Relatório Técnico
Avaliação Experimental de Agentes Inteligentes

Camila Andrade de Sena[1]

[1]Universidade Estadual do Ceará
Fortaleza, CE – Brasil

andrade.sena@aluno.uece.br

Abstract. *The adoption of Artificial Intelligence (AI) has shown significant growth in Brazil, impacting both the daily lives of citizens and the scientific field as well as the job market. In this context, the present report aims to outline fundamental concepts related to the development of intelligent agents, design reactive and model-based artificial agents, and evaluate their performance in simulated environments. The study includes the development of a simulator in the NetLogo environment, with the purpose of exploring the performance dynamics of agents in a scenario adapted from the classic "vacuum cleaner world." Finally, the experimentation covers different configurations of obstacles and dirt, allowing for the comparison between models and the analysis of their effectiveness.*

Resumo. *A adoção da Inteligência Artificial (IA) tem mostrado um crescimento significativo no Brasil, impactando tanto a rotina dos cidadãos quanto o campo científico e o mercado de trabalho. Nesse cenário, o presente relatório tem como objetivo delinear conceitos fundamentais relacionados ao desenvolvimento de agentes inteligentes, projetar agentes artificiais reativos e baseados em modelos, e avaliar seu desempenho em ambientes simulados. O estudo inclui o desenvolvimento de um simulador no ambiente NetLogo, com o intuito de explorar a dinâmica de desempenho de agentes em um cenário adaptado do clássico "mundo do aspirador de pó". Por fim, a experimentação abrange diferentes configurações de obstáculos e sujeira, permitindo a comparação entre os modelos e a análise de suas eficácias.*

1. Introdução

Norvig e Russell (2013) destacam que o campo da Inteligência Artificial (IA) surge de uma necessidade histórica do ser humano de compreender e, eventualmente, prever o mundo. O objetivo, no entanto, vai além da compreensão: trata-se de criar agentes inteligentes. Segundo os autores, um agente é definido como uma entidade capaz de perceber o ambiente por meio de sensores e agir sobre ele utilizando atuadores.

Em 2023, uma pesquisa da *Decision Report* revelou que quase 90% dos brasileiros já ouviram falar de IA, e mais da metade afirmaram que essas ferramentas impactam diretamente sua rotina diária. No âmbito científico, um relatório da Clarivate apontou que o Brasil está entre os 20 países que mais publicaram estudos sobre o tema entre 2019 e 2023, somando mais de 6.000 publicações. Essa tendência também é evidente no mercado de trabalho: em 2022, 41% das empresas brasileiras utilizavam IA, percentual que cresceu para mais de 70% entre pequenas e médias empresas em 2023

(DECISION REPORT, 2023; EXAME, 2022; MICROSOFT, 2024; MINISTÉRIO DA EDUCAÇÃO, 2024).

Dessa forma, considerando que o estudo de agentes inteligentes é uma parte essencial e integrada ao campo da Inteligência Artificial, este relatório se justifica, primeiramente, pelo fato de que a maioria dos sistemas computacionais são agentes artificiais concebidos para atingir objetivos em ambientes parcialmente observáveis. Além disso, o projetista de um agente artificial deve avaliar o desempenho do sistema e verificar se este apresenta inteligência, ou seja, se é racional. A avaliação de desempenho deve ser conduzida de forma criteriosa, a fim de determinar se o agente possui a capacidade de alcançar o objetivo para o qual foi projetado.

Dado o exposto, foram definidos os seguintes objetivos que guiarão este trabalho:

a) compreender os conceitos fundamentais necessários à aplicação da noção de agentes inteligentes na resolução de problemas em ambientes de tarefas de diversos tipos;

b) projetar programas de agentes artificiais reativos simples e reativos baseados em modelos, orientados por regras condição-ação, para resolver problemas em ambientes de tarefas determinísticos e parcialmente observáveis;

c) avaliar o desempenho de agentes inteligentes em ambientes de tarefas simulados em computador.

Assim, o relatório inicia com a apresentação do desenvolvimento de um simulador no ambiente NetLogo, com o objetivo de explorar a dinâmica de desempenho de agentes artificiais em um cenário adaptado do clássico "mundo do aspirador de pó". A proposta foca na criação de um ambiente modular, que possibilita a configuração flexível de parâmetros, como a presença ou ausência de obstáculos, a quantidade destes e a quantidade de sujeira.

