

Disciplina de Inteligência Artificial e Robótica

Material com atividades de laboratório

Fabricio Barth

https://insper.github.io/ia_bcc/

Conteúdo

1. Inteligência Artificial e Robótica - 2025/1	4
1.1 Horário das aulas	4
1.2 Horário de atendimento	4
1.3 Informações adicionais sobre os projetos e implementações	4
1.4 Contato	4
2. Introdução	5
3. Ementa	5
3.1 Objetivos	5
3.2 Conteúdo Programático	5
3.3 Bibliografia Básica	5
3.4 Bibliografia Complementar	6
4. Plano de aula	7
4.1 Introdução à Inteligência Artificial	7
4.2 Busca em Espaços de Estados	8
4.3 Busca Heurística	9
4.4 Ambientes Competitivos	10
4.5 Projeto Intermediário	10
4.6 Aprendendo políticas	11
4.7 Robótica	11
5. Atividades e Critérios de aprovação	13
5.1 N exercícios sobre agentes autônomos	13
5.2 2 projetos	13
5.3 Conversão de conceito para valor numérico	14
6. Aulas	15
6.1 Introdução	15
6.2 Busca em espaço de estados	20
6.3 Busca com heurística	34
6.4 Programação por restrições	43
6.5 Ambientes competitivos	45
6.6 Aprendizagem por reforço	46
7. Projetos atuais	58
7.1 Taxi Driver	58
8. Referências	63
8.1 Introdução à Inteligência Artificial	63
8.2 Algoritmos de Busca	64

8.3	Constraint Satisfaction Problems	65
8.4	Busca Competitiva	66
8.5	Jogos de tabuleiro e busca competitiva	67
9.	Exercícios e avaliações antigas	68
9.1	Exercícios adicionais a avaliações antigas	68

1. Inteligência Artificial e Robótica - 2025/1

1. [Ementa da disciplina.](#)
2. [Plano aula-a-aula da disciplina.](#)
3. [Critérios de avaliação.](#)

1.1 Horário das aulas

Segundas e quartas das 07:30 até 09:30.

1.2 Horário de atendimento


Segundas das 10:00 até 11:30.

1.3 Informações adicionais sobre os projetos e implementações

- Todas os projetos e implementações irão utilizar a linguagem de programação **Python**. Sendo assim, espera-se que os alunos desta disciplina tenham conhecimento de programação em Python.
- As entregas dos projetos e exercícios serão via [Github Classroom](#)

1.4 Contato

Em caso de dúvida ou comentários, favor encaminhar e-mail para [fabriciojb at insper.edu.br](mailto:fabriciojb@insper.edu.br).

 January 28, 2025

2. Introdução

Aplicações modernas apontam para um conceito de Inteligência Artificial (IA) voltado para duas principais características: **autonomia** e **adaptabilidade**. Autonomia é a habilidade de executar tarefas em contextos complexos sem constante intervenção do ser humano e adaptabilidade é a habilidade de melhorar seu desempenho aprendendo com a experiência.

3. Ementa

Introdução à Inteligência Artificial; Definições de Agente Autônomo; Arquitetura computacional de agentes e o laço percepção - planejamento e ação; Caracterização de Ambientes; Resolução de problemas usando espaço de busca; Estratégias de busca; Algoritmos de busca cega e informados; Conceito de Heurística; Teoria de Jogos e Ambientes Competitivos; Aprendizagem por Reforço; Percepção, sensores e incerteza; Noções de visão computacional e reconhecimento de padrões; Aplicação comercial de robôs e usos emergentes, soluções de plataformas robóticas e de software para robôs (R.O.S, OpenCV).

3.1 Objetivos

Ao final da disciplina o estudante será capaz de:

1. Descrever os conceitos, técnicas e métodos para o desenvolvimento de Agentes Autônomos.
2. Identificar quais tipos de problemas podem ser resolvidos através do uso de Agentes Autônomos.
3. Criar soluções para alguns problemas clássicos desta área.
4. Especificar, desenvolver e testar projetos que façam uso de Agentes Autônomos para resolver problemas complexos.
5. Planejar e executar um trabalho em equipe, fornecendo e assimilando devolutivas.

3.2 Conteúdo Programático

1. Definições de Agente Autônomo e resolução de problemas.
2. Estratégias de busca: algoritmos de busca cega e algoritmos informados.
3. Heurísticas.
4. Implementação de agentes autônomos utilizando estratégias de busca.
5. Programação por restrições (CSP).
6. Ambientes competitivos e teoria de jogos.
7. Algoritmo Min-Max e função de utilidade.
8. Implementação de agentes autônomos para ambientes competitivos.
9. Aprendizagem por Reforço.
10. Implementação de agentes autônomos usando aprendizagem por reforço.
11. Algoritmo Q-Learning.
12. Implementações de agentes autônomos usando o projeto Gym.
13. Implementação de um agente robótico.

