RobôCIn

Desafio Seletiva

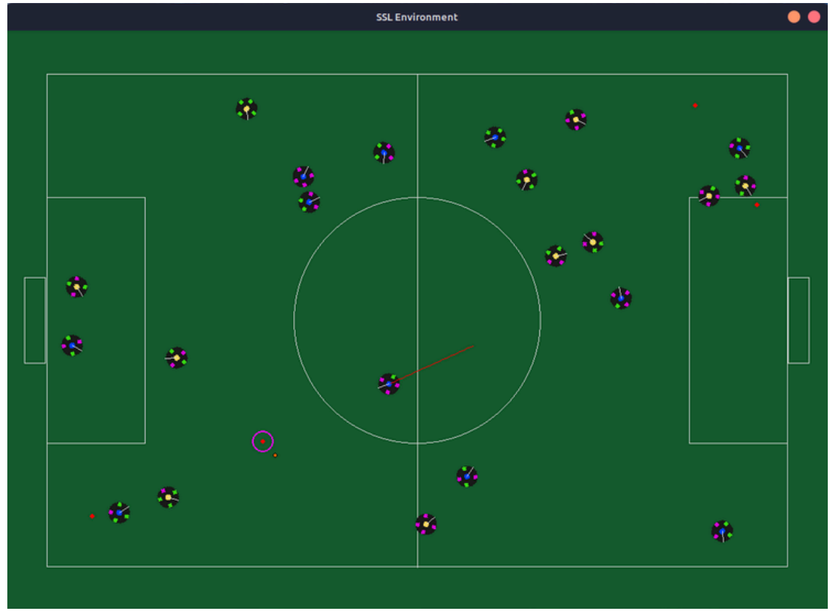
Candidato: Davi Sorrentino Brilhante

O Desafio - Visão Geral

- **Planejamento de trajetórias**
  - Desviar de obstáculos
  - Conquistar alvos

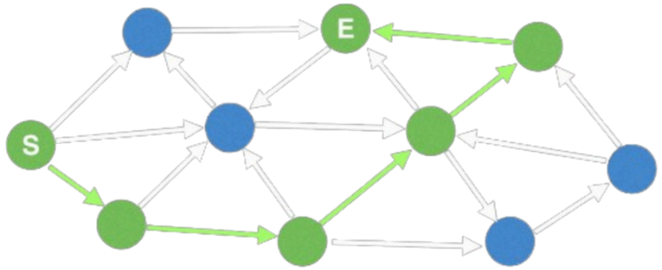
- **Atribuição de tarefas**
  - Designar alvos para cada agente disponível

- **4 Dificuldades**
  - 1. Obstáculos estáticos, 1 alvo por rodada
  - 2. Obstáculos dinâmicos, 1 alvo por rodada
  - 3. Obstáculos dinâmicos, 1 a 4 alvos por rodada
  - 4. Obstáculos dinâmicos, 1 a 6 alvos por rodada

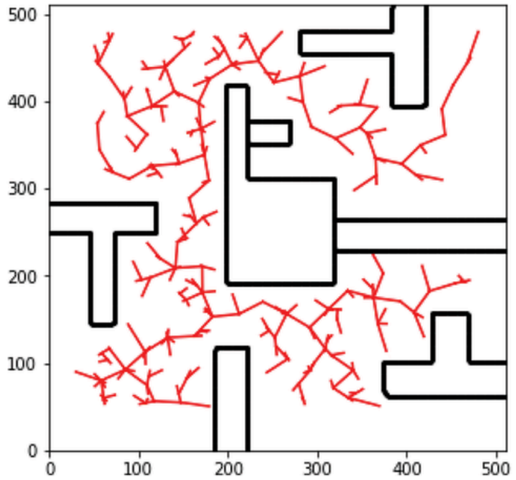


Abordagens Estudadas - Obstáculos

- **Dijkstra e A***
  - Encontram o caminho mais curto em grafos
  - Determinísticos e garantem o caminho mais curto
  - Tradução de coordenadas para grafos
  - Remodelagem
  - Precisão no campo

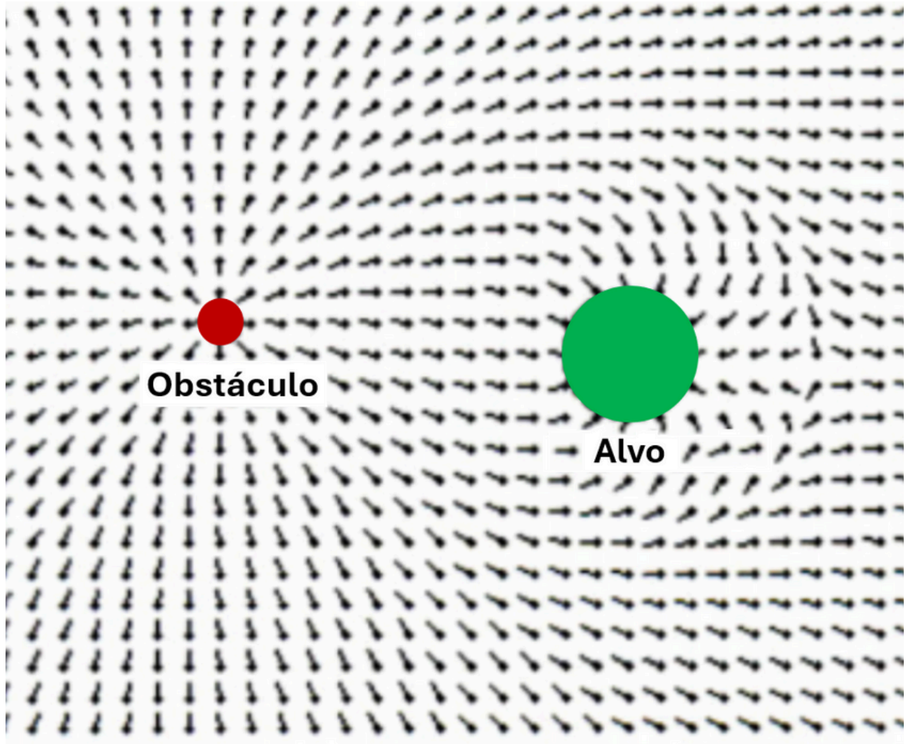


- **RRT (Rapidly-Exploring Random Tree)**
  - Considera os obstáculos para encontrar uma trajetória
  - Lida bem com obstáculos complexos
  - Limitações em obstáculos densamente ocupados por obstáculos



