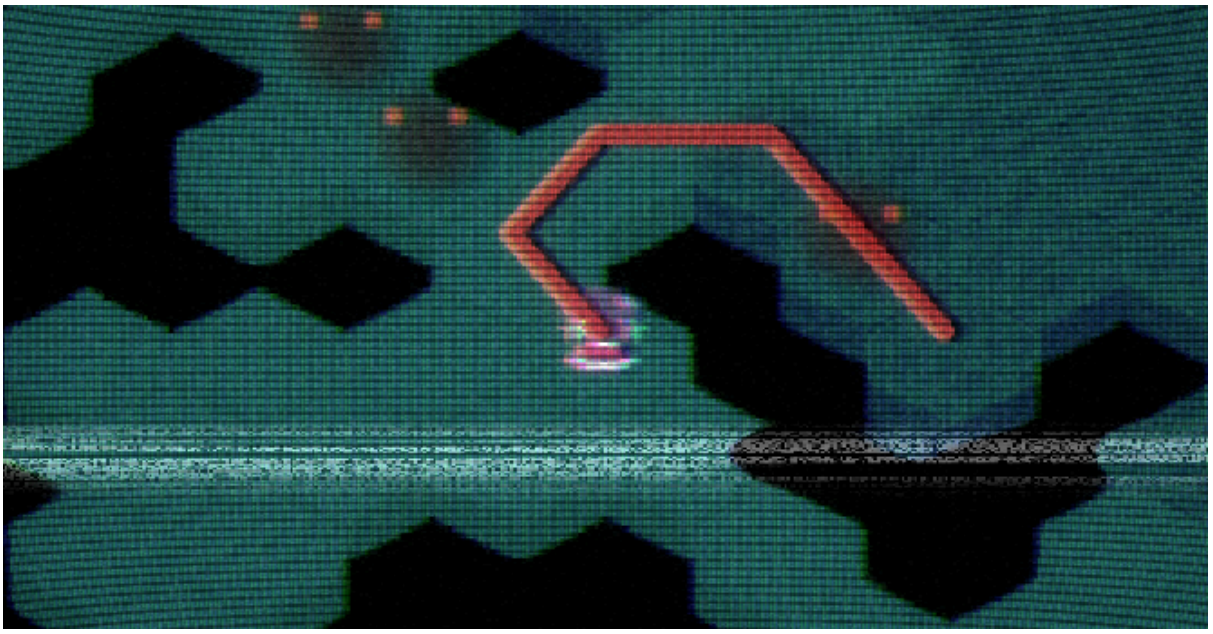
Penerapan Algoritma A* 2D Hexagonal (Diamond Isometric) di Video Permainan Bergenre Roguelike menggunakan Godot



Nama	**:**	**Vincent Timothy Kurniawan**
Kelas	**:**	**12.2**

Hexagonal movement adalah sistem pergerakan pada ruang/grid yang tersusun dari sel berbentuk segi enam (hexagon), di mana suatu objek dapat berpindah dari satu sel ke sel tetangga yang berbatasan langsung.

Berbeda dengan grid kotak yang memiliki 4 arah (atas, bawah, kiri, kanan) atau 8 arah (ditambah diagonal), pada grid hexagonal setiap sel hanya terhubung langsung ke enam sel di sekelilingnya. Akibatnya, pergerakan dilakukan dengan melangkah ke salah satu dari enam tetangga tersebut.

- ke kanan
- ke kanan-atas
- ke kiri-atas
- ke kiri
- ke kiri-bawah
- ke kanan-bawah

A* (dibaca: *A-star*) adalah algoritma pencarian jalur (pathfinding) yang digunakan untuk menemukan rute paling pendek atau paling efisien dari satu titik ke titik tujuan.

Algoritma ini sangat populer dalam pengembangan game, robotika, navigasi peta, dan kecerdasan buatan karena mampu mencari jalur optimal dengan cepat.

Rumus Utama:

f(n) = g(n) + h(n)

Keterangan:

- **g(n)** = biaya/jarak dari titik awal ke posisi sekarang
- **h(n)** = perkiraan jarak dari posisi sekarang ke tujuan (heuristic)
- **f(n)** = total nilai yang digunakan untuk menentukan jalur terbaik

Cara Kerja Utama:

1. Tentukan titik awal (start) dan titik tujuan (goal).
2. Mulai dari titik awal, periksa semua tetangga di sekitarnya.
3. Hitung nilai:
    - Jarak yang sudah ditempuh (g)
    - Perkiraan jarak ke tujuan (h)
    - Total nilai (f)
4. Pilih posisi dengan nilai f paling kecil.
5. Ulangi proses hingga mencapai tujuan.
6. Setelah tujuan ditemukan, jalur disusun kembali dari awal ke akhir.

Script ini merupakan sistem pergerakan player berbasis **A*** pada **tilemap hexagonal (diamond isometric)** yang terintegrasi dengan sistem gameplay, meliputi:

- Pathfinding A* pada grid hex
- Preview jalur sebelum bergerak
- Animasi squash & stretch saat berjalan
- Interaksi tile:
    - Ambil koin
    - Tabrak musuh
    - Pengurangan HP
- Validasi tile dapat dilalui

Inisiasi Variabel

```
@export var tileMap : TileMapLayer
@export var offSet = Vector2(0, -8)
@export var movement_speed := 0.1
@export var show_path_preview := true
@export var path_preview_color := Color(1, 1, 0, 0.8)
```

```
@export var allow_diagonal := true
```

Fungsi:

- tileMap → referensi ke tilemap hex
- offSet → penyesuaian posisi visual karakter
- movement_speed → kecepatan per tile
- show_path_preview → tampilkan garis jalur
- path_preview_color → warna preview jalur

Representasi Grid Hexagonal

Arah pergerakan hex diamond:

```
const HEX_DIRECTIONS_DIAMOND := [
     Vector2i(1, 0),    # East
     Vector2i(1, -1),   # Northeast
     Vector2i(0, -1),   # North
     Vector2i(-1, 0),   # West
     Vector2i(-1, 1),   # Southwest
     Vector2i(0, 1),    # South
]
```

Setiap tile memiliki 6 tetangga.