3.3 Bibliografia Básica

1. NORVIG, P.; RUSSELL, S., Inteligência Artificial, 3ª ed., Campus Elsevier, 2013

3.4 Bibliografia Complementar

1. O'KANE, J., A Gentle introduction to ROS, CreateSpace Publishing, 2013
2. SIEGWART, R.; NOURBAKHS, I. R.; SCARAMUZZA, D., Introduction to Autonomous Mobile Robots., 2ª ed., MIT Press, 2011
3. SILVER, D.; SINGH S.; PRECUP D.; SUTTON R. Reward is enough. Artificial Intelligence. Vol 299, 2021. Disponível em <https://doi.org/10.1016/j.artint.2021.103535>.
4. SILVER, D.; HUBERT T.; SCHRITTWIESER, J.; ANTONOGLOU, I.; LAI, M.; GUEZ, A. [A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play](#). Science 362, 1140-1144 (2018).

 July 23, 2024

4. Plano de aula

O plano de aula desta disciplina está dividido em **seis** (6) blocos. Para cada bloco as seguintes atividades estão planejadas.

Atenção!

O programa está sempre sujeito a alterações e adaptações conforme a disciplina é executada.

4.1 Introdução à Inteligência Artificial

Data	Fundamentos / Conteúdo	Dinâmica
03-Fev	Apresentação da disciplina e critérios de avaliação.	Dinâmica
	Introdução à IA e ao conceito de agente autônomo	em grupo
05-Fev	Revisão do conceito de agente autônomo, discussão sobre diferenças de agente autônomo e software convencional, e propriedades de ambientes.	Dinâmica em grupo

O conteúdo associado a este bloco é [1](#) e [2](#)

4.2 Busca em Espaços de Estados

Data	Fundamentos / Conteúdo	Dinâmica
10-Fev	Resolução de problemas através de espaço de busca.	Exercícios em sala de aula onde os alunos são convidados a definir estado, transição, estado meta e custo da solução encontrada para diversos problemas.
12-Fev	Estratégias de busca. Algoritmos de busca cegos (largura, profundidade, iterativo, custo uniforme). Critérios de comparação entre os algoritmos.	Implementação dos algoritmos de busca e dos agentes autônomos em Python para resolver alguns problemas clássicos da literatura.
17-Fev	Estratégias de busca. Algoritmos de busca cegos (largura, profundidade, iterativo, custo uniforme). Critérios de comparação entre os algoritmos.	Implementação dos algoritmos de busca e dos agentes autônomos em Python para resolver alguns problemas clássicos da literatura.
19-Fev	Estratégias de busca. Algoritmos de busca cegos (largura, profundidade, iterativo, custo uniforme). Critérios de comparação entre os algoritmos.	Implementação dos algoritmos de busca e dos agentes autônomos em Python para resolver alguns problemas clássicos da literatura.

O conteúdo associado a este bloco é [3](#), [4](#), [5](#), [6](#) e [7](#).

4.3 Busca Heurística

Data	Fundamentos / Conteúdo	Dinâmica
24-Fev	Estratégia de busca. Algoritmos de busca **informados** (busca gananciosa, A^* , família subida da montanha). Função heurística. Comparação entre os algoritmos.	Implementação dos algoritmos de busca e dos agentes autônomos em Python para resolver alguns problemas clássicos da literatura.
26-Fev	Estratégia de busca. Algoritmos de busca **informados** (busca gananciosa, A^* , família subida da montanha). Função heurística. Comparação entre os algoritmos.	Implementação dos algoritmos de busca e dos agentes autônomos em Python para resolver alguns problemas clássicos da literatura.
10-Mar	Estratégia de busca. Algoritmos de busca **informados** (busca gananciosa, A^* , família subida da montanha). Função heurística. Comparação entre os algoritmos.	Implementação dos algoritmos de busca e dos agentes autônomos em Python para resolver alguns problemas clássicos da literatura.
12-Mar	Desenvolvimento de um agente autônomo que atua em um ambiente discreto, determinístico, síncrono, simulado e <i>*single agent*</i> .	Implementação de um projeto, provavelmente, envolvendo algum framework de simulação (i.e., Gym Open AI).

O conteúdo associado a este bloco é [8](#), [9](#), [10](#) e [11](#).

4.4 Ambientes Competitivos

Data	Fundamentos / Conteúdo	Dinâmica
17-Mar	Constraint Satisfaction Problems	Implementação de um agente autônomo capaz de identificar estados que satisfazem determinadas restrições.
19-Mar	Jogos de tabuleiro, busca competitiva, algoritmo min-max e função de utilidade.	Implementação de um agente autônomo que deverá atuar em um ambiente competitivo, determinístico e completamente observável.
24-Mar	SEMANA DE PROVAS	SEMANA DE PROVAS - Prova Intermediária
26-Mar	SEMANA DE PROVAS	SEMANA DE PROVAS - Prova Intermediária

O conteúdo associado a este bloco é [12](#) e [13](#).