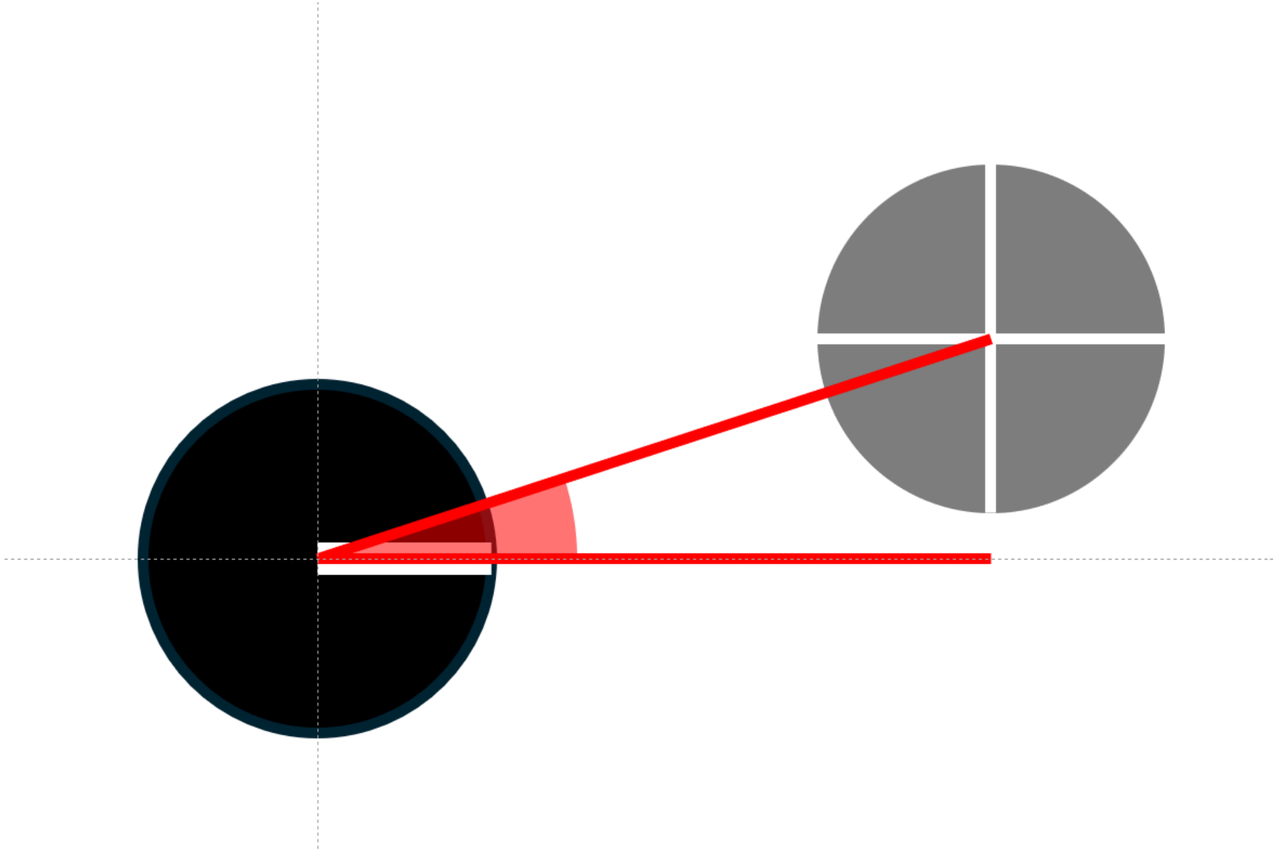
Abordagens Estudadas - Obstáculos

- **Campos Vetoriais de repulsão**
  - Obstáculos geram forças de repulsão
  - Alvo gera uma força de atração
  - Agentes são afetados pela soma vetorial dessas forças
  - Desviam dos obstáculos e alcançam o alvo
  - Altamente reativa
  - Computacionalmente eficiente e pouco custosa
  - Escudos de repulsão



