Penerapan Normalisasi Vektor untuk Pergerakan Delapan Arah pada Gim Video

Jonathan Levi - 13523132
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13523132@itb.ac.id, jonathanlevi12345678@gmail.com

***Abstract*—Pada sebuah gim video atau permainan video (*video game*)—terutama gim komputer—yang memerlukan pergerakan karakter, pergerakan tersebut pasti menggunakan tombol WASD dan/atau tanda panah pada *keyboard*. Namun, pernahkah Anda melihat atau merasakan pada gim tertentu yang mana saat bergerak secara diagonal, karakter tersebut bergerak lebih cepat daripada bergerak lurus? Istilah untuk teknik pergerakan diagonal ini disebut sebagai *straferunning* atau *trichording*. Gim seperti *Minecraft*, *Doom*, serial *Descent*, dan *Perfect Dark* memiliki kondisi seperti ini. Teknik *straferunning* dimanfaatkan oleh para *speedrunner*, terutama pada gim *Doom*, untuk menghemat waktu.**

***Keywords*—*diagonal speed boost*, normalisasi vektor, ruang Euclidean, *straferunning*.**

I. Pendahuluan

Dalam dunia *video game*, mekanika pergerakan karakter adalah salah satu aspek yang paling banyak digunakan. *Video game*—khususnya gim yang menggunakan *keyboard*—biasanya mengandalkan input seperti tombol WASD dan/atau tanda panah. *Game engine*—perangkat lunak yang berfungsi untuk membuat *video game*—seperti Godot sudah memberikan *script* atau kode untuk mengimplementasikan mekanik pergerakan karakter dengan mudah [1]. Namun, pada *game engine* lainnya—Unity dan Unreal Engine—, mekanika ini biasanya harus diimplementasikan sendiri.

Pengimplementasian mekanik tersebut sering memiliki inkonsistensi, terutama pada pergerakan diagonal. Saat menekan tombol W atau S dengan A atau D—tanda panah atas atau bawah dengan tanda panah kiri atau kanan—secara bersamaan, ada beberapa kasus yang mana pergerakan tersebut lebih cepat daripada pergerakan lurus. Kasus ini umumnya dikenal sebagai *straferunning*, atau *trichording* dalam beberapa gim tertentu.

Teknik ini dapat menguntungkan pemain gim tersebut, terutama bagi *speedrunner*—orang yang menyelesaikan suatu permainan dalam waktu secepat mungkin—, karena dapat menghemat waktu. Selain itu, teknik ini juga dapat dipakai untuk melarikan diri dari musuh dengan lebih mudah. Namun, teknik yang menguntungkan pemain ini justru dapat merugikan beberapa *game developer* (pengembang permainan video) karena kasus ini tidak dimaksudkan untuk ada di dalam gim. Para pengembang gim tidak menginginkan inkonsistensi pada permainan mereka.

Oleh karena itu, pada makalah ini, akan dibahas solusi yang sering digunakan oleh para pengembang gim untuk mengatasi masalah atau kasus pergerakan diagonal yang lebih cepat. Solusi yang paling sering digunakan oleh para pengembang gim adalah dengan cara menormalisasi vektor.

II. Teori Dasar

A. Vektor di Ruang Euclidean

Vektor adalah besaran yang memiliki nilai dan arah. Ada berbagai cara untuk merepresentasikan vektor dengan suatu simbol, di antaranya adalah $\vec{v}$ dan v. Ruang vektor R^n yang paling sering digunakan adalah vektor di R^2—vektor dua dimensi—dan vektor di R^3 —vektor tiga dimensi— [2]. Vektor yang akan dibahas adalah vektor dua dimensi, karena jumlah dimensi yang terlibat dalam pergerakan karakter pada konteks ini adalah dua dimensi.

1. Notasi dan Komponen Vektor

Penulisan vektor dapat disimbolkan sebagai berikut:

- (v_1, v_2)
- $\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$
- $\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$
- $v_1\vec{i} + v_2\vec{j}$

Vektor yang ditulis dengan bentuk seperti ini adalah vektor yang berawal dari titik O, atau dalam kata lain, vektor yang titik awalnya $(0,0)$ [2].

2. Panjang/Norma/*Magnitude* Vektor

Panjang vektor atau **norma vektor** atau ***magnitude* vektor** atau ***euclidean norm*** disimbolkan sebagai $||\vec{v}||$ atau $||v||$.