4.5 Projeto Intermediário

Data	Fundamentos / Conteúdo	Dinâmica
31-Mar	Desenvolvimento de um agente autônomo que atua em um ambiente discreto, determinístico, síncrono, simulado e *single agent* ou *multi-agent*.	Desenvolvimento de projeto em sala de aula
02-Abr	Desenvolvimento de um agente autônomo que atua em um ambiente discreto, determinístico, síncrono, simulado e *single agent* ou *multi-agent*.	Desenvolvimento de projeto em sala de aula
07-Abr	Studio	Desenvolvimento de projeto em sala de aula

O conteúdo associado a este bloco é [14](#).

4.6 Aprendendo políticas

Data	Fundamentos / Conteúdo	Dinâmica
09- Abr	Definição de aprendizagem por reforço, política de controle e algoritmo Q-Learning.	Discussão em sala. Exercícios em sala de aula envolvendo o ambiente OpenAI Gym. Implementação de agentes autônomos usando o algoritmo Q-Learning.
14- Abr	Algoritmo Q-Learning: detalhes e hiperparâmetros. Apresentação do ambiente OpenAI Gym.	Exercícios em sala de aula envolvendo o ambiente OpenAI Gym. Implementação de agentes autônomos usando o algoritmo Q-Learning.
16- Abr	Algoritmo Q-Learning: detalhes e hiperparâmetros.	Implementação de agentes autônomos usando o algoritmo Sarsa.
23- Abr	Ambientes não-determinísticos. Reinforcement Learning: métodos tabulares	Implementação de agentes autônomos usando o algoritmo Q-Learning e Sarsa
28- Abr	Ambientes não-determinísticos. Reinforcement Learning: métodos tabulares	Implementação de agentes autônomos usando o algoritmo Q-Learning e Sarsa

O conteúdo associado a este bloco é 15, 16, 17 e 18.

4.7 Robótica

Data	Fundamentos / Conteúdo	Dinâmica
30- Abr	Visão geral sobre robótica e framework ROS2	Visão geral sobre robótica e framework ROS2
05- Mai	Desenvolvimento de um agente robótico (físico).	Implementação de um projeto envolvendo um robô físico
07- Mai	Desenvolvimento de um agente robótico (físico).	Implementação de um projeto envolvendo um robô físico
12- Mai	Avaliação Final da disciplina	Avaliação Final da disciplina
14- Mai	Avaliação Final da disciplina	Avaliação Final da disciplina

🕒 April 22, 2025

5. Atividades e Critérios de aprovação

Os objetivos de aprendizagem desta disciplina serão avaliados através das seguintes atividades:

Atividade	Peso
N exercícios (APS) sobre agentes autônomos	10%
2 projetos	30%
Avaliação Intermediária	30%
Avaliação Final	30%

O critério para aprovação é:

- nota final superior ou igual a cinco (5);
- a média das avaliações intermediária e final deve ser igual ou maior que cinco (5);
- e 75% de frequência mínima nas aulas.

5.1 N exercícios sobre agentes autônomos

Seguem os enunciados que se encaixam nesta categoria:

Descrição	Prazo para entrega
Exercício sobre modelagem de agentes	10/02
Exercício sobre busca em largura e profundidade	24/02
Procurando caminhos em um mapa	10/03
Puzzle-8	23/03
Q-Learning e hiperparâmetros	15/04
Sarsa	22/04

5.2 2 projetos

Seguem os enunciados que se encaixam nesta categoria:

Descrição	Prazo para entrega
Taxi Driver anfíbio	11/04/2025

5.3 Conversão de conceito para valor numérico

O resultado de algumas avaliações poderá adotar conceitos (A+, B,..., I) ao invés de um valor numérico. Para estes casos será utilizada a seguinte tabela de conversão:

A+	A	B	C	D	I
10	9	7	5	4	2

- **Insuficiente (I):** não entregou, entregou algo que não exigiu esforço ou desviou do objetivo.
- **Em Desenvolvimento (D):** entregou algo que estava de acordo com o objetivo e exigiu algum esforço, mas possui problemas e limitações importantes. O mínimo aceitável não foi atingido.
- **Básico (C):** o mínimo aceitável do objetivo medido foi alcançado, mas o desempenho foi abaixo do esperado. É suficiente para aprovação.
- **Esperado (B):** atingiu o desempenho esperado.
- **Avançado (A):** o desempenho foi acima do esperado. É algo desejável, mas não é motivo de preocupação se não for alcançado.

🕒 April 16, 2025

6. Aulas

6.1 Introdução

6.1.1 Introdução à Inteligência Artificial

O objetivo desta aula é responder as seguintes perguntas:

- O que é inteligência artificial (IA) geral ou profunda?
- O que é inteligência artificial fraca?
- Quais são as principais aplicações de IA?
- Qual é a definição de agente autônomo?
- Qual é a relação de agente autônomo com IA?
- Quais as funcionalidades que um agente autônomo precisa ter e como podemos desenvolver tais funcionalidades?
- Qual é o escopo desta disciplina? Ver [ementa](#).
- Quais são as outras disciplinas do curso que estão relacionadas com este tópico?