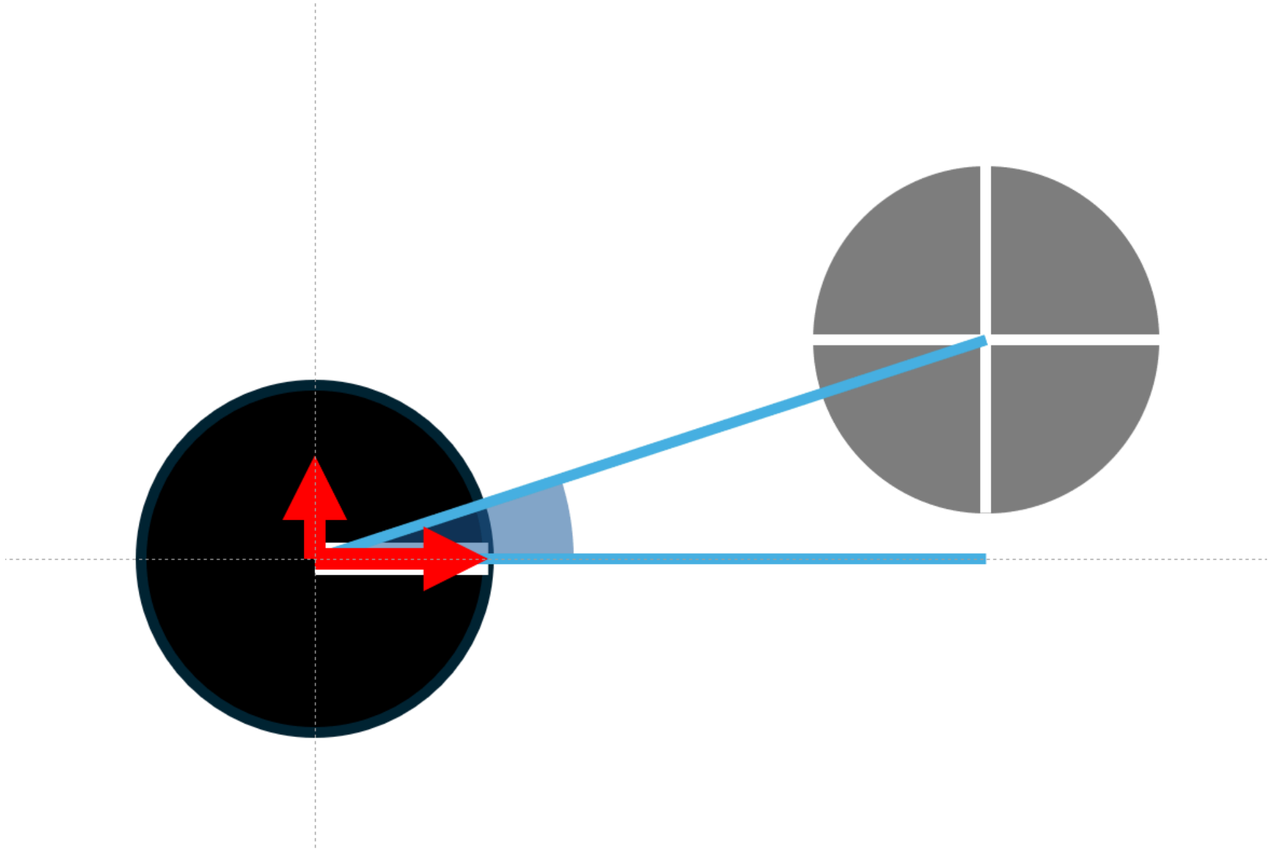
Abordagem escolhida

- **Campos Vetoriais de repulsão**
    1. Ângulo entre robô e obstáculo



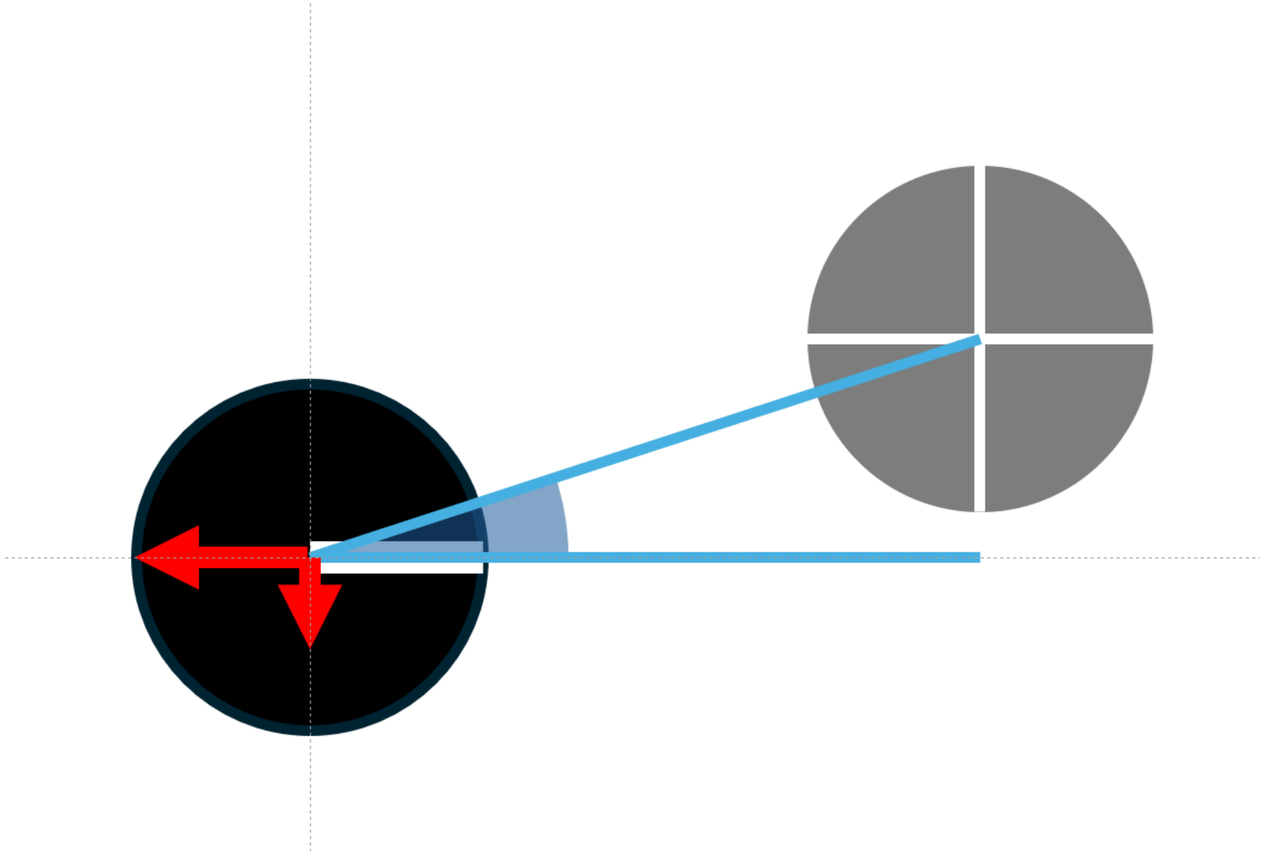
Abordagem escolhida

- **Campos Vetoriais de repulsão**
    1. Ângulo entre robô e obstáculo
    2. Projeção nos eixos



