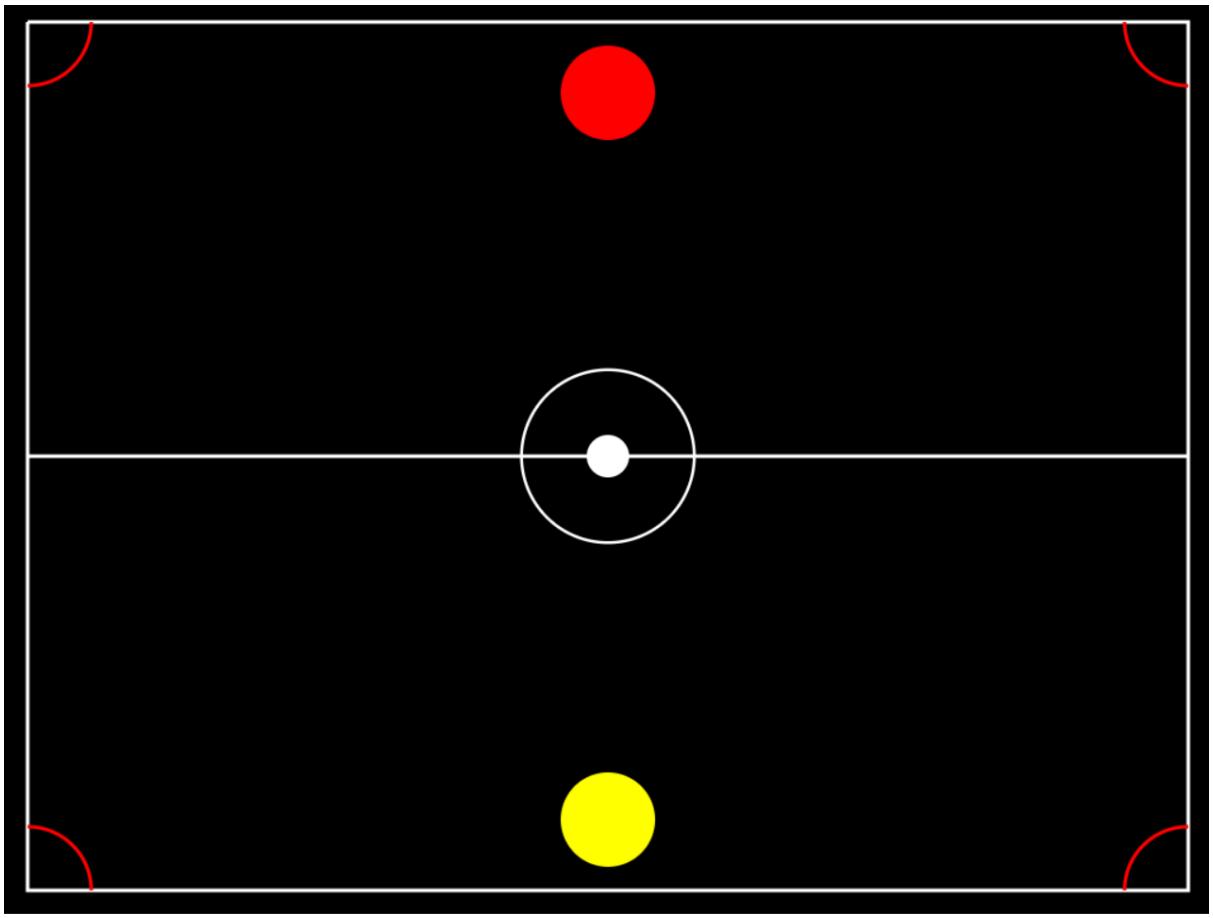
Laporan Projek
“Collision Circle Bounce” Simulation

Oleh: Sofyan Gio Verdiansyah (25122001)



Pembimbing dan Penguji:
Misbah Habib Putra

Program Studi:
Teknik Robotika dan Kecerdasan Buatan

Pada tahun:
2025

Daftar Pustaka

Kata Pengantar

Puji dan syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya, penulis dapat menyelesaikan laporan proyek yang berjudul **"Collision Circle Simulation"** dengan baik dan tepat waktu.

Laporan ini disusun sebagai salah satu tugas proyek mata kuliah **Fisika Dasar**, dengan tujuan untuk menerapkan konsep-konsep dasar fisika, khususnya **Hukum Newton II** dan **Gerak Lurus Berubah Beraturan (GLBB)**, ke dalam bentuk simulasi interaktif berupa permainan (*game*) dua dimensi.

Dalam penyusunan laporan ini, penulis menyadari masih terdapat banyak kekurangan. Oleh karena itu, penulis mengharapkan saran dan kritik yang membangun agar laporan ini dapat lebih baik di masa mendatang.

Penulis juga menyampaikan terima kasih kepada:

1. Bapak Misbah Habib Putra, selaku dosen pengampu mata kuliah **Fisika Dasar**, yang telah memberikan bimbingan dan arahan selama proses pembuatan proyek in
2. Teman-teman yang telah memberikan bantuan, ide, serta motivasi selama proses pengembangan proyek ini.

Akhir kata, semoga laporan ini dapat memberikan manfaat dan menjadi referensi dalam memahami penerapan konsep fisika dasar melalui media digital interaktif.

Bab I
Pendahuluan

Latar Belakang

Proyek ini dibuat sebagai bagian dari tugas akhir setengah semester mata kuliah Fisika Dasar, dengan tujuan agar mahasiswa tidak hanya memahami teori saja, tetapi juga melihat penerapannya langsung dalam konteks digital melalui pemrograman. Dengan mengimplementasikan konsep **GLBB** dan **Hukum Newton II** dalam game atau simulasi, diharapkan pemahaman terhadap dinamika gerak benda menjadi lebih mudah dipahami.