As respostas para as perguntas acima serão construídas de forma colaborativa em sala de aula.

Material de referência

Material com exemplos de ficção científica:

Material com exemplos de aplicações reais:

Uma apresentação sobre a evolução da IA:

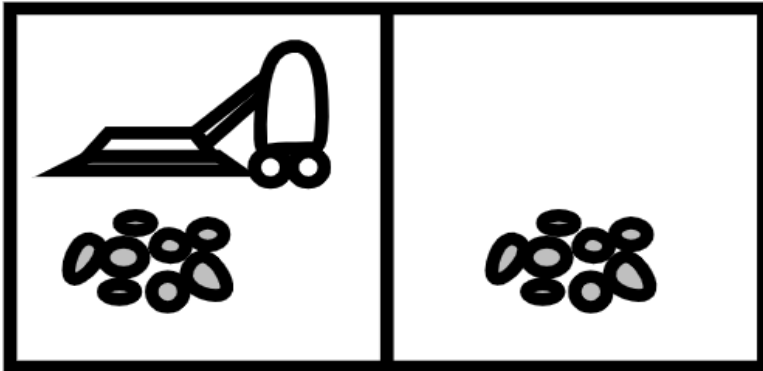
🕒 January 30, 2024

6.1.2 Agentes Autônomos

Exemplo do aspirador de pó

Um robô aspirador de pó deve limpar uma casa com duas posições. As operações que ele sabe executar são:

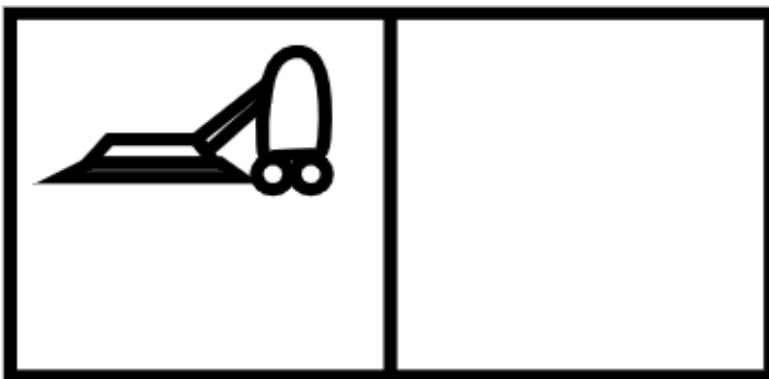
- sugar
- ir para a posição da esquerda
- ir para a posição da direita



Em **grupo** de 4 pessoas responda as seguintes perguntas (tempo de 15 minutos):

- O que é relevante representar nos estados do mundo? Como os estados são estruturados (estrutura de dados) e qual o significado dela (dos campos)?
- Quais são os estados possíveis do mundo do aspirador e as suas transições?
- Quais são as consequências das ações sobre os estados? As ações são determinísticas?
- É possível desenhar o **grafo** com todos os estados e transições para este problema?
- Apresente uma solução possível. Uma sequência de ações que fazem o robô sair do estado inicial e chegar no estado final.

Considere como estado final a situação ilustrada abaixo:



Prepararem-se para eventualmente apresentar as suas respostas.



Conceitos que devem ser explorados neste exercício

Representação de um problema na forma de **estado, transição, grafo**

🕒 February 10, 2023

6.1.3 Implementando um Aspirador de Pó

Importante!

As orientações sobre a entrega deste exercício estão abaixo. Este é um exercício individual. Os alunos poderão discutir sobre o problema, mas a implementação deve ser feita individualmente. Não pode ser utilizado nenhum tipo de ferramenta de Inteligência Artificial para a resolução destes problemas.

Configuração do ambiente

Para executar esta atividade você terá que criar um projeto e como dependência deste projeto instalar a biblioteca [AIGYM](#):

```
pip install aigyminspers
```

Esta biblioteca implementa diversos algoritmos de busca que serão utilizados ao longo da disciplina. Além disso, esta biblioteca permite o desenvolvimento de agentes inteligentes através de uma interface padrão.

Exercício: Aspirador de Pó

Para este exercício vamos utilizar o arquivo [AgentSpecification.py](#).

- Faça uma cópia deste arquivo e chame ele de `AspiradorPo.py`, por exemplo.
- No método `__init__` defina a estrutura de dados que você especificou na aula passada.
- No método `successors` codifique a execução das ações segundo o que você especificou na aula passada. **Dica:** cada novo estado é uma instância de um novo objeto da classe que é adicionado na lista retornada pelo método.
- Codifique o comportamento do método `is_goal`. Ele deve retornar `True` quando encontrar um estado igual a meta.
- O método `description()` retorna uma descrição do problema que está sendo resolvido. Por favor, altere o que está descrito no método.
- Faça um retorno para o método `env()`. Peça para o professor explicar em sala de aula.
- Altere o método `main()` para que o mesmo chame a sua classe.