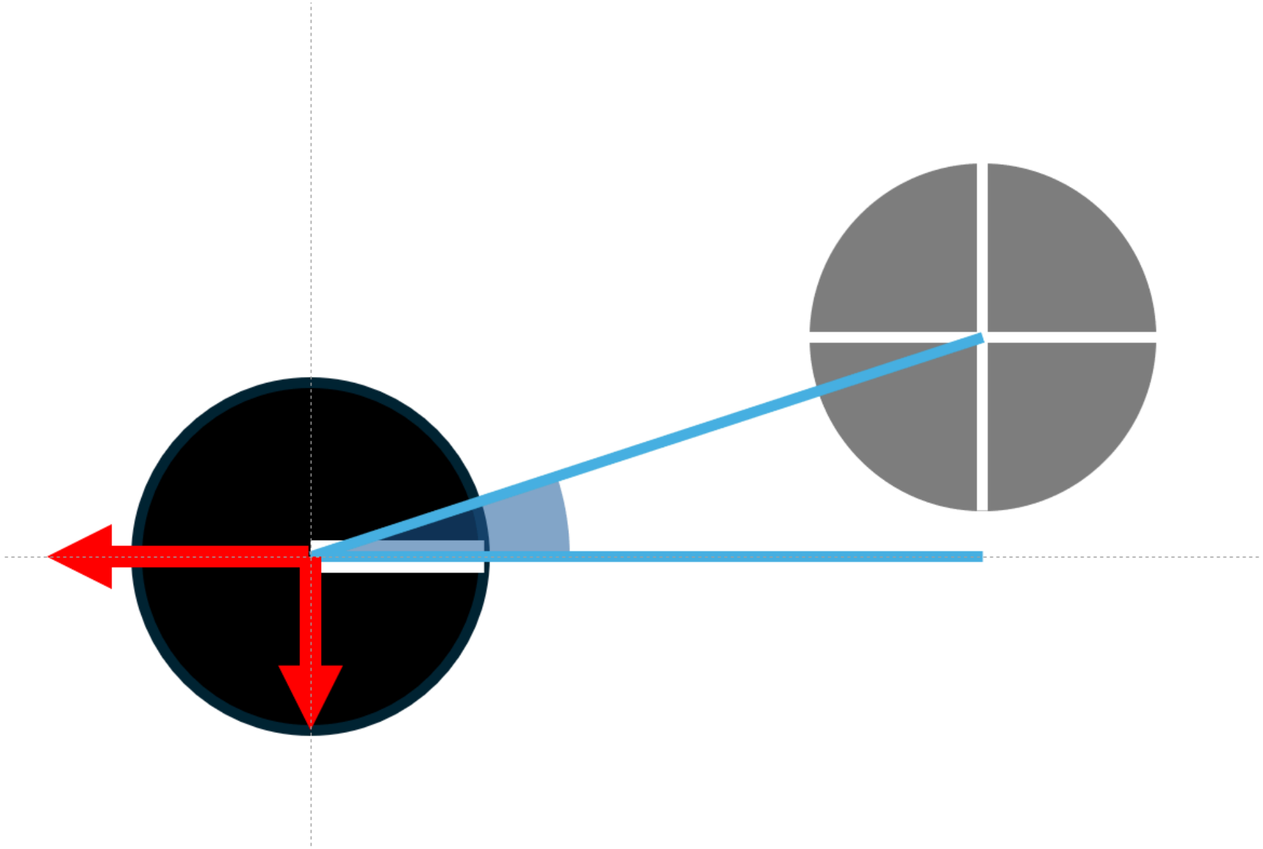
Abordagem escolhida

- **Campos Vetoriais de repulsão**
    1. Ângulo entre robô e obstáculo
    2. Projeção nos eixos
    3. Inversão de sentido



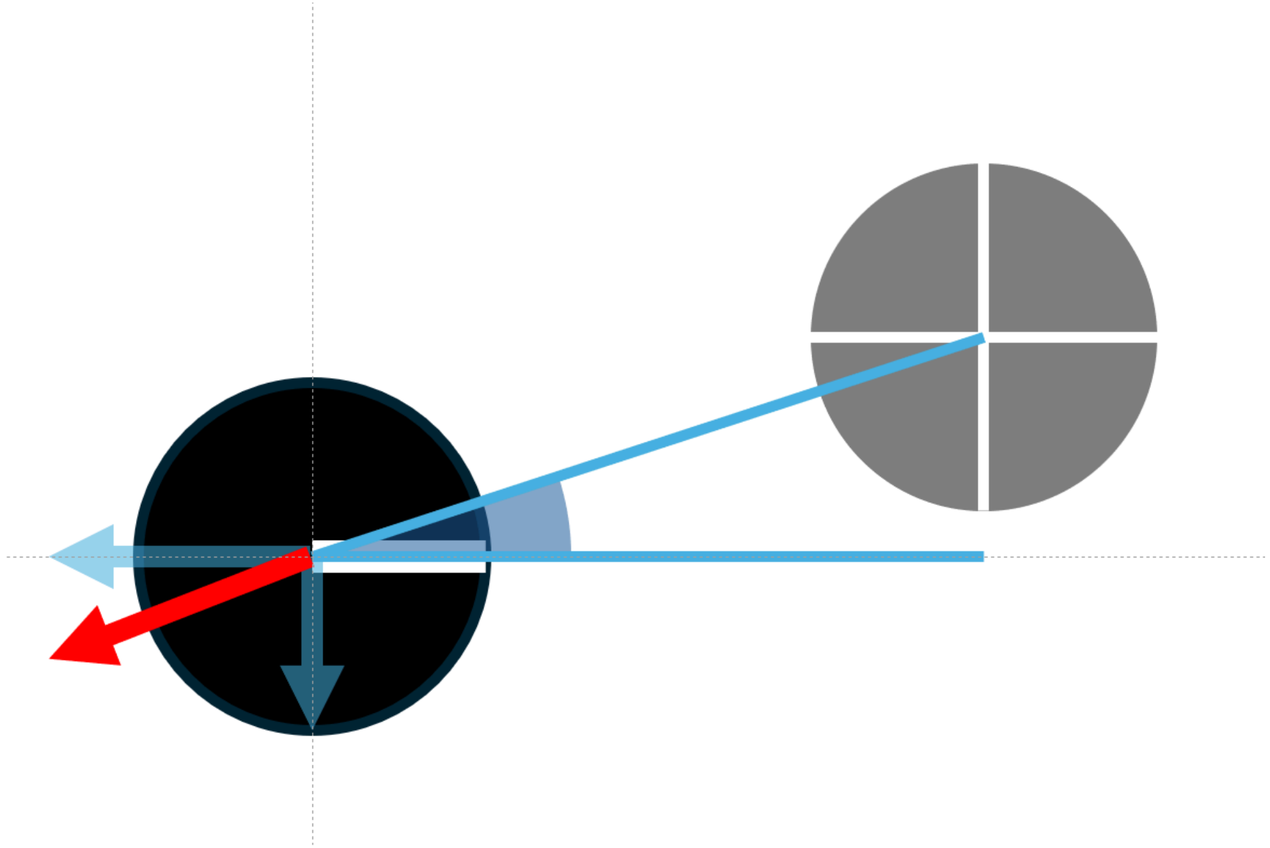
Abordagem escolhida

- **Campos Vetoriais de repulsão**
    1. Ângulo entre robô e obstáculo
    2. Projeção nos eixos
    3. Inversão de sentido
    4. Ajuste de magnitude