- Jika $\vec{v} = (v_1, v_2)$, maka:
$$||\vec{v}|| = \sqrt{v_1^2 + v_2^2}\,.$$
- Jika titik asal sebuah vektor bukan O, misalnya $\overrightarrow{P_1} = (x_1, y_1)$ dan $\overrightarrow{P_2} = (x_2, y_2)$, maka

$$||\overrightarrow{P_1P_2}|| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

[2].

3. Penjumlahan Vektor

Penjumlahan vektor adalah operasi menjumlahkan komponen ke-n setiap vektor. Misalnya, ada dua buah vektor

$$\vec{u} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix},$$
$$\vec{v} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix},$$

maka penjumlahan vektor-vektor tersebut adalah

$$\vec{u} + \vec{v} = \begin{pmatrix} u_1 + v_1 \\ u_1 + v_2 \end{pmatrix}$$

[2].

4. Pengurangan Vektor

Pengurangan vektor adalah operasi yang mana komponen ke-n pada suatu vektor dikurangi dengan komponen ke-n pada vektor berikutnya. Cara kerja pengurangan vektor sama dengan penjumlahan vektor. Namun, bedanya terdapat pada operasi matematika. Misalnya, ada dua buah vektor

$$\vec{u} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix},$$
$$\vec{v} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix},$$

maka pengurangan vektor-vektor tersebut adalah

$$\vec{u} - \vec{v} = \begin{pmatrix} u_1 - v_1 \\ u_1 - v_2 \end{pmatrix}$$

[2].

5. Perkalian Vektor

Perkalian Vektor adalah perkalian antara sebuah vektor dengan vektor lainnya atau dengan konstanta. Perkalian Vektor terbagi menjadi dua jenis, yaitu:

- Perkalian titik (*dot product*)
- Perkalian silang (*cross product*)

Perkalian titik (*dot product*) adalah operasi yang mengalikan komponen ke-n setiap vektor, lalu menjumlahkan hasil perkalian tersebut. Perkalian titik disimbolkan dengan tanda "·" di antara kedua buah vektor yang akan dikalikan titik. Misalnya, ada dua buah vektor

$$\vec{u} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix},$$
$$\vec{v} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix},$$

maka perkalian titik dari vektor-vektor tersebut adalah

$$\vec{u} \cdot \vec{v} = (u_1 \times v_1) + (u_2 \times v_2)$$

[2].

Perkalian silang (*cross product*) adalah operasi yang mengalikan kedua panjang vektor dengan sin dari sudut yang mengapit kedua vektor. Vektor dua dimensi tidak memiliki perkalian silang. Perkalian silang disimbolkan dengan tanda "×" di antara kedua buah vektor yang akan dikalikan silang. Perkalian ini hanya ditemukan pada dimensi nol, satu, tiga, dan tujuh [3].

6. Normalisasi Vektor

Normalisasi vektor adalah pembagian sebuah vektor dengan panjang atau *magnitude* vektor tersebut. Berdasarkan definisinya, proses normalisasi vektor dapat dirumuskan sebagai

$$\vec{u} = \frac{\vec{v}}{||\vec{v}||},$$

yang mana $\vec{v} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$.

Rumus tersebut dapat dijabarkan menjadi

$$\vec{u} = \left(\frac{v_1}{||\vec{v}||}, \frac{v_2}{||\vec{v}||}\right),$$

yang mana panjang vektor $||\vec{u}|| = 1$, yaitu

$$\left|\left|\vec{u}\right|\right| = \sqrt{u_1^2 + u_2^2} = 1\,,$$

atau lebih tepatnya

$$\sqrt{\left(\frac{v_1}{||\vec{v}||}\right)^2 + \left(\frac{v_2}{||\vec{v}||}\right)^2} = 1$$

[2].