Para a análise do desempenho no ambiente, foram consideradas duas métricas principais: a primeira recompensa o agente por cada célula limpa a cada intervalo de tempo, enquanto a segunda aplica uma penalização por movimentos.

Por fim, a experimentação incluiu diversas configurações iniciais de sujeira e obstáculos, registrando as pontuações dos agentes em cada execução, além da média global de desempenho. Esses resultados possibilitam comparar diferentes estratégias de tomada de decisão entre os agentes, bem como avaliar a eficácia de abordagens reativas e baseadas em modelos em contextos dinâmicos e desafiadores.

Espera-se que este estudo contribua para o avanço na compreensão do design e da avaliação de sistemas autônomos, oferecendo insights para o desenvolvimento de soluções práticas em inteligência artificial.

2. Implementação do Simulador

A plataforma utilizada no desenvolvimento foi o NetLogo, na versão 6.4, criada por Wilensky (1999). Trata-se de um ambiente de modelagem programável baseado em múltiplos agentes, que possibilita a criação e simulação de modelos complexos de sistemas dinâmicos. O código simula um agente aspirador de pó que navega por um ambiente de patches, limpando sujeira e evitando obstáculos.

Ele começa declarando variáveis globais para armazenar a pontuação do agente, o estado de sujeira no ambiente e a contagem de sujeira, além de variáveis específicas para os patches, que verificam se o patch foi limpo, visitado ou contém um obstáculo.

Figura 1. Declaração das variáveis.

```
;; variáveis globais
globals [
  score ;; pontuação do agente
  dirty? ;; variável auxiliar para checagem de sujeira
  dirty-count ;; variável auxiliar para contagem de sujeira
]

;; variáveis dos patches
patches-own [
  cleaned? ;; verifica se o caminho foi limpo
  visited? ;; verifica se o caminho já foi visitado
  obstacle? ;; verifica se existe obstáculo
]
```

A função **setup** inicializa o ambiente, limpando a tela, configurando as variáveis, criando patches e posicionando um agente (tartaruga) personalizado em um patch sem obstáculos.

Figura 2. Função setup.

```
to setup
  clear-all ;; limpa a tela

  ;; inicialização das variáveis
  set score 0
  set dirty-count num-dirty
  set dirty? false

  setup-patches ;; chama a configuração do patch

  ;; cria o aspirador em um lugar sem obstáculo
  create-turtles 1 [
    set shape "truck cab top"
    set color red
    set size 3
    move-to one-of patches with [not obstacle?]
  ]

  reset-ticks ;; limpa o tempo
end
```

Patches têm sua cor inicial definida como branca, representando espaços limpos, e recebem atributos que indicam se são obstáculos, gerados por funções específicas para criar bordas e obstáculos aleatórios. Também são adicionados patches marrons para representar sujeira, com uma quantidade configurada pelo usuário.

Figura 3. Criação e configuração dos patches, obstáculos e sujeiras.

```
to setup-patches
  ;; limpa o caminho
  ask patches [
    set pcolor white
    set cleaned? false
    set visited? false
    set obstacle? false
  ]

  create-obstacle-border
  create-random-obstacles
  place-dirty-patches
end

;; cria uma parede de obstáculos
to create-obstacle-border
  ask patches with [pxcor = max-pxcor or pxcor = min-pxcor or pycor = max-pycor or pycor = min-pycor] [
    set obstacle? true
  ]
end

;; cria obstáculos aleatórios se o usuário ativar a opção
to create-random-obstacles
  if obstacles? [
    ask n-of num-obstacles patches with [not obstacle?] [
      set obstacle? true
      set pcolor gray
    ]
  ]
end

;; cria uma quantidade de sujeira escolhida pelo usuário em locais aleatórios
to place-dirty-patches
  ask n-of num-dirty patches with [not obstacle?] [
    set pcolor brown
  ]
end
```

Na função **go**, o tipo de agente é escolhido pelo usuário na interface, podendo ser *Simple reflex* ou *Model-based*. O agente age em ciclos (ticks) até que toda a sujeira seja limpa ou não existam mais patches sujos.