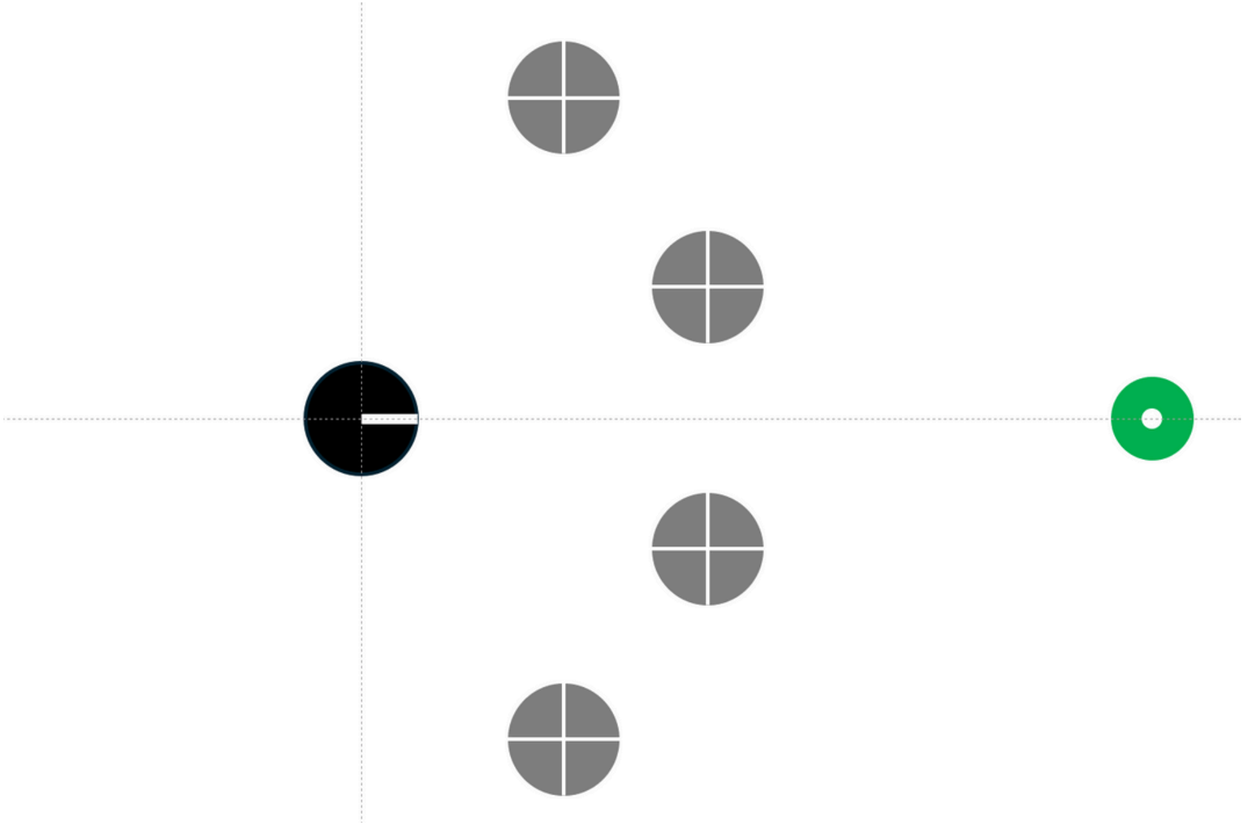
Abordagem escolhida

- **Campos Vetoriais de repulsão**
    1. Ângulo entre robô e obstáculo
    2. Projeção nos eixos
    3. Inversão de sentido
    4. Ajuste de magnitude
    5. Vetor repulsão (resultante)