Rumusan Masalah

Projek ini memiliki poin-poin permasalah sebagai berikut:

1. Bagaimana mengimplementasikan simulasi fisika interaktif menggunakan bahasa pemrograman JavaScript dan HTML5 Canvas?
2. Bagaimana cara menerapkan konsep Hukum Newton II ($F = m \cdot a$) pada sistem pergerakan bola dalam game berbasis fisika 2D?
3. Bagaimana konsep Gerak Lurus Berubah Beraturan (GLBB) dapat digunakan untuk mengatur percepatan dan kecepatan bola selama permainan berlangsung?
4. Bagaimana pengaruh gaya dorong dan gaya gesekan terhadap perubahan kecepatan dan arah gerak bola?
5. Bagaimana merancang algoritma bot sederhana yang dapat merespons posisi bola dengan logika berbasis aturan (rule-based reactive movement)?

Manfaat

Dengan dibuatnya proyek ini, diharapkan dapat memberikan manfaat baik bagi mahasiswa maupun penulis dalam memahami dan menerapkan konsep-konsep dasar Fisika, khususnya Hukum Newton II dan Gerak Lurus Berubah Beraturan (GLBB) yang dipelajari pada pertengahan semester ini. Selain memperdalam pemahaman terhadap teori fisika, proyek ini juga diharapkan dapat membantu dalam melatih kemampuan berpikir secara logika pemrograman.

Bab II
Landasan Teori dan Konsep

Projek ini dilandasi oleh beberapa teori dan konsep di antara lainnya adalah berikut:

1. Collision (Tumbukan)

Collision disini atau lebih tepatnya Circle Collision digunakan untuk mengukur apakah player dan bola akan saling bertumbukan atau bertabrakan.

Rumusnya:

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

d : Jarak antara kedua titik pusat.
(x_1, y_1) : posisi lingkaran pertama
(x_2, y_2) : posisi lingkaran kedua

Jika nilai d

$$d \leq r_1 + r_2$$

r_1: Jari-jari lingkaran pertama
r_2: Jari-jari lingkaran kedua

Maka Player dan bola mengalami tumbukan, pada moment ini kita bisa melakukan perhitungan seperti Kinematika(GLBB) atau Dinamika(Hukum Newton II).

2. Vector Normal dan Normalisasi

Merupakan perhitungan arah, gaya, pantulan, dan tabrakan dalam simulasi game ini. Rumus ini saya dapatkan ketika saya mengalami permasalahan dalam perhitungan jarak dimana saya langsung memasukkan angka dari tiap object ke rumus Fisika.

Vector Normal
Misal ada vector

$$A(x_{1,}y_{1,}) \text{ dan } B(x_{2,}y_{2,})$$

Maka vector dari A ke B dapat di tulis:

$$\vec{r} = (x_2 - x_1, y_2 - y_1)$$

Arah vektor ini menunjuk dari pusat A ke pusat B, jadi kalau bola menabrak pemain, arah itulah arah gaya menuju tumbukan.

Vektor ini disebut vektor normal (arah normal) terhadap bidang kontak atau garis yang menghubungkan pusat dua lingkaran.

Normalisasi

Normalisasi artinya membuat vektor memiliki panjang 1 (satuan).
Misal panjang vector $\vec{r}$ atau adalah:

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

Maka **unit normal** (vektor arah dengan panjang 1) adalah:

$$\hat{n} = \frac{\vec{r}}{|\vec{r}|} = \left(\frac{x_2 - x_1}{|\vec{r}|}, \frac{y_2 - y_1}{|\vec{r}|}\right)$$

Memastikan

Karena dalam fisika, sering kali kita mengetahui besarnya gaya (misalnya 10 N) dan arahnya (misalnya "ke kanan atas"), tapi arah itu harus diwakili dalam bentuk matematis.

Agar komputer (atau rumus) bisa mengerti arah tersebut, kita ubah arah itu menjadi vektor satuan:

$$\hat{n} = \frac{\vec{r}}{|\vec{r}|}$$

yang panjangnya 1:

$$|\hat{n}| = 1$$

karena vektor satuan selalu memiliki panjang 1 secara definisi.

Kemudian kita baru bisa menghitung gaya totalnya:

$$\vec{F} = F\hat{n}$$

Artinya:

- Panjang (magnitudo) dari $\vec{F}$= F
- Arah dari $\vec{F}$= sama dengan arah $\hat{n}$

Rumus diatas didapat dari definisi arah gaya dalam koordinat dua dimensi atau tiga dimensi. Misal saya punya gaya dorong 90N dan mengarah ke depan, namun arah yang saya maksud tidak bisa di tulis langsung dengan (nx, ny).
Karena sudah di normalisasikan dimana panjangnya harus 1, maka langsung bisa di kalikan dengan rumus di atas.

Mengapa perlu di normalisasikan?

Jika tidak dinormalisasikan maka panjang vektor ($|\vec{r}|$) akan ikut memengaruhi gaya, padahal gaya seharusnya hanya tergantung **arah**, bukan jarak antar benda.

Contoh:

$$A(5,5), \qquad B(2,2)$$

Di dapat:

$$\vec{r} = (-3,-3), \qquad |\vec{r}| \approx 4.243, \qquad \hat{n} = (-0.707,-0.707)$$

Dan panjang vector adalah:

$$|\hat{n}| = \sqrt{(-0.707)^2 + (-0.707)^2} = 1$$

Contoh penerapan pada gaya:

Misal $F = 100$ dengan rumusnya $\vec{F} = F\hat{n}$
Jika tanpa normalisasi:

$$\vec{F} = 100 \cdot (-3,-3) = (-300,-300)$$
$$\left|\vec{F}\right| = \sqrt{(-300)^2 + (-300)^2} = 424.3N$$

Hasilnya gaya menjadi berubah jika tidak di normalisasikan.
Sedangkan jika dinormalisasikan:

$$\vec{F} = 100 \cdot (-0.707,-0.707) = (-70.7,-70.7)$$
$$\left|\vec{F}\right| = \sqrt{(-70.7)^2 + (-70.7)^2} = 100N$$

Jadi player akan tetap membawa gaya dorong 100N dimanapun vector player dan bola berada.