Figura 4. Função go.

```
to go
  ;; verifica qual agente foi escolhido pelo usuário
  if agent = "Simple reflex" [ask turtles [simple-reflex-move]]
  if agent = "Model-based" [ask turtles [model-based-move]]
  if dirty-count = 0 or not any? patches with [pcolor = brown] [stop]
  tick
end
```

Como afirmam Norvig e Russell (2013), **agentes reativos simples** selecionam ações com base em percepções diretas do ambiente, sem a necessidade de memória ou histórico de ações passadas. Dessa forma, a função **simple-reflex-move** implementa um comportamento básico de um agente aspirador que segue uma lógica de "reflexo simples", reagindo diretamente às condições de seu ambiente imediato.

Primeiramente, ela verifica se o patch em que o agente está atualmente contém sujeira (representada pela cor marrom, pcolor = brown). Caso haja sujeira, o agente

limpa o patch, alterando a cor para branco (set pcolor white), atualiza a pontuação, aumentando em 1000 pontos, diminui a quantidade de sujeira restante (set dirty-count dirty-count - 1), e marca o patch como limpo (set cleaned? true).

Em seguida, o agente tenta se mover para um dos patches vizinhos que não tenha obstáculos (não tenha a variável obstacle? ativada) e não seja cinza, que indica uma área bloqueada. Isso é feito verificando a lista de possíveis movimentos (neighbors4) e filtrando aqueles que atendem a essas condições. Se existir ao menos um movimento válido, o agente escolhe um desses patches e se move para ele. Caso contrário, se não houver movimento possível, o agente gira aleatoriamente (usando o comando rt random 360), explorando novas direções.

Ao fim, o agente perde 1 ponto de sua pontuação a cada movimento.

Figura 5. Agente reativo simples.

```
to simple-reflex-move
  ;; verifica se há sujeira no patch atual
  if [pcolor] of patch-here = brown [
    ;; limpa o patch e atualiza as variáveis
    set pcolor white
    set score score + 1000
    set dirty-count dirty-count - 1
    set cleaned? true
  ]
  ;; tenta se mover para um patch limpo e sem obstáculo
  let possible-moves neighbors4 with [not obstacle? and pcolor != gray]
  if any? possible-moves [
    move-to one-of possible-moves
  ]
  ;; se não houver movimentos válidos, gira em uma direção aleatória
  if not any? possible-moves [
    rt random 360
  ]
  set score score - 1
end
```

Agentes reativos baseados em modelo, em contraste com o agente anterior, mantêm um modelo interno de seu ambiente ou de seu estado atual, o que lhes permite aprimorar o desempenho e se adaptar de maneira mais eficaz a diferentes situações (RUSSELL, S; NORVIG, P., 2013). Dessa forma, a função **model-based-move** implementa um comportamento mais sofisticado e planejado para o agente aspirador, baseado em um modelo de decisão. Em vez de agir de forma reativa, como no **simple-reflex-move**, o agente considera o histórico de suas ações e busca uma estratégia para explorar o ambiente de maneira mais eficiente.

O processo começa com a atualização do estado do modelo: o agente verifica o patch em que está posicionado (definido por patch-here) e marca esse patch como "visitado" (set visited? true). Essa marcação impede que o agente escolha um patch já visitado novamente, priorizando a exploração de áreas não visitadas.

Se o agente estiver em um patch sujo (onde a cor do patch é marrom, pcolor = brown), ele limpa o patch, muda sua cor para branco (set pcolor white), atualiza o

estado do patch para indicar que foi limpo (set cleaned? true), e aumenta sua pontuação em 1000 pontos. Também diminui a contagem de sujeira (set dirty-count dirty-count - 1), refletindo a remoção de sujeira.

A decisão sobre o próximo movimento é feita com base em uma análise do ambiente. O agente verifica os vizinhos ao redor (usando neighbors4, que representa os quatro patches adjacentes). Ele filtra os vizinhos que não são obstáculos (not obstacle?) e que ainda não foram visitados (not visited?). Se houver ao menos um vizinho que atenda a essas condições, o agente escolhe mover-se para um desses patches não visitados, priorizando a exploração de novas áreas. Caso todos os vizinhos já tenham sido visitados, o agente muda sua estratégia, movendo-se para qualquer vizinho sem obstáculos, mesmo que já tenha sido visitado.

Se não houver nenhum movimento possível (quando todos os vizinhos são obstáculos ou estão fora do alcance), o agente gira aleatoriamente, utilizando rt random 360, para explorar novas direções de forma aleatória.