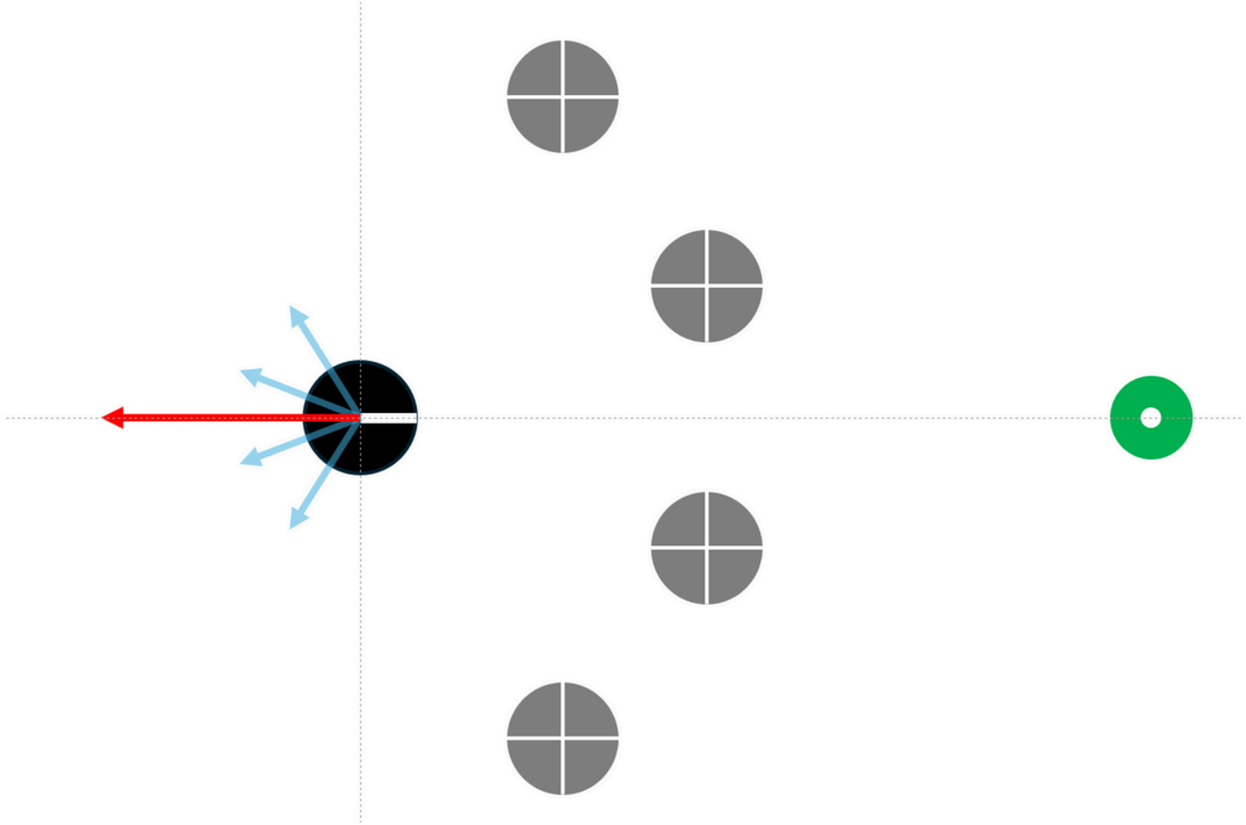
Abordagem escolhida

- **Campos Vetoriais de repulsão**
    1. Ângulo entre robô e obstáculo
    2. Projeção nos eixos
    3. Inversão de sentido
    4. Ajuste de magnitude
    5. Vetor repulsão (resultante)



Abordagem escolhida

- **Campos Vetoriais de repulsão**
    1. Ângulo entre robô e obstáculo
    2. Projeção nos eixos
    3. Inversão de sentido
    4. Ajuste de magnitude
    5. Vetor repulsão (resultante)



Abordagem escolhida

- **Campos Vetoriais de repulsão**
    1. Ângulo entre robô e obstáculo
    2. Projeção nos eixos
    3. Inversão de sentido
    4. Ajuste de magnitude
    5. Vetor repulsão (resultante)

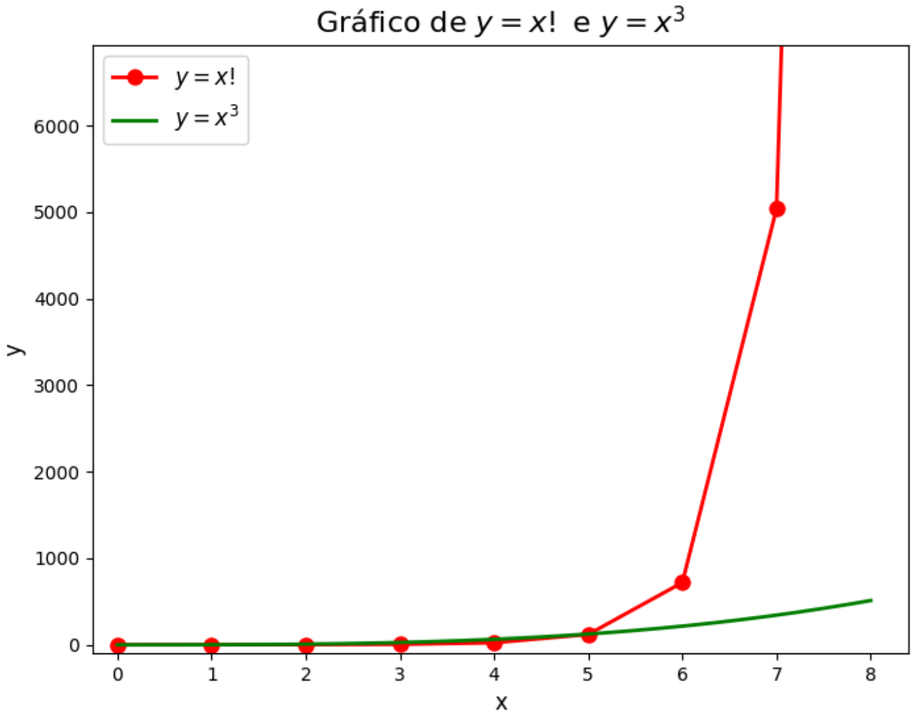
```python
def avoid_obstacles(self, current_position, target_position):
    safe_distance = 0.35
    adjustment_factor = 1.7

    adjusted_position = target_position
    for robot_id, opponent in self.opponents.items():
        obstacle_position = Point(opponent.x, opponent.y)
        distance_to_obstacle = current_position.dist_to(obstacle_position)

        if not distance_to_obstacle < safe_distance:
            continue

        angle_to_obstacle = math.atan2(
            obstacle_position.y - current_position.y,
            obstacle_position.x - current_position.x,
        )

        repulsion_x = -math.cos(angle_to_obstacle) * adjustment_factor
        repulsion_y = -math.sin(angle_to_obstacle) * adjustment_factor

        adjusted_position = Point(
            adjusted_position.x + repulsion_x,
            adjusted_position.y + repulsion_y,
        )

    return adjusted_position
```

obs: versão sem comentários. não é a versão final

Abordagens Estudadas - Atribuição

- **Permutação (n!)**
  - Brute force
  - Analisa todas as cobinações
  - Uma atribuição ótima
  - Extremamente custosa

- **Algoritmo Hungaro**
  - Resolve problemas de atribuição de tarefas
  - Uma atribuição ótima
  - Tempo polinomial



Abordagem escolhida - Hungaro

1. **Normalização da matriz**
2. **Verificar se temos uma solução ótima**
3. **Cobrimento da matriz**
4. **Atualizar custos**

Alvo / Robô	A1	A2	A3	A4
R1	10	12	15	16
R2	14	12	13	18
R3	10	16	19	15
R4	14	12	13	15

Abordagem escolhida - Hungaro

1. Normalização da matriz

- Subtrair os valores de cada linha pelo menor valor dela
- Subtrair os valores de cada coluna pelo menor valor dela

2. Verificar se temos uma solução ótima

3. Cobrimento da matriz

4. Atualizar custos

10	12	15	16
14	12	13	18
10	16	19	15
14	12	13	15

Abordagem escolhida - Hungaro

1. Normalização da matriz

- Subtrair os valores de cada linha pelo menor valor dela
- Subtrair os valores de cada coluna pelo menor valor dela