Implementasi dari normalisasi (dan panjang) vektor dengan bahasa pemrograman Python adalah sebagai berikut:

```
# Python (Normalisasi Vektor)

def magnitude(vector):
    sqrOfVectorI = 0
    for i in range(len(vector)):
        sqrOfVectorI += vector[i]**2
        return round(sqrOfVectorI**0.5)

def normalized (vector):
    #Panjang vektor v
    IIvII = magnitude(vector)
    #Vektor u (Vektor satuan dari v)
    u = []
    for i in range(len(vector)):
        u.append(vector[i]/IIvII)
    return u

#vektor v
while True:
    v = input("Masukkan bilangan,
dipisah dengan koma: ").replace(" ",
"").split(",")
    valid = True
    for i, num in enumerate(v):
        if num.isdigit():
            v[i] = int(num)
        else:
             valid = False
             break
    if valid:
        break

#Contoh penggunaan:
#u = normalized(v)
#print(f"u = {u}")
#IIvII = magnitude(v)
```

```
#print(f"||v|| ≈ {IIvII}")
#IIuII = magnitude(u)
#print(f"||u|| = {IIuII}")
```

III. Pengimplementasian Aljabar Vektor pada Pergerakan dalam Gim Video

A. Teorema Pythagoras

Teorema Pythagoras adalah sebuah teorema yang menyatakan bahwa pada segitiga siku-siku, kuadrat dari panjang sisi miring adalah jumlah dari kuadrat sisi depan dan kuadrat sisi samping—dalam kata lain, kedua sisi yang saling tegak lurus—. Definisi teorema ini dapat dirumuskan sebagai

$$c^2 = a^2 + b^2 ,$$

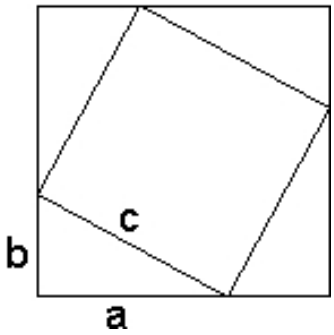
yang dapat ditulis sebagai

$$c = \sqrt{a^2 + b^2} ,$$

yang mana:

$$a, b: \textit{kedua sisi yang saling tegak lurus}$$
$$c: \textit{sisi miring}$$

Teorema ini dapat dibuktikan dengan cara yang sangat banyak—bahkan ada yang membuktikannya dengan 122 cara—[4]. Salah satu cara termudah menggunakan geometri adalah dengan menggunakan bentuk di bawah ini:



Gambar 3.1. Pembuktian Teorema Pythagoras menggunakan geometri. Sumber: https://www.cut-the-knot.org/pythagoras

Gambar tersebut merupakan gabungan empat buah segitiga siku-siku yang disusun sedemikian rupa, sehingga sisi luarnya membentuk persegi berlubang, lubang tersebut juga membentuk persegi miring.

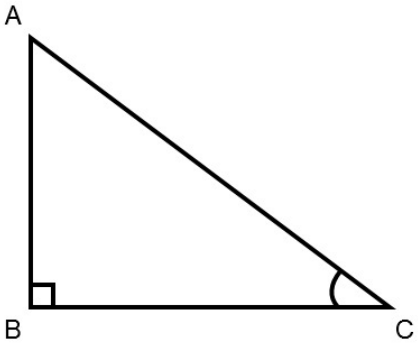
$$(a+b)^2 - c^2 = 4 \times \frac{ab}{2}$$
$$(a^2 + 2ab + b^2) - 2ab = c^2$$
$$\boldsymbol{a^2 + b^2 = c^2}$$

Dasar perhitungannya adalah luas persegi berlubang = luas keempat segitiga. Dalam kata lain:

$$\mathrm{A}_{\text{persegi}} - \mathrm{A}_{\text{lubang persegi}} = \mathrm{A}_{\text{empat segitiga}}$$

[4].

Metode lainnya adalah dengan memanfaatkan vektor, misalnya dengan segitiga siku-siku ABC seperti berikut:



Gambar 3.2. Pembuktian Teorema Pythagoras menggunakan vektor segitiga siku-siku. Sumber: Dokumen Pribadi

Dari gambar tersebut, dapat dilihat bahwa

$$||\overrightarrow{AC}|| = ||\overrightarrow{AB}|| + ||\overrightarrow{BC}|| ,$$

sehingga

$$||\overrightarrow{AC}||^2 = (\overrightarrow{AB} + \overrightarrow{BC}) \cdot (\overrightarrow{AB} + \overrightarrow{BC})$$
$$||\overrightarrow{AC}||^2 = (\overrightarrow{AB} \cdot \overrightarrow{AB}) + (\overrightarrow{AB} \cdot \overrightarrow{BC}) + (\overrightarrow{BC} \cdot \overrightarrow{AB}) + (\overrightarrow{BC} \cdot \overrightarrow{BC}) .$$