3. Koreksi Overlap

Jika kedua lingkaran yakni player dan bola saling tumpang tindih atau overlap maka kita bisa mencegahnya dengan cara:

$$o_{AB} = r_A + r_B - d$$

$$\vec{p}_B \leftarrow \vec{p}_B + (\hat{n} \cdot o_{AB})$$

Arti:

o_{AB} : Hasil nilai perhitungan Overlap antar A dan B
r : adalah radius baik A dan B
d : Jarak titik temu / collision
$\vec{p}_B$: Posisi baik X atau Y nantinya

Note: B karena adalah target yang perlu di koreksi

Contoh:

$$A(5,5)\ r = 10, \qquad B(8,5)\ r = 5$$

$$d = \sqrt{(8-5)^2 + (5-5)^2} = 3$$

$$o_{AB} = 10 + 5 - 9 = 6$$

$$\hat{n} = \left(\frac{8-5}{3}, \frac{5-5}{3}\right) = (1,0)$$

Karena y 0 maka kita cukup koreksi titik x saja:

$$8 + (1 \cdot 6) = 14$$

Jadi nantinya titik x bola harus pindah ke titik 14 agar tidak terjadi overlap dengan player.

4. Kinematika GLBB

Dimana kinematika adalah bidang ilmu yang mempelajari bagaimana gerak suatu benda tanpa memperhatikan penyebabnya, dan jika memperhatikan penyebabnya di sebut Dinamika.

Dalam Kinematika ada macam-macam gerak lurus dan gerak umum benda, diantara lainnya adalah:

- Gerak Lurus Beraturan (GLB)
- Gerak Lurus Berubah Beraturan (GLBB)
- Gerak Melingkar Beraturan (GMB)
- Gerak Melingkar Berubah Beraturan (GMBB)
- Gerak Parabola

Fokus kita adalah GLBB dimana GLBB adalah gerak lurus di mana kecepatan benda berubah secara tetap setiap satuan waktu. Perubahan kecepatan bisa berubah tergantung bertambah atau berkurang:

- GLBB dengan **percepatan** tetap: kecepatan meningkat setiap detik.
- GLBB dengan **perlambatan** tetap: kecepatan menurun setiap detik.

GLBB berbeda dengan **GLB,** di mana kecepatan konstan dan tidak ada percepatan.

Perbedaan kecepatan dan percepatan:

- **Kecepatan** adalah seberapa cepat dan arah mana suatu benda dari satu posisi ke posisi lain. Dimana rumusnya adalah:

  $$v = \frac{s}{t}$$

  v : Kecepatan (m/s)

s : Jarak (m)
t : Waktu (s)

Inti maknanya adalah:
Kecepatan hanya menunjukkan perubahan posisi terhadap waktu.

> Contoh mobil bergerak 10 meter setiap 2 detik, jadi kecepatannya adalah 5m/s

- **Percepatan** adalah seberapa cepat kecepatan benda berubah tiap detik.

$$a = \frac{v_0 - v_1}{t}$$

a : Percepatan (m/s^2)
v_0 dan v_1: kecepatan awal dan akhir

Inti makna:
Jika percepatan positif, benda makin cepat.
Jika percepatan negatif (perlambatan), benda makin lambat.

> Contoh pada mobil diam (0 m/s) lalu dalam 3 detik kecepatannya menjadi 30 m/s. Jadi percepatannya adalah 10 m/s.

Setelah mengetahui perbedaan kecepatan dan percepatan berikut adalah rumus GLBB yang akan di terapkan:

a) **Kecepatan Akhir**

$$v = v_0 + a \cdot t$$

Setiap detik, kecepatan bertambah sebesar a. Jadi jika percepatannya tetap, maka pertambahan kecepatan juga tetap.

> Contoh sebuah mobil mula-mula diam ($v_0 = 0$) dan dipercepat $a = 3\text{m/s}^2$ selama $t = 4\text{s}$
> $v = 0 + 3 \cdot 4 = 12\ m/s$

5. Hukum Newton II

Hukum ini menjelasakan hubungan antara gaya, massa, dan percepatan suatu benda, dimana bunyinya:
”Percepatan yang dialami oleh suatu benda berbanding lurus dengan gaya total yang bekerja padanya dan berbanding terbalik pada massanya”

Dengan kata lain:

- Jika gaya diperbesar, percepatan akan bertambah.

- Jika massa diperbesar, percepatan akan berkurang.

Rumus Hukum Newton II:

$$F = m \cdot a$$

Keterangan:
F : Gaya (N)
m : Massa (kg)
a : Percepatan ($m/2^2$)

Makna Rumus:
- Gaya menyebabkan benda berubah kecepatannya (artinya benda mengalami percepatan).
- Tanpa gaya ($F = 0$), benda tidak akan berubah kecepatannya; ini berkaitan dengan Hukum Newton I.
- Besarnya percepatan tergantung pada seberapa besar gaya dan besar massa benda tersebut.

Namun pada projek ini fokus yang digunakan adalah mencari percepatannya dimana:

$$a = \frac{F}{m}$$

Arti:
a : Percepatan
F : Gaya dorong
m : Massa bola

Mengapa saya menggunakan percepatan dari Hukum Newton II karena pada game fokus kita adalah pada bola dan bukan player, dimana bola itu memiliki massa 0.5kg. Sedangkan jika menggunakan percepatan dari GLBB yang dimana hanya menghitung posisi/kecepatan dari rumus waktu dan tanpa memperdulkan penyebabnya, sedangkan pada game penyebab di pengaruhi oleh player.