Ao final, o agente perde 1 ponto de sua pontuação a cada movimento.

Figura 6. Agente reativo baseado em modelo.

```
to model-based-move
  ;; Atualiza o estado do modelo com base no patch atual
  let current-patch patch-here
  ask current-patch [
    set visited? true
  ]

  ;; Verifica e limpa sujeira no patch atual
  if [pcolor] of current-patch = brown [
    set pcolor white
    set cleaned? true
    set score score + 1000
    set dirty-count dirty-count - 1
  ]

  ;; Decide o próximo movimento com base no modelo
  let possible-moves neighbors4 with [not obstacle? and not visited?]
  ifelse any? possible-moves [
    ;; Prioriza mover para um patch ainda não visitado
    move-to one-of possible-moves
  ] [
    ;; Caso todos os vizinhos já tenham sido visitados, move-se para qualquer vizinho sem obstáculo
    set possible-moves neighbors4 with [not obstacle?]
    ifelse any? possible-moves [
      move-to one-of possible-moves
    ] [
      ;; Se não houver movimentos possíveis, gira aleatoriamente para explorar
      rt random 360
    ]
  ]

  ;; Penaliza movimento
  set score score - 1
end
```

Com o código finalizado, as demais configurações e a execução do ambiente são realizadas por meio da Interface. Inicialmente, é possível configurar a quantidade de sujeira (de 0 a 100) utilizando o primeiro Deslizador. Em seguida, o usuário pode optar por adicionar ou não obstáculos, através de um Interruptor, e definir a quantidade desses obstáculos (de 0 a 100) por meio de outro Deslizador. O Selecionador permite a escolha entre o agente simples ou o agente baseado em modelo. Ao final, há o botão Setup para

configurar o ambiente e o botão Go para executar a simulação. O Mostrador abaixo exibe a pontuação obtida em cada simulação.

A Interface foi utilizada para realizar os ajustes necessários nos experimentos, que são apresentados a seguir.

3. Testes e Experimentação

Em relação à experimentação, foram definidos os seguintes cenários para comparar o desempenho de ambos os agentes em cinco simulações. As pontuações obtidas serão registradas em uma tabela, a partir da qual será calculada a média final.

1. 0 obstáculo, 10 sujeira;
2. 0 obstáculo, 50 sujeira;
3. 0 obstáculo, 100 sujeira;
4. 50 obstáculo, 10 sujeira;
5. 50 obstáculo, 50 sujeira;
6. 50 obstáculo, 100 sujeira;
7. 100 obstáculo, 10 sujeira;
8. 100 obstáculo, 50 sujeira;
9. 100 obstáculo, 100 sujeira.

Os resultados estão apresentados a seguir (*Nota*: Outliers e simulações nas quais o agente entrou em loop, devido à impossibilidade de alcançar a sujeira, foram desconsideradas).

Tabela 1. 0 obstáculo, 10 sujeira.

Tipo de Agente	Simulações					Média Global
	1	2	3	4	5	
Agente Simples	5341	5480	7898	4163	4335	5443
Agente Baseado em Modelo	7996	7766	1801	6889	8780	6646

Tabela 2. 0 obstáculo, 50 sujeira.

Tipo de Agente	Simulações					Média Global
	1	2	3	4	5	
Agente Simples	23027	29608	40983	40530	34306	33691
Agente Baseado em Modelo	44930	46734	44071	44062	45602	45080

Tabela 3. 0 obstáculo, 100 sujeira.

Tipo de Agente	Simulações					Média Global
	1	2	3	4	5	
Agente Simples	78451	78735	85566	87568	90782	84220
Agente Baseado em Modelo	92150	96697	95677	96385	95599	95302

Tabela 4. 50 obstáculo, 10 sujeira.

Tipo de Agente	Simulações					Média Global
	1	2	3	4	5	
Agente Simples	1299	2911	1643	3745	3684	2656
Agente Baseado em Modelo	7496	6391	5664	6875	8066	6898

Tabela 5. 50 obstáculo, 50 sujeira.

Tipo de Agente	Simulações					Média Global
	1	2	3	4	5	
Agente Simples	26869	30287	36270	32941	36150	32503
Agente Baseado em Modelo	45811	46763	46786	27053	47753	42833

Tabela 6. 50 obstáculo, 100 sujeira.