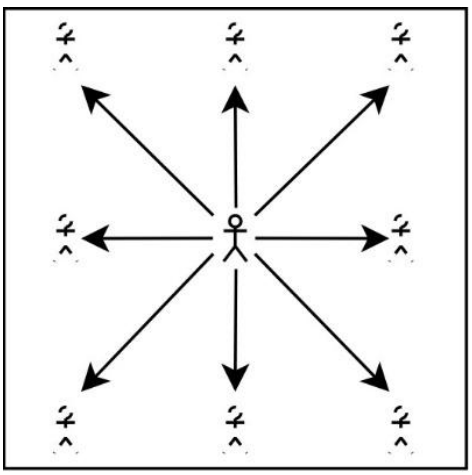
Namun, $\overrightarrow{AB}$ dan $\overrightarrow{BC}$ tegak lurus, sehingga $(\overrightarrow{AB} \cdot \overrightarrow{BC}) = (\overrightarrow{BC} \cdot \overrightarrow{AB}) = 0$. Maka

$$||\overrightarrow{AC}||^2 = (\overrightarrow{AB} \cdot \overrightarrow{AB}) + (\overrightarrow{BC} \cdot \overrightarrow{BC})$$
$$\boldsymbol{||\overrightarrow{AC}||^2 = ||\overrightarrow{AB}||^2 + ||\overrightarrow{BC}||^2}$$

[5].

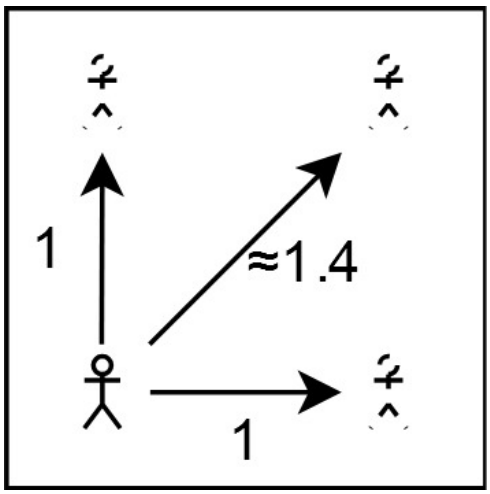
B. Hubungan Teorema Pythagoras dengan Masalah Pergerakan Diagonal pada Gim Video

Agar lebih mudah memahami masalah pada pergerakan diagonal, akan diberikan gambaran awal mengenai pergerakan di dalam gim terlebih dahulu. Pergerakan dalam konteks ini memiliki delapan arah, yaitu kiri, kanan, depan atau atas, belakang atau bawah, dan gabungan antara kiri atau kanan dengan depan/atas atau belakang/bawah. Untuk gambaran lebih jelasnya, dapat dilihat di ilustrasi di bawah ini.



Gambar 3.3. Ilustrasi Pergerakan Delapan Arah pada Gim Video (Sebelum Normalisasi). Sumber: Dokumen Pribadi

Berdasarkan gambar di atas, diambil sebuah contoh arah pergerakan. Misalnya, karakter bergerak ke arah kanan depan atau kanan atas.



Gambar 3.4. Ilustrasi Teorema Pythagoras pada Pergerakan Karakter Gim Video ke Arah Tertentu.
Sumber: Dokumen Pribadi

Dapat dilihat alasan mengapa karakter bergerak lebih cepat jika bergerak secara diagonal. Dapat dikatakan, dalam satu satuan waktu, jika karakter bergerak ke kanan sebanyak satu satuan panjang dan bergerak ke atas atau depan sebanyak satu satuan panjang secara bersamaan, maka

$$Jarak\ diagonal = \sqrt{1^2 + 1^2}$$
$$Jarak\ diagonal = \sqrt{2}$$
$$\boldsymbol{Jarak\ diagonal \approx 1{,}4}.$$

Dengan menggunakan teorema Pythagoras seperti di atas, dapat dilihat bahwa selama satu satuan waktu, karakter bergerak ke kanan atas atau kanan depan sebanyak 1,4 satuan panjang.