Karena player dan bola menggunakan besaran vector sebagai posisi dan arah maka setiap gaya palyer harus di sesuaikan atau di normalisasi agar jarak tidak mempengaruhi gaya. Maka rumus menjadi:

$$a = \frac{\vec{F}}{m}$$

Dimana $\vec{F}$ adalah gaya totalnya (No 2 Normalisasi).

6. Gesekan / drag

Pada game saya menambahkan gaya gesek udara dan papan mendatar dimana agar membuat bola berhenti dengan lebih halus.

Papan mendatar:

Gaya gesek (F_g) adalah gaya yang **melawan arah gerak benda** ketika benda bersentuhan dengan permukaan lain, seperti papan.

Rumus umumnya adalah:

$$F_g = \mu \cdot N$$

Keterangan:

- F_g : gaya gesek (Newton, N)
- μ : koefisien gesek antara papan dan benda (tanpa satuan)
- N = gaya normal (Newton, N)

Karena khasus kita adalah papan mendatar dimana

$$N = m \cdot g$$

Sehingga

$$F_g = \mu \cdot m \cdot g$$

Dan karena gaya gesek akan bekerja saat bola bergerak maka
μ menjadi μ_k dimana μ_k adalah koefisien gesek kinetik.

Dan karena pergerakan bola adalah vector maka kita perlu menjadikannya besaran vector:

$$\overrightarrow{F_g} = -\mu_k \cdot m \cdot g \cdot \hat{v}$$

dengan $\hat{v}$ adalah **unit vector arah kecepatan** bola.

Secara vector:

$$\vec{v} = (v_x, v_y)$$

Jika di hitung besaran kecepatannya adalah:

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

Maksud dari rumus di atas adalah agar tahu seberapa cepat bola bergerak (tanpa peduli arah), dan untuk membuat **vektor unit arah gerak** $\hat{v}$.

Dengan

$$\hat{v} = \frac{\vec{v}}{|\vec{v}|} = \left(\frac{v_x}{|\vec{v}|}, \frac{v_y}{|\vec{v}|}\right)$$

Karena $\vec{F} = m\vec{a} \Rightarrow \overrightarrow{a_g} = \frac{\overrightarrow{F_g}}{m} = \frac{\mu_k mg}{m}$ maka:

Massa di atas dan di bawah bisa saling di hilangkan menjadi:

$$\overrightarrow{a_g} = -\mu_k g\hat{v}$$

Ini berarti percepatan (atau perlambatan) akibat gesekan **arahnya berlawanan dengan arah gerak**, dan **besarannya tetap sebesar** $\mu_k g$

Pada kode bisa kita tulis:

$$\vec{v}_{t+\Delta t} = \vec{v}_t - (\mu_k g\hat{v})\Delta t$$

yang merupakan **integrasi numerik (metode Euler)** dari rumus percepatan:

$$\frac{d\vec{v}}{dt} = -\mu_k g\hat{v}$$

Gaya gesek Udara:

Pada benda yang bergerak dalam udara atau air, ada **gaya hambat (drag force)** yang arahnya **berlawanan dengan arah gerak**.

Model drag untuk kecepatan tinggi biasanya:

$$F_d = -kv^2$$

Tanda minus (−) artinya gaya drag melawan arah kecepatan.

- F_d : gaya hambat (N)
- k : konstanta hambatan (N·s/m)
- v : kecepatan (m/s)

Jadi Gaya gesek sebanding dengan kecepatan.

Dengan $k = \frac{1}{2} C_d \rho A v^2$

Keterangan:

- C_d : koefisien drag (tanpa satuan)
- ρ : massa jenis fluida (kg/m³)
- A : luas penampang (m²)
- v : kecepatan (m/s)

Hukum Newton II:

$$\sum F = m \cdot a$$

Karena percepatan a pada game adalah **turunan kecepatan terhadap waktu** $\boldsymbol{a = \frac{dv}{dt}}$ maka:

$$\boldsymbol{\sum F = m\frac{dv}{dt}}$$

Jadi dapat di tulis juga:

$$\boldsymbol{m\frac{dv}{dt} = -kv^2}$$

Bentuk sederhana

Dalam banyak simulasi (terutama game engine atau grafika komputer), gaya drag sering **disederhanakan** menjadi model linier:

$$\frac{dv}{dt} = -\mu v$$

Dengan μ = koefisien friksi/drag (frictionCoef).
μ juga dapat memakili massa (sudah dalam perhitungan).

Hasil persamaan diferensial di atas adalah:

$$v(t) = v_0 e^{-\mu t}$$

Dengan menurunkan rumus

$$\frac{dv}{dt} = -\mu v$$

kita tahu dengan pasti bagaimana kecepatan berubah seiring waktu secara eksponensial akibat gaya gesekan linier, sehingga bisa digunakan secara akurat dalam simulasi atau analisis fisika.

Tanpa penurunan, kita hanya punya persamaan diferensial tanpa gambaran bentuk solusi kecepatan terhadap waktu.

Bentuk kontinu ke bentuk diskrit (numerik)

Karena pada game waktu tidak lah kontinu karena setiap frame di hitung dalam langkah waktu kecil Δt. Jadi kita butuh bentuk diskrit dari solusi di atas:

$$v_{t+\Delta t} = v_t e^{-\mu \Delta t}$$

Dimana $v_{t+\Delta t} = v_t$ adalah vx pada bola, dan di dapat dari perhitungan GLBB.

Jadi kita definisikan

$$drag\_factor = e^{-\mu \Delta t}$$

Maka tiap frame:

$$v \leftarrow v \cdot drag\_factor$$

7. Pantulan pada dinding (koefisien restitusi)

Ketika bola menabrak permukaan dinding, kecepatan bola berubah arah akibat adanya gaya reaksi dari permukaan.

Namun, tidak semua energi kinetik kembali utuh, sebagian hilang karena gesekan atau pantulan.

Untuk mengukur seberapa "elastis" tabrakan itu, digunakan koefisien restitusi.