Tipo de Agente	Simulações					Média Global
	1	2	3	4	5	
Agente Simples	82432	91326	83315	75346	91491	84782
Agente Baseado em Modelo	93418	95106	93923	95643	90520	93722

Tabela 7. 100 obstáculo, 10 sujeira

Tipo de Agente	Simulações					Média Global
	1	2	3	4	5	
Agente Simples	1793	1930	3489	2249	1382	2169
Agente Baseado em Modelo	7772	4085	8998	8986	7265	7421

Tabela 8. 100 obstáculo, 50 sujeira.

Tipo de Agente	Simulações					Média Global
	1	2	3	4	5	
Agente Simples	29446	38558	35291	36357	40311	35993
Agente Baseado em Modelo	44931	47557	44510	45496	46540	45807

Tabela 9. 100 obstáculo, 100 sujeira.

Tipo de Agente	Simulações					Média Global
	1	2	3	4	5	
Agente Simples	87148	85050	80533	86688	83932	84670
Agente Baseado em Modelo	89259	94899	97034	95489	95495	94435

Os resultados e as conclusões obtidos após cerca de 100 simulações são apresentados no capítulo seguinte.

4. Resultados e Conclusões

Em primeiro lugar, é fundamental destacar que as experimentações práticas confirmaram a superioridade do agente baseado em modelo em comparação ao agente simples, evidenciada por sua média de pontuação mais alta em todas as simulações. A estratégia de armazenar um histórico na memória mostrou-se altamente eficiente, resultando em pontuações significativamente maiores nesse caso.

Figura 7. Pontuações dos agentes por cenário.

Cabe ressaltar que, devido ao sistema de pontuação adotado, a pontuação dos agentes aumentou proporcionalmente à quantidade de sujeiras presentes. Essa diferença foi, de fato, o principal fator de distinção entre os cenários analisados. Ademais, verificou-se que, quanto maior a quantidade de sujeiras, independentemente da presença de obstáculos, mais acentuada foi a discrepância entre as pontuações do agente simples e do agente baseado em modelo. Este último apresentou um desempenho consistentemente superior.

Finalmente, destaca-se o desempenho mais eficiente do agente baseado em modelo em todos os cenários sem obstáculos, provavelmente decorrente de sua capacidade de lembrar os locais já visitados.

Como sugestão para pesquisas futuras, recomenda-se o desenvolvimento de um sistema de pontuação mais equilibrado e a adoção de modelos que realizem um número menor de movimentos, permitindo uma avaliação mais precisa do desempenho dos agentes.

Referências

DECISION REPORT. **Inteligência artificial já impacta o dia a dia de 54% dos brasileiros, diz pesquisa**. [S. l.], 14 mar. 2023. Disponível em: https://decisionreport.com.br/inteligencia-artificial-ja-impacta-o-dia-a-dia-de-54-dos-brasileiros-diz-pesquisa/. Acesso em: 20 nov. 2024.

EXAME. **Pesquisa mostra que 41% das empresas brasileiras já utilizam Inteligência Artificial**. [*S. l.*], 2022. Disponível em: https://exame.com/esferabrasil/pesquisa-mostra-que-41-das-empresas-brasileiras-ja-utilizam-inteligencia-artificial/. Acesso em: 20 nov. 2024.

MICROSOFT. **Inteligência artificial já é parte do dia a dia de 74% das MPMEs brasileiras**. [*S. l.*], 4 mar. 2024. Disponível em: https://news.microsoft.com/pt-br/inteligencia-artificial-ja-e-parte-do-dia-a-dia-de-74-das-mpmes-brasileiras/. Acesso em: 20 nov. 2024.

MINISTÉRIO DA EDUCAÇÃO (Brasil). CAPES. **Brasil produz 6,3 mil estudos sobre inteligência Artificial**. [*S. l.*], 23 ago. 2024. Disponível em: https://www.gov.br/capes/pt-br/assuntos/noticias/brasil-produz-6-3-mil-estudos-sobre-inteligencia-artificial. Acesso em: 20 nov. 2024.

RUSSELL, Stuart; NORVIG, Peter. **Inteligência Artificial**. [S. l.]: GEN LTC, 2013.

WILENSKY, Uri. **Netlogo**. Evanston, IL: Center for Connected Learning and Computer-Based Modeling, Northwestern University, 1999. Disponível em: https://ccl.northwestern.edu/netlogo/. Acesso em: 20 nov. 2024.