C. Penyelesaian Masalah Kecepatan Pergerakan Diagonal pada Gim

Untuk pemgembang gim yang menggunakan Godot Engine, hal ini bukanlah sebuah masalah, karena vektor pergerakan diagonal sudah otomatis "dinormalisasi". Berikut *script*—kode untuk menjalankan suatu fungsi—pergerakan delapan arah yang diberikan pada dokumentasi Godot dalam bahasa GDScript yang mirip dengan Python:

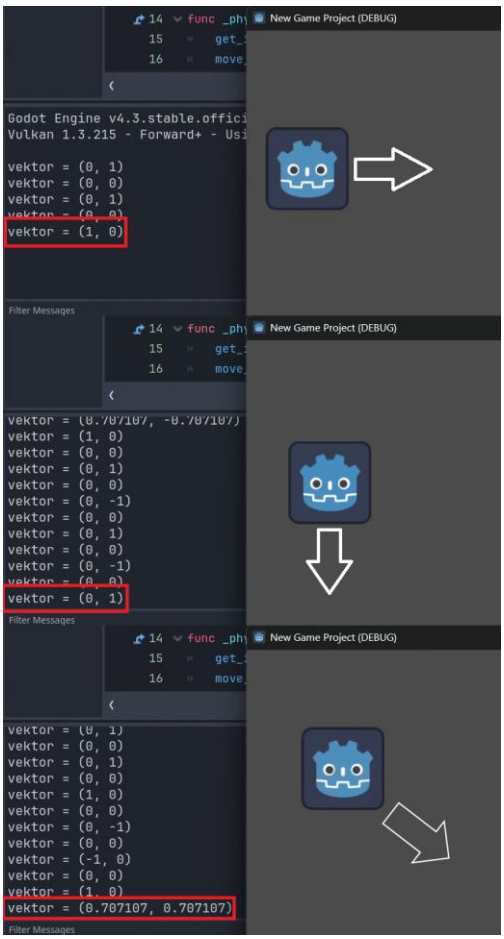
```
# GDScript (Godot)

extends CharacterBody2D

@export var speed = 400

func get_input():
    var input_direction =
Input.get_vector("left", "right", "up",
"down")
    velocity = input_direction * speed

func _physics_process(delta):
    get_input()
    move_and_slide()
```

Selain itu, Godot juga mendukung bahasa pemrograman C#, dengan *script* berikut:

```
/* C# (Godot) */

using Godot;

public partial class Movement :
CharacterBody2D {
    [Export]
    public int Speed { get; set; } =
400;

    public void GetInput() {
        Vector2 inputDirection =
Input.GetVector("left", "right", "up",
"down");
        Velocity = inputDirection *
Speed;
    }

    public override void
_PhysicsProcess(double delta) {
        GetInput();
        MoveAndSlide();
    }
}
```

Perhatikan kedua *script* tersebut [1]. Fungsi—*function*— `Input.get_vector()` (GDScript) atau `Input.GetVector()` (C#) yang diberikan oleh Godot sudah secara otomatis mengatasi masalah pergerakan diagonal. Fungsi `Input.get_vector()` cara kerjanya sebenarnya bukan menormalisasi vektor, melainkan membatasi panjang vektor agar tidak melebihi satu. Ini berarti, jika panjang vektor vertikal dan panjang vektor horizontal sangat kecil, panjang vektor diagonal bisa saja melebihi panjang salah satu vektor tersebut. Namun, fungsi `Input.get_vector()` secara otomatis menetapkan panjang vektor sebagai satu, sebagaimana acuan panjang vektor yang biasanya digunakan dalam pengembangan gim adalah satu [6]. Akan tetapi, tidak ada salahnya menambahkan fungsi `normalized()` dalam fungsi `get_input()`, sehingga hasil akhir *script* adalah `velocity = input_direction.normalized() * speed` (GDScript) atau `Velocity = inputDirection.Normalized() * Speed;` (C#).

Berikut hasil percobaan pergerakan karakter delapan arah pada Godot Engine, beserta dengan vektornya:



Gambar 3.5. Hasil percobaan pergerakan karakter dan vektornya pada Godot Engine. Sumber: Dokumen Pribadi (Video Percobaan: https://youtu.be/oON4SndMuR0)

Perhatikan bahwa sumbu y pada Godot negatif saat menuju ke atas dan positif saat menuju ke bawah. Ini karena pada *computer science*, sumbu y umumnya menuju ke bawah, yaitu y positif dimulai dari atas, menuju ke bawah layar. Hal ini disebabkan oleh tradisi atau standar yang sudah berlaku sejak lama. Ada teori yang mengatakan bahwa standar ini berasal dari monitor CRT (*Cathode Ray Tube*). Cara kerja CRT adalah menggambarkan layar dari kiri ke kanan, atas ke bawah [8].