Koefisien restitusi, disimbolkan dengan e, adalah **ukuran seberapa elastis suatu tumbukan** yaitu **seberapa besar kecepatan benda setelah memantul dibanding sebelum menumbuk**.

$$e = \frac{v_{2f} - v_{1f}}{v_{2i} - v_{2i}}$$

Arti:

e : **Koefisien restitusi** — ukuran seberapa elastis suatu tumbukan
v_{1i}: Kecepatan benda 1 **sebelum** tumbukan
v_{2i}: Kecepatan benda 2 **sebelum** tumbukan
v_{1f}: Kecepatan benda 1 **setelah** tumbukan
v_{2f}: Kecepatan benda 1 **setelah** tumbukan

Karena dinding diam maka: ($v_{2i} = v_{2f} = 0$) jadi rumus dapat disederhanakan:

$$e = \frac{|v_{setelah}|}{|v_{sesudah}|}$$

Artinya:

- Kecepatan setelah pantulan sebanding dengan kecepatan sebelum tumbukan dikalikan e
- Arah kecepatan berbalik, karena benda memantul ke arah berlawanan

Setelah bola menabrak permukaan (misal dinding vertikal), maka:

- Komponen kecepatan **sejajar dinding** tetap (tidak berubah)
- Komponen kecepatan **tegak lurus dinding** berubah arah dan nilainya dikalikan dengan $-e$

Secara matematis:

$$v' = -ev$$

Dimana v adalah kecepatan sebelum tumbukan.

Dalam implementasi program, nilai **restitution** digunakan sebagai konstanta yang merepresentasikan seberapa elastis tumbukan antara objek dan permukaan. Nilai ini tidak dihitung secara otomatis dari kecepatan sebelum dan sesudah tumbukan, tetapi diberikan secara manual agar simulasi terlihat realistis. Rumus yang digunakan tetap mengikuti persamaan umum fisika $v' = -e\,v$, di mana $e$adalah koefisien restitusi dan v'merupakan kecepatan setelah tumbukan.

Contoh material dan rentan e:

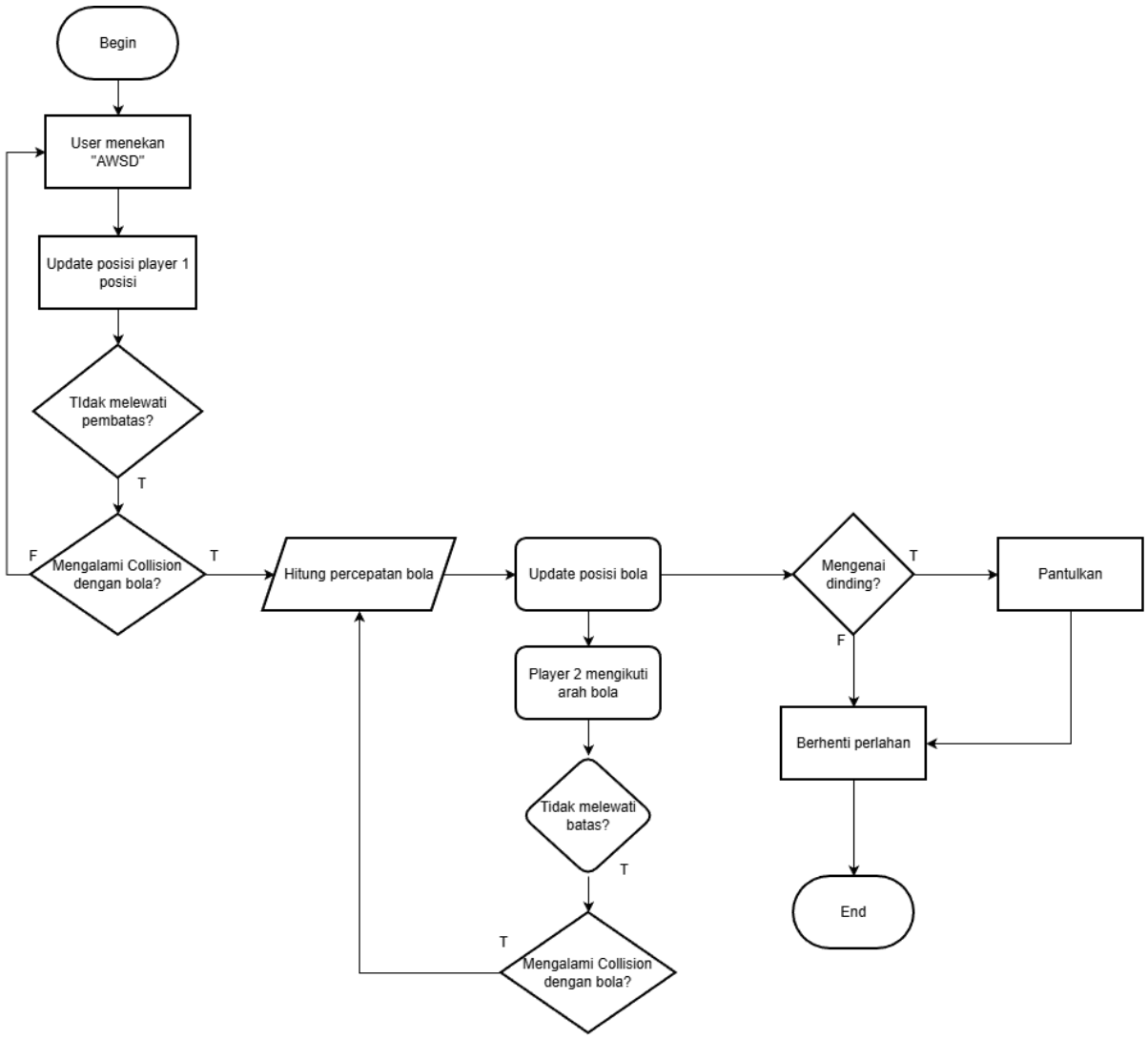
Material	**Rentang (e)**	**Karakteristik**
Bola baja di lantai baja	0.9 – 1.0	Pantulan sangat kuat (hampir elastis sempurna)
Bola tenis / karet	0.7 – 0.9	Pantulan sedang
Kayu	0.4 – 0.6	Pantulan rendah
Lumpur / kain	0.0 – 0.2	Hampir tidak memantul