Algoritma A* (Fungsi Inti)

Method utama:

```
func _get_route_astar(targetPosition) -> Array[Vector2i]
```

Rumus: f(n) = g(n) + h(n)

Keterangan:

- g_score → biaya dari start ke node sekarang
- f_score → estimasi total biaya ke target
- open_set → node yang akan diperiksa
- came_from → menyimpan jalur

Method Utama: MOVE()

```
FUNCTION MOVE():
    IF isMoving = TRUE
        RETURN

    IF player_hp EXISTS AND player NOT alive
        RETURN

    targetPosition ← mouse global position
    IF targetPosition INVALID
        RETURN

    isMoving ← TRUE

    movementArray ← GET_ROUTE_ASTAR(targetPosition)

    IF movementArray EMPTY
        isMoving ← FALSE
        RETURN

    FOR EACH movement IN movementArray:
        playerTile ← tileMap.local_to_map(player.position)
        targetTile ← playerTile + movement
        targetWorldPos ← tileMap.map_to_local(targetTile) +
offset

        ANIMATE_MOVEMENT(player, targetWorldPos)

        CHECK_TILE_INTERACTIONS(targetTile)

        IF player NOT alive
            BREAK

    isMoving ← FALSE
END FUNCTION
```

Method: GET_ROUTE_ASTAR()

```
FUNCTION GET_ROUTE_ASTAR(targetPosition):
    start_tile ← tileMap.local_to_map(player.position)
    target_tile ← tileMap.local_to_map(targetPosition)
    open_set ← [start_tile]
    came_from ← empty dictionary
    g_score[start_tile] ← 0
    f_score[start_tile] ← HEX_DISTANCE(start_tile,
target_tile)
    iterations ← 0
    max_iterations ← 1000

    WHILE open_set NOT empty AND iterations < max_iterations:
        iterations++
        current ← node IN open_set WITH lowest f_score
        IF current = target_tile
            RETURN RECONSTRUCT_PATH(came_from, current)
        REMOVE current FROM open_set
        FOR EACH direction IN HEX_DIRECTIONS:
            neighbor ← current + direction
            IF NOT WALKABLE(neighbor)
                CONTINUE
            tentative_g ← g_score[current] + 1
            IF neighbor NOT IN g_score
               OR tentative_g < g_score[neighbor]:
                came_from[neighbor] ← current
                g_score[neighbor] ← tentative_g
                f_score[neighbor] ← tentative_g
                                    + HEX_DISTANCE(neighbor,
target_tile)
                IF neighbor NOT IN open_set
                    ADD neighbor TO open_set
    RETURN empty array
END FUNCTION
```

Perbandingan Algoritma Pathfinding

Algoritma	Cara Kerja	Kelebihan	Kekurangan	Cocok Untuk
A*	Menggunakan jarak asli + estimasi ke tujuan	Cepat, optimal, cerdas, fokus ke target	Perlu heuristic yang bagus	Game, AI, hex grid, navigasi
Dijkstra	Mengecek semua jalur dengan biaya terkecil	Pasti menemukan jalur terpendek	Lambat karena eksplorasi luas	Map kecil, sistem tanpa target jelas
BFS (Breadth First Search)	Menyebar ke semua arah selangkah demi selangkah	Sederhana, mudah dibuat	Tidak efisien di map besar	Grid kecil, tanpa bobot jarak
DFS (Depth First Search)	Menelusuri satu jalur sampai mentok	Sangat ringan	Tidak menjamin jalur terpendek	Eksplorasi sederhana
Greedy Best First	Hanya pakai estimasi jarak ke target	Sangat cepat	Bisa salah pilih jalur	AI sederhana
Theta*	Versi lanjutan A* yang lebih smooth	Jalur lebih natural	Lebih kompleks	Navigasi realistis

---

Kelebihan Algoritma Pathfiniding A*

1. Menemukan jalur terpendek (optimal), jika fungsi heuristic (h) akurat, A* akan selalu menemukan rute terbaik dari start ke goal.
2. Lebih cepat dari Dijkstra**,** karena A* menggunakan estimasi jarak ke tujuan, ia tidak mengecek semua kemungkinan jalur.
3. Efisien untuk game**,** sangat cocok untuk: NPC mengejar player, Karakter bergerak otomatis, Sistem taktik berbasis tile.
4. Fleksibel**,** bisa digunakan pada: Grid kotak, Grid hexagonal, Graph bebas
5. Mendukung obstacle/rintangan**,** tile yang tidak bisa dilewati langsung diabaikan oleh algoritma.

Kekurangan Algoritma Pathfinding A*

1. Bergantung pada heuristic, jika fungsi h(n) buruk: Jalur bisa tidak optimal, dan Pencarian bisa lebih lambat.
2. Memakan memori lebih besar karena menyimpan temporary variable yang cukup banyak: open_set, g_score, f_score, came_from
3. Bisa berat pada map sangat besar walaupun cepat, tetap bisa melambat jika map terlalu luas, dan banyak obstacle.
4. Lebih kompleks dibanding algoritma sederhana implementasi lebih rumit dibanding BFS, dan Djikstra.
5. Masih bisa mahal secara komputasi**,** jika target sangat jauh, A* tetap harus mengecek banyak node.