Jika menggunakan *game engine* lainnya, seperti Unity, pengembang gim biasanya perlu menulis sebuah *script* pergerakan karakter secara manual. Letak kesalahan yang sering dilakukan oleh pengembang gim pemula terdapat pada *script* yang mereka tulis, karena *script* tersebut sering tidak menormalisasikan vektornya. Bahkan ada video tutorial di YouTube yang mana *script* pergerakan karakternya adalah seperti berikut:

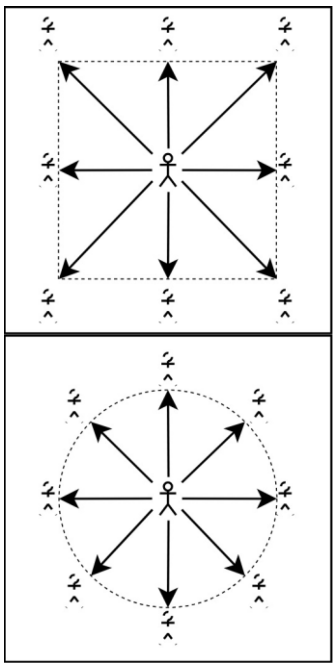
```
/* C# (Unity) */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMovement :
MonoBehaviour {
    private float speed = 1f;
    Vector2 movement;
    public Rigidbody2D rb;

    Void Update() {
        movement.x =
Input.GetAxisRaw(“Horizontal”);
        movement.y =
Input.GetAxisRaw(“Vertical”);
    }

    Void FixedUpdate() {
        rb.MovePosition(rb.position +
movement * speed *
Time.fixedDeltaTime);
    }
}
```

Perhatikan *script* tersebut [9]. Terdapat suatu kesalahan dalam salah satu fungsi bertipe `Void` pada *script* tersebut, yaitu pengembang gim tidak menggunakan fungsi yang menormalisasi vektor pergerakannya, sehingga pergerakan karakter secara diagonal menjadi lebih cepat daripada seharusnya. Untuk mengatasi hal tersebut, *script* yang benar seharusnya adalah sebagai berikut:

```
/* C# (Unity) */

/* ... */

    Void FixedUpdate() {
        rb.MovePosition(rb.position +
movement.normalized * speed *
Time.fixedDeltaTime);
    }
}
```

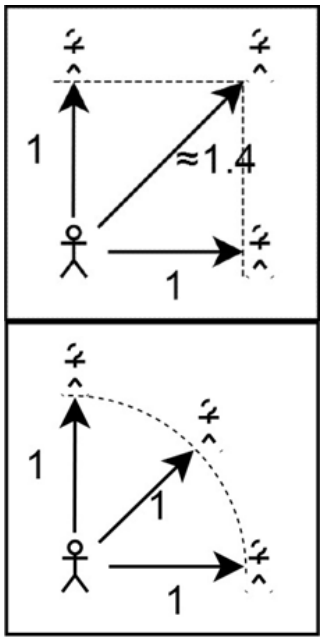
Perhatikan fungsi `FixedUpdate()` tersebut [10]. Sekarang, masalah mengenai pergerakan karakter secara diagonal sudah diselesaikan, karena variabel `movement` telah dinormalisasikan dengan fungsi `normalized`. Perhatikan juga bahwa fungsi `normalized` berbeda dengan `Normalize()` yang juga terdapat pada Unity [11]. Fungsi `Normalize()` digunakan untuk memodifikasi secara langsung vektor yang diberikan, sehingga vektor yang sebelumnya tidak tersimpan (fungsi tersebut tidak memakai `return`), sedangkan fungsi `normalized` dapat menyimpan vektor yang dinormalisasi ke dalam variabel lainnya (memakai `return`; contohnya `u = v.normalized`). Dalam kata lain, fungsi `v.Normalize()` setara dengan `v = v.normalized`.

Berikut adalah perbandingan visualisasi antara sebelum dan sesudah dilakukan normalisasi vektor:



Gambar 3.6. Perbandingam Pergerakan Delapan Arah pada Gim Video Sebelum dan Setelah Normalisasi Vektor. Sumber: Dokumen Pribadi

Jika diambil contoh arah sebelumnya, yaitu jika karakter bergerak ke arah kanan depan atau kanan atas, maka ilustrasinya adalah seperti berikut:



Gambar 3.7. Perbandingan Pergerakan Karakter ke Arah Tertentu Sebelum dan Setelah Normalisasi Vektor. Sumber: Dokumen Pribadi