Karena pada game mirip seperti bola tenis atau karet maka saya menggunakan nilai $e = 0.9$

Bab III
Implementasi

Pada bab ini akan fokus membahas dan menampilkan hasil dan uji coba berdasarkan rumus atau konsep yang telah disebutkan pada Bab II:

Alur Simulas



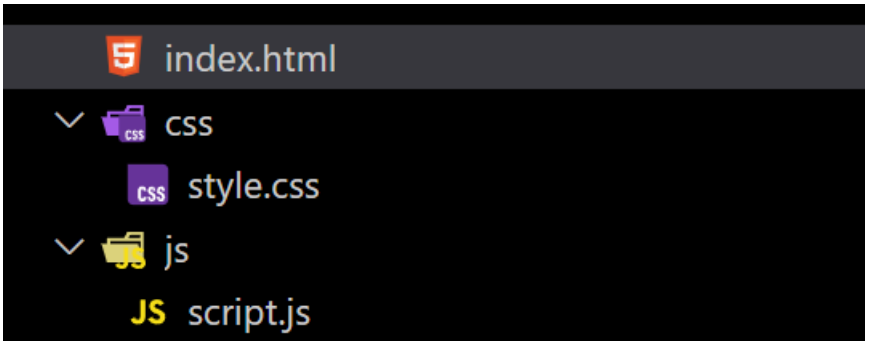
Setup projek

Pada projek ini saya menggunakan tech stack HyperText Markup Language (**HTML)** sebagai kerangka website, Cascading Style Sheet (**CSS)** sebagai styling website dan JavaScript (**JS)** sebagai penggerak game.

Implementasi:

Pertama-tama saya membuat sebuah folder **css** dan **js** sebagai tempat penyimpan file-file **css** dan **js**, lalu saya membuat file di folder root bernama *index.html,* file *style.css* di folder css, dan *script.js* di folder js.

Contoh hasil folder:



Isi file *index.html*:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>TA Fisika Semester 1</title>
  <link rel="stylesheet" href="./css/style.css">
</head>

<body>
  <canvas id="main"></canvas>

  <script src="./js/script.js"></script>
</body>

</html>
```

Dimana struktur folder di atas bisa langsung kita generate dengan menekan tanda seru *"!"*.

```
<link rel="stylesheet" href="./css/style.css">
```

Adalah lokasi yang menghubungkan ke file style, begitupun juga untuk yang *script.js*

```
<script src="./js/script.js"></script>
```

Pada tag canvas adalah tempat untuk membuat tampilan, player, dan animasi yang bergerak-gerak.

Pada file *style.css*

```css
body {
  padding: 0;
  margin: 0;
  overflow: hidden;
  display: flex;
  justify-content: center;
}
```

Dimana inti dari code ini hanya untuk menghilangkan margin dan padding pada tag body serta menengahkan isi dari kontent body, yakni adalah canvas

Pada file *script.js* inilah inti dari pembentukan gamenya, dimana isi dari file *script.js* adalah:

- Menggambar canvas (ring, player, bola).
- Menangani event (aksi AWSD).
- Perhitungan (Collision, Vector, Fisika, Pantulan).
- Pengkondisian atau pelogikaan game.

Penjelasan singkat pada awal-awal kode.

1. Pada awal file kita memilih tag HTML dengan id main sebagai canvas yang dimana di pakai untuk menggambar dalam bentuk 2D.
   ```js
   const canvas = document.getElementById("main");
   const ctx = canvas.getContext("2d");
   ```

2. Pada variabel-variabel **const**ant (tanpa kurung buka-tutup setelah =) adalah variabel yang di tentukan oleh penulis, baik nilai untuk fisika atau untuk game.

3. Pada event *keydown* dan keyup adalah event ketika user menekan dan melepas keyboard.
   ```js
   document.addEventListener("keydown", e => keyboard[e.code] = true);
   document.addEventListener("keyup", e => keyboard[e.code] = false);
   ```
   Dimana hasil dari *e.code* adalah kode keyboard pada JavaScript, dimana jika kode sama dengan variabel keyboard maka akan mengganti nilai menjadi true atau false.

4. Penjelasan singkat fungsi dibawah ini:
   - *drawBackground()* : mewarnai 1 canvas penuh dengan warna hitam.
   - *drawRing()* : menggambar garis-garis putih di dalam kotak.
   - *drawOffsetArea()* : membuar seperempat lingkaran di sudut-sudut ring.
   - *drawPlayersBall()* : menggambar Player 1(bawah), player 2(atas) dan bola.

5. Fungsi *reset()* berfungsi untuk mengembalikan semua nilai constant pada awal kode ke nilai awal.

Penerapan teori dan konsep dengan JavaScript

Pada poin ini akan membahas penerapan teori dan konsep yang mana ada pada Bab II dengan menggunakan bahasa pemrograman JavaScript.

> Note: Sumbu Kartesius pada website jika di gambarkan maka sumbu y yang jika kebawah maka menjadi positif dan bukan negatif, jadi sumbu y disini terbalik.

Logika gerak player 1