Conceitos que devem ser explorados neste exercício

Representação de um problema na forma de **estado, transição, grafo**

Conceitos que devem ser explorados neste exercício

Algoritmos de Busca, começando com o algoritmo de busca em largura.

Exercício 2: Aspirador de Pó com 4 quartos.

Vamos implementar um aspirador de pó que atua em um ambiente com 4 quartos? Um quadrado (2×2) ?

Mas, antes de implementar usando uma estrutura similar a descrita acima, responda as questões abaixo:

- O que é relevante representar nos estados do mundo? Como os estados são estruturados (estrutura de dados) e qual o significado dela (dos campos)?
- Mostre como ficam representados os estados inicial e final segundo a representação adotada.
- Quais as operações sobre os estados? (detalhe como cada operação irá alterar os estados e quais as condições para cada operação ser executada)
- Qual a estimativa do tamanho do espaço de busca (número de estados possíveis)?

Entrega

A entrega deste exercício deve ser feita no Github Classroom: <https://classroom.github.com/a/A77CyNi2>. A entrega é individual e deve ser feita até o meio dia do dia 10/02. O projeto base existente no Github classroom tem um arquivo de teste. Você deve implementar o código de forma que todos os testes passem.

Exercício adicional: Aspirador de Pó com poltrona.

Se você já terminou o exercício acima então você pode implementar uma solução para esta situação adicional. Este exercício é opcional e não será considerado para a nota.

Neste problema os quartos da casa podem possuir poltronas. Para limpar o quarto da casa que tem poltrona o agente deve antes de executar a ação limpar **virar** a poltrona. Claro que depois de limpar o agente deve também **desvirar** a poltrona para deixar o quarto em ordem.

Para este problema considere um ambiente (casa) também com 4 quartos (um quadrado (2×2)).

Mas, antes de implementar usando uma estrutura similar a descrita acima, responda as questões abaixo:

- O que é relevante representar nos estados do mundo? Como os estados são estruturados (estrutura de dados) e qual o significado dela (dos campos)?
- Mostre como ficam representados os estados inicial e final segundo a representação adotada.
- Quais as operações sobre os estados? (detalhe como cada operação irá alterar os estados e quais as condições para cada operação ser executada)
- Qual a estimativa do tamanho do espaço de busca (número de estados possíveis)?

🕒 February 10, 2025

6.2 Busca em espaço de estados

6.2.1 Alterando os algoritmos de busca

Nas aulas anteriores implementamos algumas versões do problema do aspirador de pó. Para a atividade de hoje vamos considerar a implementação mais simples, que é a que considera um aspirador de pó em um ambiente com 2 quartos.

Completando o método `env`

Nesta implementação você vai adicionar o seguinte comportamento no método `env()`:

```
def env(self):
    return json.dumps(self.__dict__)
```

Alterando a chamada do método `search`

Na hora de chamar o método `search` altere o código de:

```
algorithm = BuscaLargura()
result = algorithm.search(state)
```

para:

```
algorithm = BuscaLargura()
result = algorithm.search(state, trace=True)
```

adicionando o parâmetro `trace=True`. Tente entender o que acontece: (i) qual o resultado encontrado, (ii) qual o conteúdo do log impresso.

Alterando o algoritmo de busca

Depois disto, altere o código de:

```
algorithm = BuscaLargura()
result = algorithm.search(state, trace=True)
```

para:

```
algorithm = BuscaProfundidade()
result = algorithm.search(state, trace=True, m=10)
```

Tente entender o que acontece: (i) qual o resultado encontrado, (ii) qual o conteúdo do log impresso. Por que os resultados são diferentes?

Objetivo desta aula

O objetivo desta aula é introduzir o conceito de algoritmo de busca, mais especificamente, usando os algoritmo de busca em largura e o algoritmo de busca em profundidade.

Material de referência

🕒 February 11, 2025

6.2.2 Algoritmos de Busca em Largura, Profundidade e Profundidade Iterativa

Nas aulas anteriores implementamos um agente aspirador de pó. Na última aula executamos os algoritmos de Busca em Largura e Profundidade e discutimos alguns conceitos relacionados a esses algoritmos. Nesta aula vamos explicar com mais detalhes e de forma mais estruturada como esses algoritmos funcionam.

Objetivo desta aula

O objetivo desta aula é entender como os algoritmos de busca em largura, profundidade e profundidade iterativa funcionam.

Ao final desta aula...

Ao final desta aula você deverá saber a diferença entre os algoritmos de busca:

- em largura;
- em profundidade, e;
- em profundidade iterativa.

Além disso...

Além disso você também deverá saber como avaliar um algoritmo de busca, quais indicadores utilizar, e, consequentemente, saber como aplicá-los nos problemas apresentados.

