📘 Documentação: Simulação de Detecção & Perseguição 2D

("Ligeirinho" vs "Frajola")

1. Visão Geral

Esta aplicação é um laboratório interativo para testar algoritmos de perseguição autônoma e visão computacional básica. Diferente de um jogo controlado pelo usuário, esta é uma **simulação automatizada** onde você ajusta os parâmetros comportamentais de um agente perseguidor ("Frajola") que tenta capturar um alvo ("Ligeirinho") usando apenas informações visuais simuladas.

O sistema demonstra como atrasos de processamento (latência) e ruído visual afetam a eficácia de diferentes estratégias de navegação autônoma.

2. Funcionalidades Principais

🧠 Inteligência Artificial (Agentes)

- **Alvo (Ligeirinho):** Move-se em linhas retas com velocidade constante, ricocheteando nas bordas do cenário ou reaparecendo (respawn) se sair da área.
- **Perseguidor (Frajola):** Utiliza algoritmos geométricos e preditivos para interceptar o alvo baseando-se apenas na posição percebida pelo sistema de visão.

👁️ Visão Computacional Simulada

O "Frajola" não sabe a coordenada `x, y` exata do alvo magicamente. Ele precisa "ver":

- **Subtração de Fundo (Frame Differencing):** O sistema compara o frame atual com o anterior para detectar mudanças de pixels.
- **Limiarização (Threshold):** Ignora mudanças sutis de cor para filtrar ruído.
- **Cálculo de Centróide:** Determina o centro de massa da mancha de pixels detectada para estimar a posição do alvo.

📊 Coleta de Dados

- **Modo Batch:** Capacidade de rodar 50 simulações aceleradas em sequência.
- **Exportação:** Gera arquivos `.csv` com métricas detalhadas (tempo de captura, estratégia usada, velocidades) para análise externa.

---

3. Guia de Uso (Painel de Controle)

O painel lateral esquerdo permite ajustar a "física" e o "cérebro" da simulação em tempo real.

Controles de Simulação

Controle	Função
Iniciar / Pausar	Alterna o estado da execução.
Passo	Avança exatamente 1 frame (útil para debug).
Reset	Reinicia as posições dos agentes e limpa a memória de visão.
Rodar 50 cenários	Executa uma bateria de testes acelerada para gerar estatísticas.

Parâmetros Físicos

- **Velocidade Ligeirinho/Frajola:** Define quantos pixels por frame cada agente move.
- **Taxa de atualização (FPS):** Controla a velocidade do loop da simulação (10 a 144Hz).

Parâmetros de Visão e IA

- **Sensibilidade (Threshold):** Define quão diferente um pixel precisa ser do frame anterior para ser considerado "movimento". Valores altos deixam o Frajola "cego" a movimentos sutis.
- **Atraso de Reação (Delay):** Simula latência de processamento. Um valor de `10` significa que o Frajola persegue onde o alvo estava a 10 frames atrás.
- **Estratégia de Perseguição:** (Detalhado na seção 4).

---

4. Arquitetura Técnica

4.1 Pipeline de Detecção (Visão)

O script utiliza um `OffscreenCanvas` para desenhar apenas o alvo, isolado do fundo.

1. **Captura:** Obtém `ImageData` do canvas oculto.
2. **Diferenciação:** Itera sobre o array de pixels (`Uint8ClampedArray`). $Diff = |R - R_{ant}| + |G - G_{ant}| + |B - B_{ant}|$
3. **Filtragem:** Se $Diff > Threshold$, o pixel é considerado ativo.

4. **Localização:** Calcula a média das posições X e Y dos pixels ativos para encontrar o centro do objeto.

4.2 Estratégias de Perseguição (Navegação)

O núcleo da inteligência do "Frajola" reside na função `pursuitStep`. Existem três modos implementados:

A. Pursuit Direto (`pure`)

O perseguidor move-se diretamente para a última posição conhecida do alvo.

- **Vantagem:** Simples e robusto.
- **Desvantagem:** Ineficiente; o perseguidor acaba seguindo o "rastro" do alvo (tail-chase) em vez de cortá-lo.

B. Previsão Linear (`lead`)

Estima a velocidade do alvo baseada nas duas últimas detecções e projeta um ponto futuro.

- **Lógica:** P f u t u r o = P a t u a l + V e s t i m a d a × F a t o r L e a d
- **Vantagem:** Corta caminho em trajetórias retas.
- **Desvantagem:** Tende a errar (overshoot) se o alvo mudar de direção bruscamente.

C. Interceptação Analítica (`intercept`)

Utiliza geometria analítica para calcular o **Ponto de Colisão Ideal**. Resolve uma equação quadrática para encontrar o menor tempo t onde as trajetórias se cruzam, considerando as velocidades escalares máximas de ambos.

A equação baseia-se na lei dos cossenos relativa ao triângulo de velocidades: t = − b ± b 2 − 4 a c 2 a Onde a , b , c são derivados das posições e vetores de velocidade relativos. Se uma solução real positiva existe, o Frajola se move para o ponto de interceptação exato.

---

5. Estrutura do Código

Classes e Objetos

- `Agent`: Classe genérica para renderizar e armazenar estado (x, y, cor, tamanho).
- `cfg`: Objeto global de configuração que armazena o estado dos sliders do DOM.
- `metrics`: Armazena o histórico de execuções para a tabela e exportação CSV.

Loop Principal

O loop utiliza a técnica de **Delta Time** acumulado para garantir que a lógica da simulação rode na frequência correta (`cfg.fps`), independente da taxa de atualização da tela (renderização).

```
// Exemplo simplificado da lógica de tempo
while(accumulator >= stepMs){
  simStep(); // Atualiza física e IA
  accumulator -= stepMs;
}
render(); // Desenha na tela
```

6. Interpretação dos Resultados

Ao rodar a simulação, a tabela lateral mostra:

- **Sucesso:** Se o Frajola tocou no Ligeirinho antes de um timeout (2000 frames) ou antes do alvo sair da tela.
- **Frames:** Quão rápido foi a captura. Menos frames = Melhor desempenho.

Experimento sugerido: Aumente o *Delay* para 10 frames.

- A estratégia **Pure** começará a falhar miseravelmente (correndo atrás do passado).
- A estratégia **Lead** tentará compensar, mas pode oscilar.
- A estratégia **Intercept** geralmente oferece o melhor desempenho, mas exige uma estimativa de velocidade do alvo muito precisa.