```
function playerController(dt) {
  const moveSpeed = 350; // px/s

  if (keyboard.KeyW && player1.y > (ringPosition.mv + player1.r))
    player1.y -= moveSpeed * dt;

  if (keyboard.KeyS && player1.y < (ringPosition.by - player1.r))
    player1.y += moveSpeed * dt;

  if (keyboard.KeyA && player1.x > (ringPosition.lx + player1.r))
    player1.x -= moveSpeed * dt;

  if (keyboard.KeyD && player1.x < (ringPosition.rx - player1.r))
    player1.x += moveSpeed * dt;
}
```

Variabel *moveSpeed* digunakan untuk kecepatan gerak pada player, jika di kalikan delta time maka jika 1 detik maka akan bergerak 350 px per detik.

Pada pengkondisian asal tidak melewati batas ring maka:

- Jika user menekan W maka akan keatas.
- Jika user menekan S maka akan kebawah.
- Jika user menekan A maka akan kekiri.
- Jika user menekan D maka akan kekanan.

Collision

Fungsi collision jika pada code dapat dituliskan seperti ini:

```
function collision(x1, y1, x2, y2) {
  const dx = x2 - x1;
  const dy = y2 - y1;
  return Math.sqrt((dx ** 2) + (dy ** 2));
}
```

Maksud fungsi *Math.sqrt()* adalah akar pangkat dan ** adalah pangkat di JavaScript.

Hitung percepatan

Setelah player dan bola mengalami collision maka disinilah teori dan konsep fisika mulai di pakai.

```
const d1 = collision(player1.x, player1.y, ball.x, ball.y);
if (d1 <= ball.r + player1.r) {
  const nx = (ball.x - player1.x) / d1;
  const ny = (ball.y - player1.y) / d1;

  const overlap = ball.r + player1.r - d1;
  ball.x += nx * overlap;
  ball.y += ny * overlap;

  // Fv = F * n
  const Fvx = Fpush * nx;
  const Fvy = Fpush * ny;

  // a = Fv / m
  ball.ax += Fvx / ball.m;
  ball.ay += Fvy / ball.m;
}
```

Pada saat collision terdapat 4 konsep teori dan konsep yang di terapkan:

1. **Collision**, dimana jika jarak (hasil rumus collision) <= kedua radius player dan bola maka keduanya pasti bertumbukan, saat inilah penerapan konsep dan teori diterapkan.
2. **Normalisasi** diperlukan karena jika tidak maka jarak *d1* maka akan mempengaruhi gaya yang di bawa oleh player, sehingga mengakibatkan kecepatan bola tidak sesuai.
3. **Overlap** penulis tambahkan agar player dan bola tidak saling tumpang tindih saat bertumbukan.
4. **Percepatan** dari hukum newton II dimana yang awalnya:

   $$F = m \cdot a$$

   Karena gaya sudah di tentukan oleh penulis namun percepatan belum, maka percepatan lah yang di cari:

   $$a = \frac{F}{m}$$

Update percepatan

Setelah nilai percepatan di hitung, sekarang gunakanlah rumus GLBB untuk mencari percepatan: $v = v_0 + at$

```
ball.vx += ball.ax * dt;
ball.vy += ball.ay * dt;

ball.x += ball.vx * dt;
ball.y += ball.vy * dt;
```

v += artinya sama aja kecepatan sekarang di tambah kecepatan sebelumnya.

Karena pada komputer, waktu biasanya dihitung per frame(FPS), kita tidak bisa langsung menghitung *t* secara kontinu, jadi kita pakai selang waktu kecil yang di sebut *dt* (ditulis di fungsi main).

Maka daripada menghitung langsung rumus kuadrat yang dimana bisa membuat angka semakin besar,
kita lakukan pendekatan per langkah kecil (diskrit):

Kecepatan diskrit:

$$v_{t+\Delta t} = v_t + a\Delta t$$

Kecepatan pada waktu **t + Δt** sama dengan kecepatan pada waktu **t** ditambah percepatan **a** dikalikan selang waktu **Δt**.

Jika di pemrograman nilai di atas pastinya di simpan ke dalam variabel, sehingga kita bisa menulis rumus diskrit perpindahan yang mulanya:

$$s_{t+\Delta t} = s_t + v_{t+\Delta t}\Delta t$$

karena program tidak langsung menghitung "berapa jauh total akibat percepatan selama t detik", melainkan menghitung "berapa jauh akibat percepatan dalam selang kecil dt, lalu diulang terus", maka pembagian $\frac{1}{2}$ **tidak perlu.**

Gesekan papan & udara

1. Gesekan Udara
   Dimana rumusnya:

   $$v_{t+\Delta t} = v_t e^{-\mu\Delta t}$$

   $$drag\_factor = e^{-\mu\Delta t}$$

   Maka tiap frame:

   $$v \leftarrow v \cdot drag\_factor$$

   Jika di terapkan dalam kode maka:

```
const drag = Math.pow(1 - frictionCoef * dt, dt * 60);
ball.vx *= drag;
ball.vy *= drag;
```

2. Gesekan papan

```
const ballV = Math.sqrt(ball.vx ** 2 + ball.vy ** 2);
if (ballV > 0){
  // arah kecepatan
  const vxNorm = ball.vx / ballV;
  const vyNorm = ball.vy / ballV;

  // a
  const af = mu * g;

  ball.vx -= vxNorm * af * dt;
  ball.vy -= vyNorm * af * dt;

  if (Math.abs(ball.vx) < 0.01) ball.vx = 0;
  if (Math.abs(ball.vy) < 0.01) ball.vy = 0;
}
```

Dimana kita harus mengetahui dulu berapa besaran kecepatan pada bola:

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

Lalu buat **vektor unit arah gerak** $\hat{v}$ berdasarkan besaran kecepatannya:

$$\hat{v} = \frac{\vec{v}}{|\vec{v}|} = \left(\frac{v_x}{|\vec{v}|}, \frac{v_y}{|\vec{v}|}\right)$$

Kita perlu mengetahui percepatan atau perlambatan akibat gaya geseknya dengan mencari percepatan dari hukum newton II $\vec{F} = m\vec{a} \Rightarrow \overrightarrow{a_g} = \frac{\overrightarrow{F_g}}{m} = \frac{\mu_k mg}{m}$ maka:

$$\overrightarrow{a_g} = -\mu_k g\hat{v}$$

Sehingga bisa di masukan kedalam rumus gesekan papan mendatar secara diskrit:

$$\vec{v}_{t+\Delta t} = \vec{v}_t - (\mu_k g\hat{v})\Delta t$$

Logika bot mengikut bola