2. Verificar se temos uma solução ótima

3. Cobrimento da matriz

4. Atualizar custos

0	2	4	3
2	0	0	3
0	6	8	2
2	0	0	0

Abordagem escolhida - Hungaro

1. Normalização da matriz

2. Verificar se temos uma solução ótima

- Há zeros de forma em que seja possível escolher N deles em linhas e colunas diferentes?
- Se sim, essa é uma solução ótima e o algoritmo encerra

3. Cobrimento da matriz

4. Atualizar custos

0	2	4	3
2	0	0	3
0	6	8	2
2	0	0	0

Abordagem escolhida - Hungaro

1. **Normalização da matriz**
2. **Verificar se temos uma solução ótima**
   - Há zeros de forma em que seja possível escolher N deles em linhas e colunas diferentes?
   - Se sim, essa é uma solução ótima e o algoritmo encerra
3. **Cobrimento da matriz**
4. **Atualizar custos**

0	2	4	3
2	**0**	0	3
0	6	8	2
2	0	**0**	0

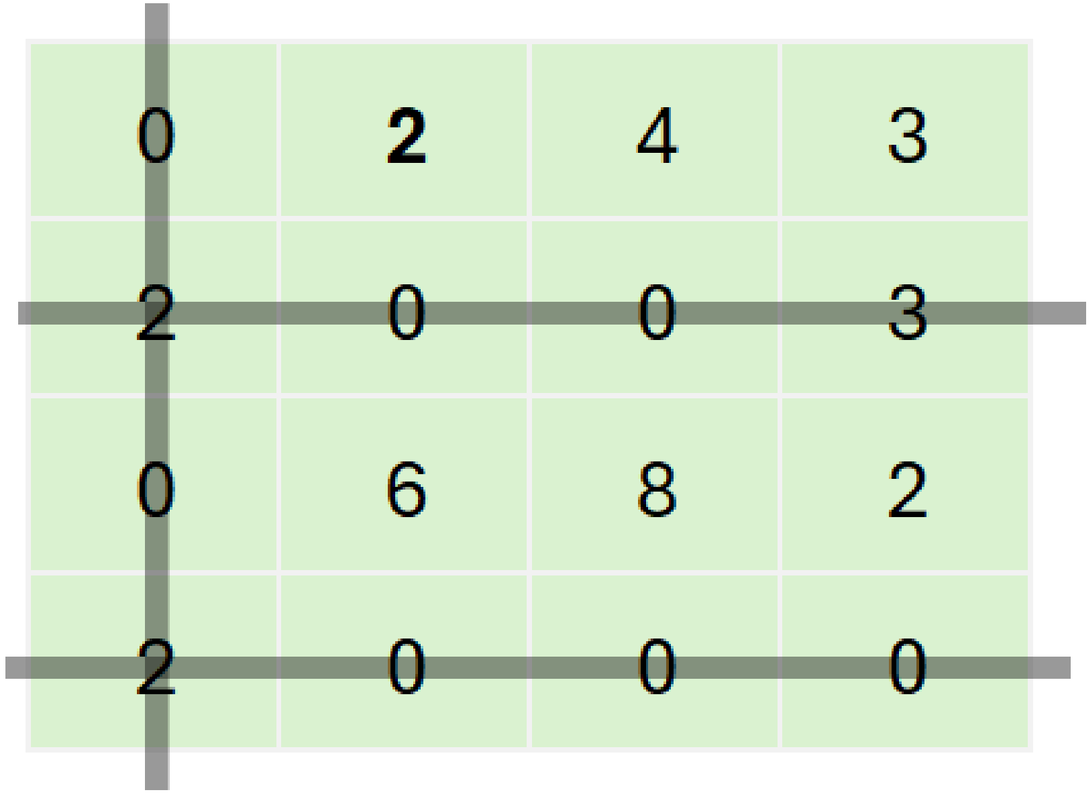
Abordagem escolhida - Hungaro

1. **Normalização da matriz**
2. **Verificar se temos uma solução ótima**
3. **Cobrimento da matriz**
   - Marcar traços em linhas e colunas até todos os zeros estarem riscados
   - Marcar o menor número possível de traços
4. **Atualizar custos**

