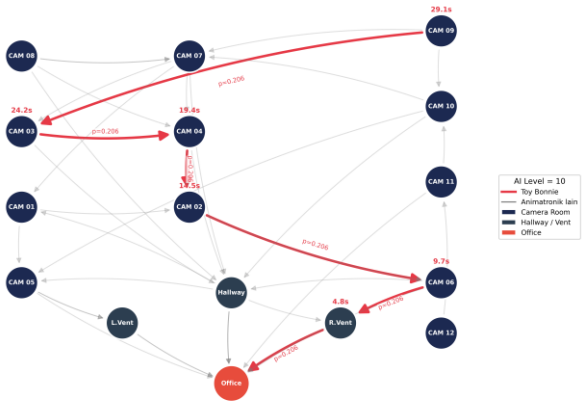
Gambar 11. Hasil Expected Absorption Time - Mangle

	12	11	10	7	1	2	6	101	102	104
12	0.794	0.206	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
11	0.000	0.794	0.206	0.000	0.000	0.000	0.000	0.000	0.000	0.000
10	0.000	0.000	0.794	0.206	0.000	0.000	0.000	0.000	0.000	0.000
7	0.000	0.000	0.000	0.794	0.206	0.000	0.000	0.000	0.000	0.000
1	0.000	0.000	0.000	0.000	0.794	0.206	0.000	0.000	0.000	0.000
2	0.000	0.000	0.000	0.000	0.000	0.794	0.206	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	0.000	0.794	0.206	0.000	0.000
101	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.794	0.206	0.000
102	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.794	0.206
104	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

Gambar 12. Matriks Transisi – Mangle

Mangle memiliki rute terpanjang di antara ketiga animatronik yang dianalisis, yaitu CAM12-CAM11-CAM10-CAM07-CAM01-CAM02-CAM06-Hallway-R.Vent-Office, dengan sembilan transient state. Matriks transisi pada Gambar 14 menunjukkan ukuran matriks yang jauh lebih besar dibandingkan Foxy maupun Toy Bonnie, namun tetap mempertahankan struktur bidiagonal yang sama karena setiap animatronik direpresentasikan sebagai rute linear menuju Office. Akibat banyaknya transient state yang harus dilewati, expected absorption time Mangle merupakan yang terbesar di antara ketiga animatronik, sejalan dengan karakteristik pergerakannya pada gim yang dikenal lebih berkeliaran dan menempuh jalur yang lebih jauh sebelum mendekati ruang penjaga.

C. Toy Bonnie



Gambar 13. Hasil Expected Absorption Time - Toy Bonnie

	9	3	4	2	6	102	104
9	0.794	0.206	0.000	0.000	0.000	0.000	0.000
3	0.000	0.794	0.206	0.000	0.000	0.000	0.000
4	0.000	0.000	0.794	0.206	0.000	0.000	0.000
2	0.000	0.000	0.000	0.794	0.206	0.000	0.000
6	0.000	0.000	0.000	0.000	0.794	0.206	0.000
102	0.000	0.000	0.000	0.000	0.000	0.794	0.206
104	0.000	0.000	0.000	0.000	0.000	0.000	1.000

Gambar 14. Matriks Transisi - Toy Bonnie

Toy Bonnie melewati rute CAM09-CAM03-CAM04-CAM02-CAM06-R.Vent-Office, terdiri atas enam transient state sebelum mencapai Office. Setiap transisi pada rute ini menggunakan peluang berpindah standar sebesar 0.206 dan peluang bertahan 0.794, sehingga matriks transisi pada Gambar 12 menunjukkan pola bidiagonal yang konsisten di sepanjang rute. Dengan jumlah transient state yang lebih banyak dibandingkan Foxy, expected absorption time Toy Bonnie berada pada posisi menengah, mencerminkan rute pergerakan yang lebih jauh dari Office meskipun peluang berpindah pada setiap sisinya tergolong sama.