Material utilizado

O material utilizado para esta aula está nos slides 17 até 28 do conjunto de slides abaixo:

Atividade de laboratório

Um robô elevador está programado para transportar cargas dentro de um armazém vertical. Ele começa no térreo (andar 0) e pode realizar duas operações para se mover entre os andares:

- Subir 1 andar: o elevador move-se do andar $(A = A+1)$.
- Subir 2 andares: o elevador move-se do andar $(A = A+2)$.

O objetivo do robô é alcançar um andar específico $(A_{\{f\}})$, definido pelo usuário.

Implemente um agente chamado `Elevador` que resolve este problema utilizando os algoritmos de busca em largura, busca em profundidade e busca em profundidade iterativa.

Este robô sabe executar duas ações: "subir 1 andar" e "subir 2 andares". Na sua implementação você deve usar exatamente este nome para as ações. No método `successors` você deve primeiro adicionar a ação "subir 2 andares" e depois a ação "subir 1 andar".

Este exercício é composto por algumas partes: implementando o agente com os algoritmos e avaliando o comportamento do agente.

IMPLEMENTANDO O AGENTE

Você deve implementar um arquivo `Elevador.py` com uma função `main(inicio, objetivo, algoritmo)` que recebe 3 parâmetros: o andar de início, o andar objetivo e o algoritmo que será utilizado. O parâmetro `algoritmo` aceita três valores: BL, BP, BPI. Estes valores representam os algoritmos de busca em largura, busca em profundidade e busca em profundidade iterativa, respectivamente. A sua implementação deve passar por todos os testes do arquivo `test_elevador.py`.

Esta primeira parte é pré-requisito da segunda parte. Se na entrega esta parte não estiver implementada, a segunda parte não será avaliada.

AVALIANDO O COMPORTAMENTO DO AGENTE

Utilize este código para testar os três algoritmos vistos até o momento (busca em largura, busca em profundidade e busca em profundidade iterativo) variando o estado final de (1) até (50) . No caso do algoritmo em profundidade, teste duas versões ($m = 10$) e ($m = 100$)).

Armazene o tempo de processamento criando uma tabela similar a esta:

Algoritmo	Objetivo	Tempo de processamento em segundos
Busca em Largura	1	0.000036
Busca em Largura	\dots	\dots
Busca em Largura	10	0.000378
Busca em Largura	50	\dots
Busca em Profundidade com $(m = 10)$	1	0.000033
Busca em Profundidade com $(m = 10)$	\dots	\dots
Busca em Profundidade com $(m = 100)$	\dots	\dots
Busca em Profundidade Iterativa	1	\dots
Busca em Profundidade Iterativa	\dots	\dots
Busca em Profundidade Iterativa	50	\dots

Em alguns casos a combinação do algoritmo com o objetivo não fornece um resultado. Você deve informar na tabela estes casos. Nestas situações você deve colocar o tempo de processamento como NaN, ou seja, valor faltante. De forma alguma você pode colocar o valor zero nestas situações. Utilize os conhecimentos adquiridos na disciplina de Ciência de Dados do semestre passado e faça um *plot* destes dados em um único gráfico.

Em um documento, coloque a tabela, o gráfico e responda as seguintes perguntas:


- Segundo o que discutimos em sala de aula, quais destes algoritmos são **ótimos**? Os resultados encontrados neste exercício são coerentes com esta informação? Justifique a sua resposta.
- Segundo o que discutimos em sala de aula, quais destes algoritmos são **completos**? Os resultados encontrados neste exercício são coerentes com esta informação? Justifique a sua resposta.
- Teve algum algoritmo que travou por falta de memória no seu computador? Se sim, qual é a explicação?

ENTREGA

Esta atividade é individual e deve ser submetida via Github Classroom. O link para a atividade é <https://classroom.github.com/a/3HHDtNJo>. O prazo para entrega é **24/02/2025**.