Perhitungan matematikanya adalah sebagai berikut:

$$\vec{v}_{(vektor\ diagonal)} = (1,1)$$

$$||\vec{v}|| = \sqrt{1^2 + 1^2} = \sqrt{2}$$

$$\vec{u} = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right) \approx (0{,}707, 0{,}707)$$

$$||\vec{u}|| = \sqrt{\left(\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\right)^{\mathbf{2}} + \left(\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\right)^{\mathbf{2}}} = \sqrt{\frac{\mathbf{1}}{\mathbf{2}} + \frac{\mathbf{1}}{\mathbf{2}}} = \mathbf{1}$$

Hal tersebut berarti bahwa setelah dilakukan normalisasi vektor, saat karakter dalam permainan video bergerak secara diagonal, karakter tersebut akan berpindah sebanyak satu satuan panjang dalam satu satuan waktu. Vektor $\vec{u}$ tersebut juga sama dengan vektor pada gambar 3.5 saat karakter bergerak secara diagonal, sehingga terbukti bahwa pergerakan diagonal karakter telah ternormalisasi.

IV. Kesimpulan

Normalisasi vektor dapat digunakan untuk memecahkan masalah mengenai kecepatan pergerakan diagonal pada permainan video, karena panjang normalisasi vektor selalu sama dengan satu. Panjang normalisasi vektor ini sama dengan acuan panjang vektor horizontal dan acuan panjang vektor vertikal. Panjang vektor sama dengan jarak pergerakan karakter per satuan waktu.

V. Referensi

[1] "2D movement overview," *Godot Engine Documentation*, Aug. 2024. https://docs.godotengine.org/en/stable/tutorials/2d/2d_movement.html (accessed Dec. 25, 2024).

[2] H. Anton and C. Rorres, *Elementary Linear Algebra: Applications Version*, 11th ed. John Wiley & Sons, Inc, 2013, ch. 3.

[3] User122283, "Why is cross product only defined in 3 and 7 dimensions?," *Mathematics Stack Exchange*, Mar. 10, 2014. https://math.stackexchange.com/a/706044 (accessed Dec. 27, 2024).

[4] A. Bogomolny, "Pythagorean Theorem and its many proofs," *Cut the Knot*, 2016. https://www.cut-the-knot.org/pythagoras/#4 (accessed Dec. 26, 2024).

[5] T. Wagner, "How to prove the Pythagoras theorem using vectors," *Mathematics Stack Exchange*, Nov. 23, 2010. https://math.stackexchange.com/a/11516 (accessed Dec. 27, 2024).

[6] "Input," *Godot Engine Documentation*, Aug. 2024. https://docs.godotengine.org/en/stable/classes/class_input.html#class-input-method-get-vector (accessed Dec. 28, 2024).

[7] "Vector2," *Godot Engine Documentation*, Aug. 2024. https://docs.godotengine.org/en/stable/classes/class_vector2.html#class-vector2-method-normalized (accessed Dec. 29, 2024).

[8] User2672165, "Why does the Y coordinate increase downwards?," *Stack Overflow*, Apr. 09, 2014. https://stackoverflow.com/a/22972083/23528590 (accessed Jan. 02, 2025).

[9] Brackeys, "TOP DOWN MOVEMENT in Unity!," *YouTube*. Aug. 12, 2019. Accessed: Dec. 28, 2024. [YouTube Video]. Available: https://www.youtube.com/watch?v=whzomFgjT50

[10] "Vector2.normalized," *Unity Documentation*, Dec. 27, 2024. https://docs.unity3d.com/ScriptReference/Vector2-normalized.html (accessed Dec. 28, 2024).

[11] "Vector2.Normalize," *Unity Documentation*, Dec. 27, 2024. https://docs.unity3d.com/ScriptReference/Vector2.Normalize.html (accessed Dec. 29, 2024).

VI. Lampiran

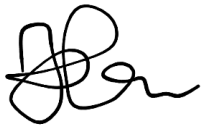
- Video demonstrasi singkat mengenai pergerakan delapan arah pada karakter video gim yang sudah ternormalisasi, beserta vektornya: https://youtu.be/oON4SndMuR0.
- File *project* Godot (perlu memasang aplikasi Godot; dibuat dan dicoba pada Godot versi 4.3): https://drive.google.com/file/d/1ASH49ILfTWF1FD

ERh06Kdlf-K2o1Togh.

VII. 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, 29 Desember 2024



Jonathan Levi, 13523132