```
function botMoveLogic(dt) {
  const speed = 350; // px/s

  if (ball.y < ringPosition.mv - 20) {
    if (ball.x < player2.x) player2.x -= speed * dt;
    if (ball.x > player2.x) player2.x += speed * dt;
    if (ball.y < player2.y) player2.y -= speed * dt;
    if (ball.y > player2.y) player2.y += speed * dt;
  } else if (ball.y > ringPosition.mv + 20) {
    if (ball.x < player2.x && player2.x > (ringPosition.lx + player2.r))
      player2.x -= (speed * dt) / 2;

    if (ball.x > player2.x && player2.x < (ringPosition.rx - player2.r))
      player2.x += (speed * dt) / 2;

    if (ball.y < (ringPosition.mv + (ringPosition.mv / 2.5)) - 20 && (player2.y < (ringPosition.mv - (player2.r + 5))))
      player2.y += (speed * dt);

    if (ball.y > player2.y && player2.y > (ringPosition.ty + (player2.r + 20)))
      player2.y -= (speed * dt) / 5;

  }
}
```

Dimana inti dari *if* pertama adalah ketika bola melewati ring tengah (ke atas) maka bot atau player 1

1. Jika bola di kiri bot maka bot akan kekiri.
2. Jika bola di kanan bot maka bot akan kekanan.
3. Jika bola di belakang bot maka bot akan mundur.
4. Jika bola di depan bot maka bot akan maju.

Pada *if* kedua menjelaskan jika bola berada di bawah ring tengah(rt) maka:

1. Jika bola berada kiri bot dan tidak melewati ring kiri maka bot kekiri.
2. Jika bola berada kanan bot dan tidak melewati ring kanan maka bot kekanan.
3. Jika bola berada di $\frac{5}{2}$ kebawah dari rt dan tidak melewati rt maka bot maju.
4. Jika bola berada di depan bot dan tidak melewati ring atas maka bot mundur perlahan.

Pantulan pada dinding

```
// wall bounce
if (ball.x - ball.r < ringPosition.lx) {
  ball.x = ringPosition.lx + ball.r; // bola agar langsung menempel dan pantulan jadi lebih akurat / elastis
  ball.vx *= -restitution;
}
if (ball.x + ball.r > ringPosition.rx) {
  ball.x = ringPosition.rx - ball.r;
  ball.vx *= -restitution;
}
if (ball.y - ball.r < ringPosition.ty) {
  ball.y = ringPosition.ty + ball.r;
  ball.vy *= -restitution;
}
if (ball.y + ball.r > ringPosition.by) {
  ball.y = ringPosition.by - ball.r;
  ball.vy *= -restitution;
}
```

Pada pantulan, karena kecepatan dinding adalah 0, maka kita bisa langsung rumus koefesiensi restitusi:

$$v' = -ev$$

Dimana v adalah kecepatan sebelum tumbukan.
Pada kode bisa di sesuaikan dengan setiap dinding atas, bawah, kiri, kanan.

Area Offset

```
// Offset Area Collision
if (collision(ball.x, ball.y, 20, 20) <= (ball.r + (offsetAreaSize))) reset();
if (collision(ball.x, ball.y, canvas.width - 20, 20) <= (ball.r + (offsetAreaSize))) reset();
if (collision(ball.x, ball.y, canvas.width - 20, canvas.height - 29) <= (ball.r + (offsetAreaSize))) reset();
if (collision(ball.x, ball.y, 20, canvas.height - 20) <= (ball.r + (offsetAreaSize))) reset();
```

Pada area offset karena bentuk sebenarnya area offset seperti lingkaran yang di potong ¼ maka bisa menggunakan fungsi collision.

Pada parameter 3 dan 4 di pengkondisian adalah titik pembatas dimana di tambahkan beberapa angka agar pas. Variabel *offsetAreaSize(OAS)* adalah ukuran bola dikali 3.

Pada pengkondisian:

- Jika bola berada di titik kiri (20) dan atas (20) dan posisinya = di tambah radius dan OAS maka reset.
- Jika bola berada di titik ujung kanan – 20 dan di atas dan posisinya = di tambah radius dan OAS maka reset.
- Jika bola berada di ujung kanan -29 dan ujung bawah – 20 dan posisinya = di tambah radius dan OAS maka reset.
- Jika bola berada di titik kiri (20) dan di ujung bawah – 20 dan posisinya = di tambah radius dan OAS maka reset.

Bagaimana game di mulai dalam program

Pada program saya hanya mendefisinikan fungsi-fungsi dimana jika tidak di panggil maka fungsi itu hanya sekedar fungsi yang tidak berguna. Maka penulis membuat sebuah 1 fungsi yang mencangkup semua fungsi agar berjalan dan berurutan.

```
function update(dt) {
  playerController(dt);
  botMoveLogic(dt);
  handleCollisions(dt);
  updateBall(dt);
}

function draw() {
  drawBackground();
  drawRing();
  drawOffsetArea();
  drawPlayersBall();
}

// loop utama
let lastTime = performance.now();

function main(now) {
  const dt = (now - lastTime) / 1000;
  lastTime = now;

  update(dt);
  draw();

  // renderFormulas();
  requestAnimationFrame(main); // default 60FPS
}

requestAnimationFrame(main);
```

Pada fungsi *update()* berguna untuk menangani perhitungan matematis dan fisika.

sedangkanfungsi *draw()* berguna untuk menggambar gambar dasar seperti background, ring, offset, player dan bola.