Relatório de Alterações nas Funções do Algoritmo Genético

Este documento apresenta, para cada função alterada no código do simulador e do algoritmo genético, uma descrição das modificações realizadas, a justificativa para cada mudança e o impacto esperado no desempenho do robô durante o treinamento.

---

1. `Robo.mover()`

Comportamento Original:

- Atualização de ângulo e velocidade simples.
- Colisão apenas zerava a velocidade e aplicava rotação aleatória.
- Consumo e recuperação de energia básicos.

Alterações Implementadas:

1. Substituição de cálculos manuais por `math.hypot` para precisão na distância.
2. Introdução de lógica de desvio de obstáculos baseada no centro do obstáculo mais próximo.
3. Adição de contador `passos_desde_coleta` para detectar quando o robô fica sem coletar recursos por muito tempo.
4. Manutenção de `self.tempo_parado` para forçar aceleração mínima e pequena rotação caso o robô fique parado >5 passos.

Justificativa:

- A precisão no cálculo de distâncias melhora a detecção de colisões e movimentos.
- A lógica de desvio baseada em centro de obstáculo direciona o robô de volta à área livre de forma mais eficiente.
- O contador de passos sem coleta evita loops improdutivos longe dos recursos.
- Forçar movimento previne que o robô trave em cantos ou obstáculos.

Impacto Esperado:

- Redução de colisões repetitivas.
- Maior eficiência na coleta de recursos e menor tempo parado.
- Melhor autonomia no alcance da meta.

---

2. `Robo.get_sensores()`

Comportamento Original:

- Cálculo de distâncias e ângulos para recurso, obstáculo e meta.

Alterações Implementadas:

1. Normalização de distâncias usando `np.hypot`.
2. Inclusão de novos sensores:
    - `tempo_parado`
    - `recursos_restantes`
    - `direcao_meta_x`, `direcao_meta_y` (vetor unitário até meta)
    - `direcao_recursos_x`, `direcao_recursos_y` (soma de vetores direção a todos recursos)
    - `recursos_cone_frontal` (razão de recursos no cone ±30°)
    - `passos_desde_coleta` (normalizado)

Justificativa:

- Novos sensores fornecem mais contexto ao indivíduo genético, permitindo decisões mais informadas (por exemplo, quando todos os recursos coletados, forçar retorno).
- Vetores unitários dão direção contínua em vez de apenas ângulos.
- Estatísticas de dispersão de recursos ajudam a guiar o robô para regiões de alta densidade.

Impacto Esperado:

- Árvore de decisão pode priorizar retornos à meta após coleta completa.
- Comportamentos mais sofisticados de navegação e coleta.

---

3. `Simulador.simular()`

Comportamento Original:

- Loop principal executa sempre a árvore genética para aceleração e rotação.

Alterações Implementadas:

1. Lógica de fallback: se `recursos_restantes == 0`, comandos derivados diretamente de `direcao_meta_x` e `angulo_meta`.
2. Tratamento de retorno de tupla (aceleração, rotação) quando ápice do operador `goto_meta`.
3. Condição de término refinada: encerra também quando recursos zerados **e** meta atingida.

Justificativa:

- Evita que o indivíduo genético continue tentando coletar quando não há recursos, economizando energia e tempo.
- Integração do novo operador `goto_meta` que retorna dupla de comandos.

Impacto Esperado:

- Simulações mais rápidas ao terminar a coleta e retornar eficientemente à meta.
- Aumento na taxa de sucesso de alcance da meta antes de zerar energia.

---

4. `IndividuoPG.criar_arvore_aleatoria()`

Comportamento Original:

- Operadores: `+`, , , `/`, `max`, `min`, `abs`, `if_positivo`, `if_negativo`.
- Folhas: constantes e variáveis básicas.

Alterações Implementadas:

1. Inclusão de novos operadores:
   - Lógicos: `and`, `or`, `not`.
   - Ternário: `if_then_else`.
   - Comportamento direto: `goto_meta`.
2. Reorganização da criação de subárvores para suportar ramos `then` / `else`.

Justificativa:

- Operadores lógicos oferecem condicional mais rico.
- `if_then_else` permite expressar decisões aninhadas.
- `goto_meta` fixa estratégia explícita de retorno ao objetivo.

Impacto Esperado:

- Maior expressividade da árvore genética e possibilidade de estratégias complexas.

---

5. `IndividuoPG.avaliar_no()`

Comportamento Original:

- Tratamento de `folha`, `abs`, `if_positivo`, `if_negativo` e operadores binários.

