RELATÓRIO TÉCNICO – (UNIDADE 3)

Disciplina: Inteligência Artificial - Turma 01 (2025.2)

Professor: Hendrik Macedo

Data: 23 de Fevereiro de 2026

Equipe (Membros Ativos):

1. Ellen Karolliny dos Santos
2. Ellen Vitoria Menezes Lima
3. João Santos Rocha
4. Larissa Batista dos Santos
5. Tasso Marcel de Oliveira

1. Definição do Problema

O desafio de modelagem e resolução computacional selecionado foi o de Navegação Autônoma de um Robô de Resgate em Terreno Perigoso.

Neste cenário, um agente autônomo (o robô) é introduzido em um ambiente desconhecido, o qual é mapeado como uma grade (grid bidimensional). O objetivo principal do robô é determinar o caminho ótimo do ponto de partida (aleatório) até o ponto de extração/recompensa (o alvo do resgate), minimizando o número de passos e evitando as armadilhas e perigos dispostos pelo terreno.

Do ponto de vista do aprendizado, o ambiente é dinâmico, sendo que cada célula da grade representa um estado distinto. O agente pode executar quatro ações cardinais (Mover para Cima, Baixo, Esquerda ou Direita). Como o robô opera sem um mapa prévio das recompensas ou punições, ele deve aprender as consequências de suas ações através da interação direta com o ambiente, seguindo uma estratégia de tentativa e erro.

2. Adequação do Algoritmo de IA (Q-Learning) ao Problema

O problema descrito se enquadra perfeitamente em um Processo de Decisão de Markov (MDP), exigindo a tomada de decisões sequenciais em um contexto de incerteza. Para sua solução, foi adotado o algoritmo Q-Learning, uma técnica de Aprendizado por Reforço Model-Free (livre de modelo). O Q-Learning é adequado à solução pelas seguintes razões:

- **Exploração vs. Utilização (Exploitation):** Por meio da política "epsilon-greedy" (ε-greedy), o agente é capaz de inicialmente explorar o mapa (focando na descoberta do terreno) e, subsequentemente, utilizar o conhecimento adquirido ao longo dos episódios para otimizar a rota.
- **Recompensas Esparsas:** O robô recebe sinais de recompensa (positivos ou negativos) somente ao final do trajeto. O Q-Learning, aplicando a Equação de Bellman e o fator de desconto gama (γ), propaga o valor dessa recompensa futura

para os estados imediatamente anteriores, estabelecendo uma "trilha de migalhas" matemática que orienta o agente.
- **Convergência:** Com uma taxa de aprendizado alfa (α) apropriada e um número suficiente de iterações, o algoritmo assegura a descoberta de uma política de navegação ótima, cuja eficácia é demonstrada no código-fonte desenvolvido.

3. Mapeamento: Pseudocódigo vs. Implementação

Com o intuito de satisfazer os critérios da avaliação, a implementação foi estruturada de forma a garantir a clara identificação da lógica central do Q-Learning. A arquitetura seguiu com uma correspondência lógica direta ao pseudocódigo acadêmico de referência.

Equação de Atualização do Valor Q (Q-Value): Q(s, a) = Q(s, a) + α * [R + γ * max Q(s', a') - Q(s, a)]

Passo a Passo da Implementação

- **Passo 1:** Inicializar Q(s,a) arbitrariamente (ex: com zeros para todos os pares estado-ação).
    - **No Código:** Ocorre dentro da classe de ambiente/setup, durante a geração da matriz bidimensional.
    - **Trecho Correspondente:** nova_linha.append([0, 0, 0, 0]) no método responsável pela criação inicial do mapa.
- **Passo 2:** Repetir (para cada episódio)
    - **No Código:** O loop principal de treinamento.
    - **Trecho Correspondente:** for ep in range(episodios): no método de treinamento.
- **Passo 3:** Inicializar o estado S
    - **No Código:** Definição da posição inicial aleatória no início de cada episódio, garantindo que não seja um estado terminal.
    - **Trecho Correspondente:** estado = self.inicio_mapa(...) (ou equivalente na classe de Ambiente).
- **Passo 4:** Repetir (para cada passo do episódio)
    - **No Código:** O loop de controle de passos até atingir o objetivo ou o limite de movimentos.
    - **Trecho Correspondente:** while not episodio_concluido and passos < max_passos:
- **Passo 5:** Escolher a ação A no estado S baseada na política derivada de Q (ex: ε-greedy)
    - **No Código:** Invocação da heurística de escolha de ação.
    - **Trecho Correspondente:** acao = self.escolher_acao(estado, ep), onde um valor aleatório é comparado com a propriedade self.greedy.
- **Passo 6:** Tomar a ação A, observar a recompensa R e o próximo estado S'
    - **No Código:** O agente se move e avalia a matriz para verificar a pontuação obtida na nova coordenada.
    - **Trechos Correspondentes:** novo_estado = self.mover(estado, acao) recompensa = self.obter_recompensa(novo_estado, acao)
- **Passo 7:** Atualizar Q(S,A) conforme a equação

    - **No Código:** O núcleo do algoritmo, onde o valor da ação da célula atual é atualizado com base no maior Q-value possível da próxima célula.
    - **Trecho Correspondente:** O método atualizar_valores(estado, acao, recompensa, novo_estado) traduz rigorosamente a fórmula: novo_valor = valor_atual + self.alpha * (recompensa + self.taxa_desc * max_futuro - valor_atual)
- **Passo 8:** S = S'
    - **No Código:** Transição do estado atual para a próxima iteração do loop while.
    - **Trecho Correspondente:** estado = novo_estado