D. Pembahasan

Secara umum, hasil pada ketiga animatronik menunjukkan bahwa expected absorption time berkorelasi positif dengan jumlah transient state pada rute masing-masing animatronik: semakin panjang rute yang harus dilalui, semakin besar pula ekspektasi waktu untuk mencapai Office. Struktur matriks transisi yang konsisten berbentuk bidiagonal pada seluruh animatronik mengonfirmasi bahwa representasi rute sebagai rantai Markov absorbing dengan bentuk kanonik QR berlaku secara seragam, terlepas dari panjang maupun bobot transisi tiap rute. Selain itu, perbedaan rumus bobot transisi pada Foxy turut memengaruhi laju pergerakannya, menunjukkan bahwa expected absorption time tidak hanya bergantung pada panjang rute, tetapi juga pada parameter probabilitas transisi yang spesifik untuk tiap animatronik. Dengan demikian, expected absorption time dapat digunakan sebagai indikator kuantitatif untuk membandingkan tingkat ancaman antar animatronik: nilai yang lebih kecil menandakan animatronik yang secara rata-rata mencapai Office lebih cepat dan karenanya berpotensi lebih berbahaya bagi pemain.

V. KESIMPULAN

Penelitian ini berhasil memodelkan pergerakan animatronik pada *Five Nights at Freddy's 2* sebagai graf berarah berbobot, yang kemudian direpresentasikan sebagai rantai Markov *absorbing* dengan *Office* sebagai *absorbing state*. Bobot transisi pada tiap sisi diturunkan langsung dari mekanisme waktu gerak pada *source code* permainan, sehingga model yang dihasilkan merefleksikan perilaku animatronik secara akurat sesuai *AI Level* yang digunakan.

Hasil perhitungan *expected absorption time* pada *AI Level* 10 menunjukkan bahwa panjang rute, yaitu jumlah *transient state* yang dilalui, berbanding lurus dengan waktu yang dibutuhkan suatu animatronik untuk mencapai *Office*. Foxy dengan rute terpendek memiliki waktu absorbsi tercepat, sedangkan Mangle dengan rute terpanjang memiliki waktu absorbsi terlama, dengan Toy Bonnie berada pada posisi menengah. Dengan demikian, dapat disimpulkan bahwa pendekatan rantai Markov pada graf berarah berbobot mampu memodelkan perilaku pergerakan setiap animatronik secara formal sekaligus berfungsi sebagai metode kuantitatif untuk mengukur dan membandingkan tingkat ancaman setiap animatronik dalam permainan, menjawab pertanyaan penelitian yang diajukan pada bagian pendahuluan.

VIDEO LINK YOUTUBE

https://youtu.be/wv_6PfL38xQ

UCAPAN TERIMA KASIH

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas rahmat dan karunia-Nya sehingga makalah ini dapat diselesaikan dengan baik. Penulis juga ingin menyampaikan rasa terima kasih yang tulus kepada Bapak Rinaldi Munir selaku dosen pengampu mata kuliah Matematika Diskrit, atas bimbingan, arahan, dan ilmu yang telah diberikan selama proses perkuliahan, yang sangat membantu penulis dalam menyusun makalah ini. Tidak lupa, penulis juga mengucapkan terima kasih kepada kanal YouTube Tech Rules, yang konten-kontennya menjadi sumber inspirasi bagi penulis dalam menentukan topik makalah ini. Akhir kata, penulis berharap makalah ini dapat memberi manfaat bagi pembaca dan menjadi kontribusi kecil bagi perkembangan ilmu pengetahuan.

REFERENSI

[1] Rinaldi M, "Graf (Bagian 1)," Bahan Kuliah IF1220 Matematika Diskrit, Program Studi Teknik Informatika STEI-ITB, 2026. [Daring]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf. [Diakses: 12 Juni 2026].

[2] D. A. Levin dan Y. Peres, *Markov Chains and Mixing Times*, 2nd ed. University of Oregon. [Daring]. Tersedia: https://pages.uoregon.edu/dlevin/MARKOV/markovmixing.pdf. [Diakses: 13 Juni 2026].

[3] M. Hajihashemi, "Multi-strategy evolutionary games: A Markov chain approach," *PLoS ONE*, vol. 17, no. 2, e0263979, Feb. 2022. [Daring]. Tersedia: https://pmc.ncbi.nlm.nih.gov/articles/PMC8853582/. [Diakses: 13 Juni 2026].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Juni 2026

Muhammad Pandu Pulunggana
13525059