RUBRICA DE AVALIAÇÃO

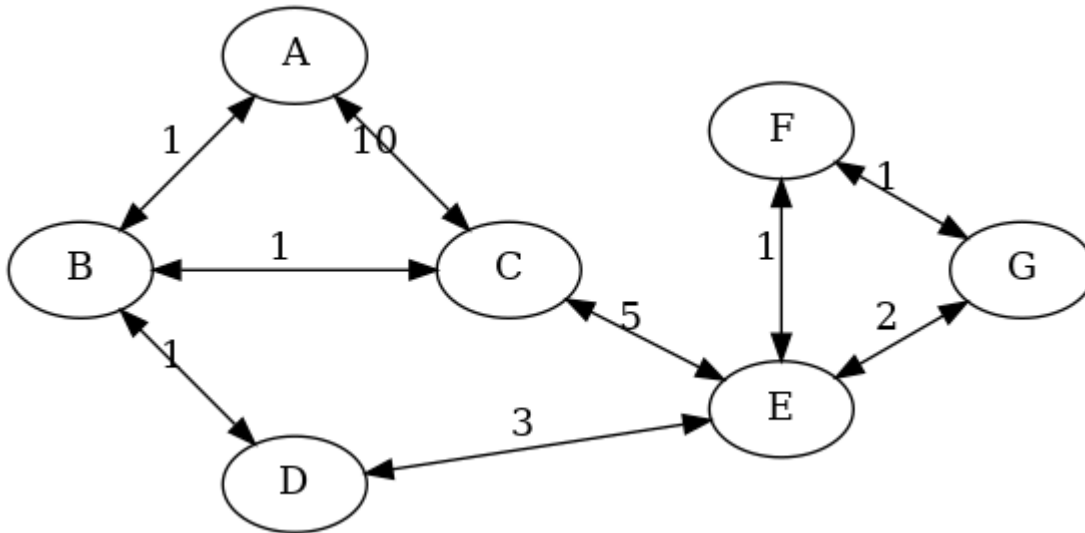
Conceito	Descrição
A+	Submeteu um documento que apresenta a tabela completa, responde todas as perguntas de forma correta e apresenta um único <i>plot</i> que sumariza todos os dados de forma correta e objetiva.
A	Submeteu um documento que apresenta a tabela completa, responde todas as perguntas de forma correta e apresenta o <i>plot</i> , mas este plot poderia ser melhor feito.
C	Submeteu um documento que apresenta a tabela, mas não responde todas as perguntas de forma correta ou não apresenta o <i>plot</i> .
D	A implementação passou por todos os testes.
I	A implementação não passou por todos os testes.

 February 17, 2025

6.2.3 Algoritmo de custo uniforme

Procurando caminhos em um mapa

Considere o mapa abaixo:



Os $\text{nodos} = \{A, B, C, D, E, F, G\}$ são locais. As arestas são as ligações entre os locais. Cada aresta tem um custo. Por exemplo, O caminho entre A e B tem custo 1. O caminho entre A e C tem custo 10. Este é um grafo bi-direcional. Ou seja, O custo de A para C é o mesmo de C para A.

Implemente uma solução que é capaz de encontrar o menor caminho entre qualquer par de locais do mapa.

Crie um repositório a partir do github classroom. O link é: <https://classroom.github.com/a/J4yPgJgt>. Neste repositório já existem alguns arquivos. Leia estes arquivos, inclusive o de teste, para entender o que é esperado.

Quando você estiver com o código pronto, faça o push para o repositório.

Referências

O material utilizado para esta aula está nos slides 30 até 32 do conjunto de slides abaixo:

🕒 February 25, 2025

6.2.4 Revisão sobre algoritmos de busca sistemáticos

Algoritmo de busca em Largura

```
function BL(Estado inicial): Nodo
  Fila abertos
  abertos.add(new Nodo(inicial))
  while abertos.size() > 0 do
    Nodo n  $\leftarrow$  abertos.removeFirst()
    if n.getEstado().éMeta() then
      return n
    end if
    abertos.append(n.sucessores())
  end while
  return null
```

Análise do algoritmo BL

- Completo: sim
- Ótimo: sim, para problemas de custo uniforme
- Tempo: explorar $O(b^d)$ nodos
 - ★ b = fator de ramificação
 - ★ d = profundidade do estado meta
 - ★ $O(b^d) = 1 + b + b^2 + b^3 + \dots + b^d$
- Espaço: guardar $O(b^d)$ nodos.

Algoritmo de busca em Profundidade

```
function BP(Estado inicial, int m): Nodo
  Pilha abertos
  abertos.add(new Nodo(inicial))
  while abertos.size() > 0 do
    Nodo n ← abertos.removeTopo()
    if n.getEstado().éMeta() then
      return n
    end if
    if n.getProfundidade() < m then
      abertos.insert(n.sucessores())
    end if
  end while
  return null
```

Análise do algoritmo BP

- Completo: não (caso a meta esteja em profundidade maior que m)

Se $m = \infty$, é completo se o espaço de estados é finito e existe poda para não haver loops entre as operações

- Ótimo: não
- Tempo: explorar $O(b^m)$ nodos (ruim se m é muito maior que d)
- Espaço: guardar $O(bm)$ nodos. (em profundidade 12, ocupa 12 Kbytes!)

Algoritmo de busca em Profundidade Iterativa

```
function BPI(Estado inicial): Nodo  
  int  $p \leftarrow 1$   
  loop  
    Nodo  $n \leftarrow \text{BP}(\textit{inicial}, p)$   
    if  $n \neq \text{null}$  then  
      return  $n$   
    end if  
     $p \leftarrow p + 1$   
  end loop
```

Análise do algoritmo BPI

- Completo: sim
- Ótimo: sim, se todas as ações tem o mesmo custo
- Tempo: explorar $O(b^d)$ nodos
- Espaço: guardar $O(bd)$ nodos.

Algoritmo de busca de Custo Uniforme

```
function BCU(Estado inicial): Nodo  
Set abertos ordenados por custo  
abertos.add(new Nodo(inicial))  
while abertos.size() > 0 do  
    Nodo n  $\leftarrow$  abertos.removeFirst()  
    if n.getEstado().éMeta() then  
        return n  
    end if  
    abertos.append(n.sucessores())  
end while  
return null
```

Algoritmo de busca de custo uniforme

- Expande nós de acordo com o custo.
- Se $\text{custo} = \text{profundidade do nó}$ então temos uma busca em largura.