Alterações Implementadas:

1. Suporte a novos operadores:
   - `if_then_else`: avalia ramo correto.
   - `goto_meta`: retorna `(aceleracao, rotacao)` baseado em sensores.
   - `not`: nega valor arg.
   - `and`, `or` convertidos para `float`.
2. Unificação de tuple/raw em valor único quando necessário.

Justificativa:

- Adequar a avaliação à nova gama de operadores, entregando comandos compostos.

Impacto Esperado:

- Execução correta de estruturas condicionais complexas.

---

6. `IndividuoPG.mutacao_no()`

Comportamento Original:

- Mutação de nó com probabilidade fixa, alterando valores ou operadores básicos.

Alterações Implementadas:

1. Recursão específica para `if_then_else` com ramos `then` e `else`.

2. Lista de variáveis e operadores ampliada para incluir novos elementos lógicos e sensores.

Justificativa:

- Garantir que todas as estruturas condicionais sejam percorridas durante mutação.
- Permitir mutação de novos sensores na árvore.

Impacto Esperado:

- Maior diversidade genética e cobertura de novos comportamentos.

---

7. `IndividuoPG.crossover_no()`

Comportamento Original:

- Cópia simples de subárvore de um dos pais.

Alterações Implementadas:

1. Crossover com probabilidade de troca de subárvore (`p_corte`).
2. Tratamento de operadores `abs` , `not` , `if_then_else` e `goto_meta` durante reconciliação de ramos.
3. Uso de `copy.deepcopy` para preservar estruturas.

Justificativa:

- Melhor controle do ponto de corte e consistência das árvores misturadas.

Impacto Esperado:

- Filhos genéticos mais coerentes e preservação de estruturas complexas.

---

8. `ProgramacaoGenetica.__init__()`

Comportamento Original:

- Inicialização com `tamanho_populacao` e `profundidade` .

Alterações Implementadas:

1. Novos parâmetros: `metodo_selecao` e `elite_size` .
2. Inicialização de estatísticas: `media_fitness` , `std_fitness` , `diversidade` .

Justificativa:

- Flexibilizar seleção por torneio ou roleta e permitir elitismo.
- Monitorar métricas de convergência e diversidade.

Impacto Esperado:

- Melhoria no controle de exploração/exploração e análise de desempenho.

---

9. `ProgramacaoGenetica.avaliar_populacao()`

Comportamento Original:

- Avaliação média de fitness clássico (pontuação por recursos, distância, colisões e energia).

Alterações Implementadas:

1. Reward shaping:
   - `peso_recursos`, `peso_tempo`, `peso_proximidade`, penalidade de loop.
2. Cálculo de potencial Φ como soma de distâncias negativas para recursos.
3. Penalidade ao percorrer muitos pixels sem coletar.
4. Acúmulo de estatísticas de fitness e diversidade (via `difflib`).

Justificativa:

- Direcionar o aprendizado com recompensas imediatas e potenciais, reduzindo comportamento de looping.
- Medir diversidade estrutural da população.

Impacto Esperado:

- Convergência mais rápida a indivíduos de bom desempenho e menor estagnação.

---

10. `ProgramacaoGenetica.selecionar()`

Comportamento Original:

- Apenas torneio fixo (tamanho 3).

Alterações Implementadas:

1. Suporte a seleção por roleta (`selecionar_roleta`).
2. Manutenção de lógica de torneio como padrão.

Justificativa:

- Fornecer diversidade de métodos de seleção para evitar prematura convergência.

Impacto Esperado:

- Melhor balanceamento entre exploração e exploração.

---

11. `ProgramacaoGenetica.evoluir()`

Comportamento Original:

- Loop fixo de gerações, crossover e mutação simples.

Alterações Implementadas:

1. Mutação adaptativa com taxa decaindo exponencialmente.
2. Elitismo mantido via `elite_size` (absoluto ou percentual).
3. Log de métricas: melhor, média ± desvio e diversidade por geração.

Justificativa:

- Taxa de mutação decrescente favorece exploração inicial e refinamento final.
- Elitismo preserva indivíduos de alto fitness.

Impacto Esperado:

- Redução de ruído genético em estágios tardios e melhor refinamento.

---

12. `ProgramacaoGenetica.plotar_estatisticas()`

Descrição:

- Nova função para geração de gráfico contendo histórico do melhor fitness, média ± desvio-padrão e diversidade.

Justificativa:

- Facilitar análise visual do desempenho e convergência do algoritmo.

Impacto Esperado:

- Melhor diagnóstico e ajustes de parâmetros em futuras execuções.