0	2	4	3
2	0	0	3
0	6	8	2
2	0	0	0

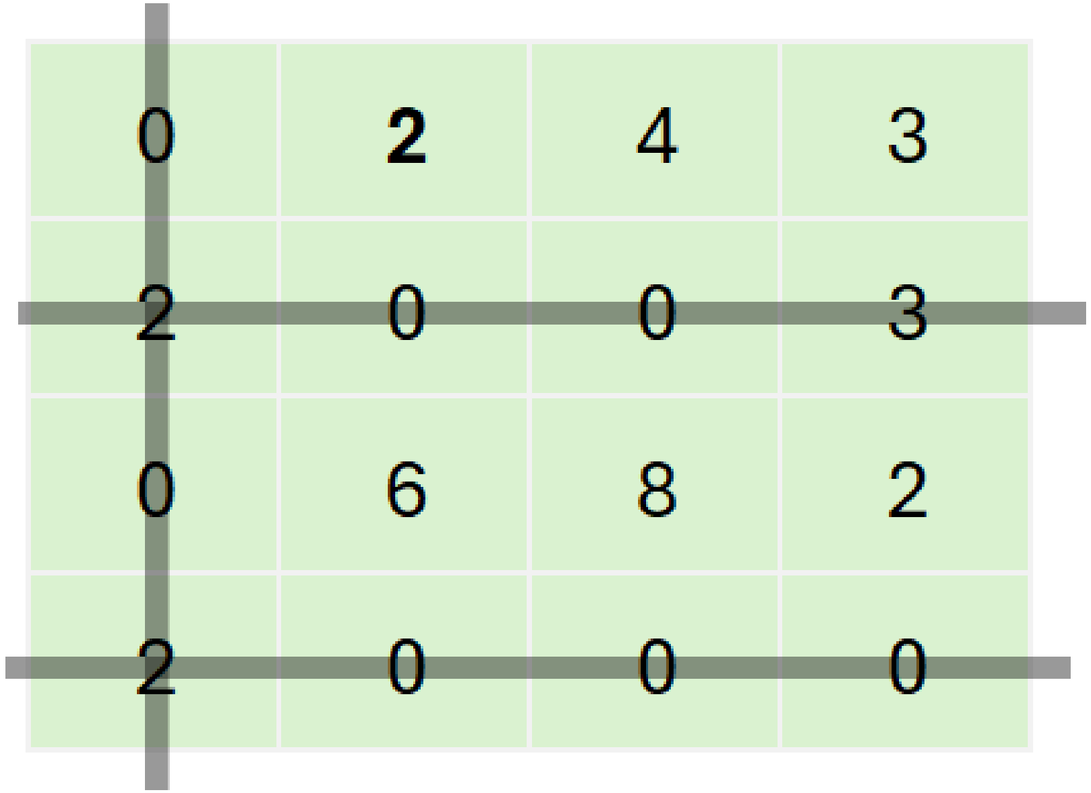
Abordagem escolhida - Hungaro

1. **Normalização da matriz**
2. **Verificar se temos uma solução ótima**
3. **Cobrimento da matriz**
   - Marcar traços em linhas e colunas até todos os zeros estarem riscados
   - Marcar o menor número possível de traços
4. **Atualizar custos**



Abordagem escolhida - Hungaro

1. **Normalização da matriz**
2. **Verificar se temos uma solução ótima**
3. **Cobrimento da matriz**
4. **Atualizar custos**
   - Seja X o menor número não riscado
   - Subtrair cada número não riscado por X
   - Somar cada número riscado duas vezes por X
   - Voltar ao passo 2



Abordagem escolhida - Hungaro

1. **Normalização da matriz**
2. **Verificar se temos uma solução ótima**
3. **Cobrimento da matriz**
4. **Atualizar custos**
   - Seja X o menor número não riscado
   - Subtrair cada número não riscado por X
   - Somar cada número riscado duas vezes por X
   - Voltar ao passo 2

0	**0**	**2**	**1**
4	0	0	3
0	**4**	**6**	**0**
4	0	0	0

Abordagem escolhida - Hungaro

1. Normalização da matriz

2. Verificar se temos uma solução ótima

- Há zeros de forma em que seja possível escolher N deles em linhas e colunas diferentes?
- Se sim, essa é uma solução ótima e o algoritmo encerra

3. Cobrimento da matriz

4. Atualizar custos

Alvo / Robô	A1	A2	A3	A4
R1	0	0	2	1
R2	4	0	0	3
R3	0	4	6	0
R4	4	0	0	0

Progresso e desafios

- **Primeiros passos**
  - BaseAgent
  - Point
  - Navigation
  - math
- **Algoritmo Hungaro**
  - Implementação
- **Obstáculos**
  - Ajuste de parâmetros
  - safe_distance
  - adjustment_factor
- **Post_decision**
  - Maximização

Progresso e desafios

- **Post_decision**
  - Maximização

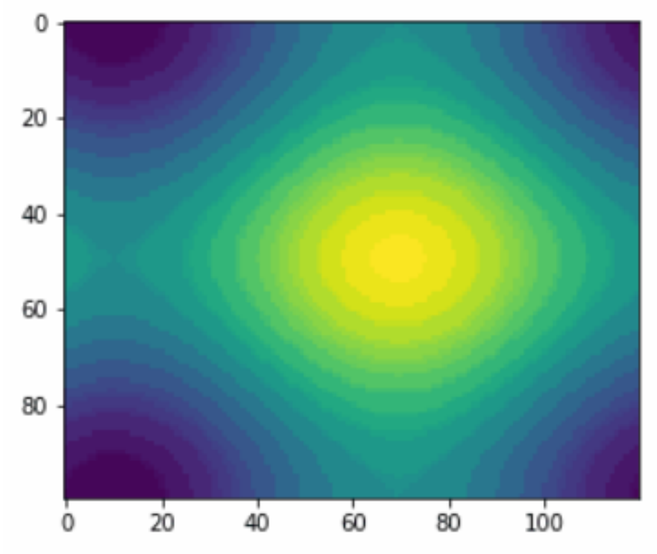
```
def post_decision(self):
    if self.id in self.assignment:
        return

    closest_target = None
    closest_distance = float('inf')
    current_position = Point(self.robot.x, self.robot.y)

    for target in self.targets:
        distance = current_position.dist_to(target)
        if distance < closest_distance:
            closest_distance = distance
            closest_target = target

    if not closest_target:
        return

    adjusted_target = self.avoid_obstacles(current_position, closest_target)
    target_velocity, target_angle_velocity = Navigation.goToPoint(self.robot, adjusted_target)
    velocity_factor = 0.5

    self.set_vel(target_velocity * velocity_factor)
    self.set_angle_vel(target_angle_velocity)
    return
```

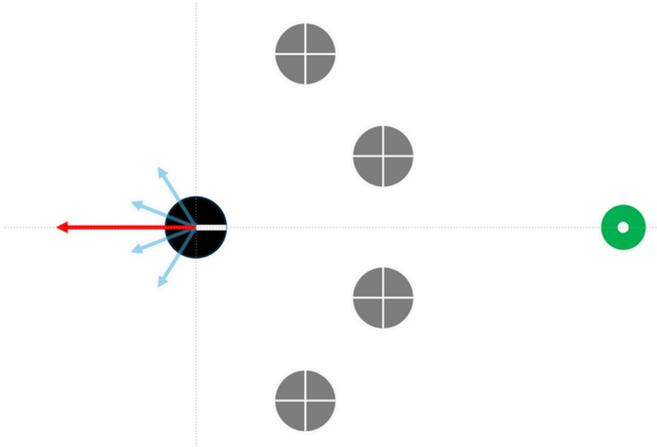
obs: versão sem comentários. não é a versão final

Possíveis Melhorias

- **Ajuste fino sistemático dos parâmetros – Desvio de obstáculos**
  - safe_distance e adjustment_factor
  - Potencialização máxima de eficiência
  - Otimização bayesianas
  - Algoritmos Evolutivos

- **Otimização escudos – Campos vetoriais**
  - Lidar com deadlocks





Resultado - Dificuldade 4

Link: https://drive.google.com/file/d/1aunDU92X5QLj2dbQduBKITqoFlZtEMHE/view?usp=sharing

RobôCIn

Desafio Seletiva

Candidato: Davi Sorrentino Brilhante