Resumo

	BL	BP	BPI	BCU
Completo	sim	não	sim	sim
Ótimo	sim	não	sim	sim
Tempo	$O(b^d)$	$O(b^m)$	$O(b^d)$	$O(b^d)$
Espaço	$O(b^d)$	$O(bm)$	$O(bd)$	$O(b^d)$

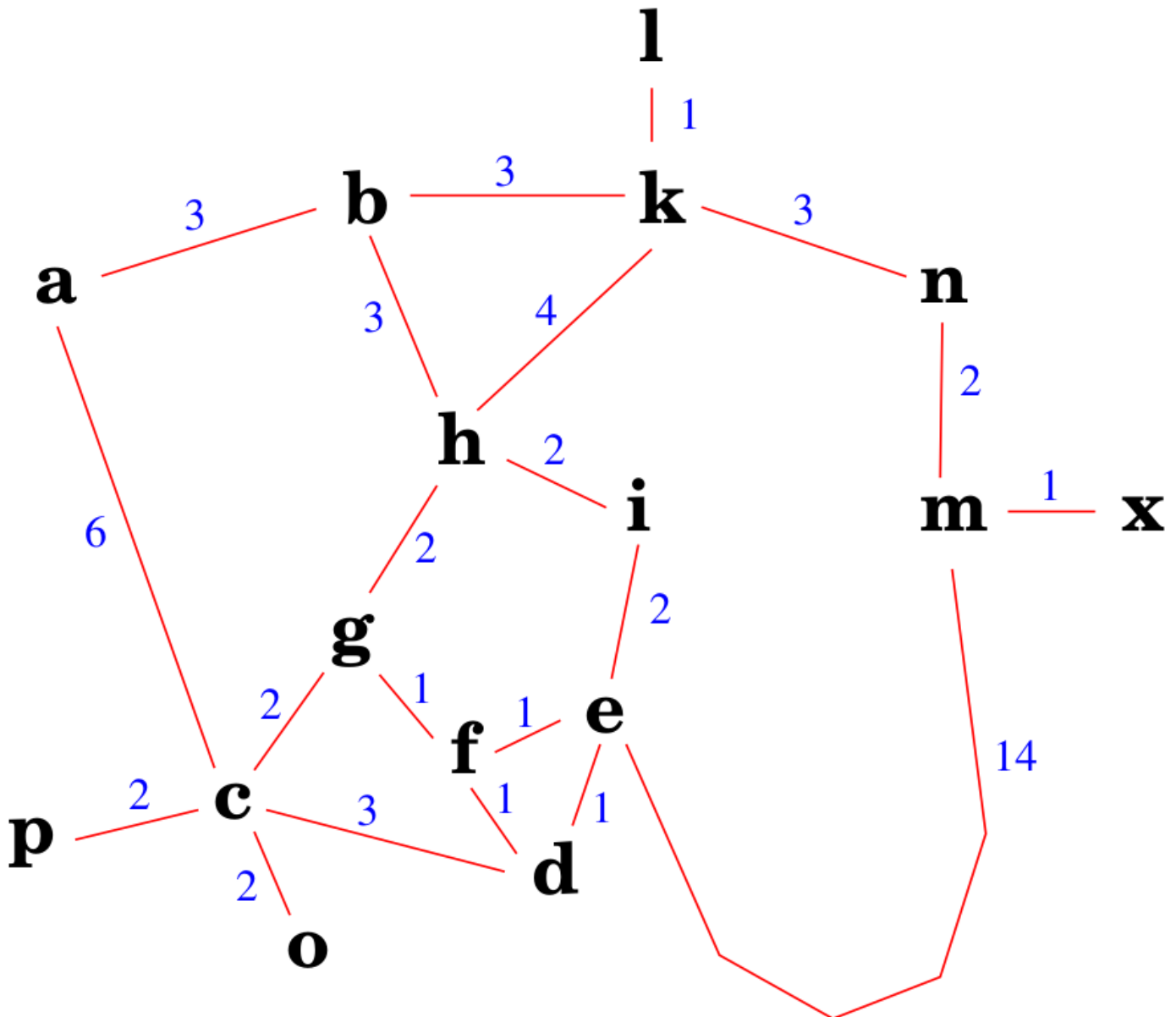
Referências

🕒 September 12, 2023

6.3 Busca com heurística

6.3.1 Menor caminho entre duas cidades

Considere o mapa abaixo:



Implemente um agente autônomo que consegue resolver este problema. Antes de iniciar a implementação, pense na resposta para as seguintes perguntas:

- O que é relevante representar nos estados do mundo? Como os estados são estruturados (estrutura de dados) e qual o significado dela (dos campos)?
- Mostre como ficam representados os estados inicial e final segundo a representação adotada.
- Quais as operações sobre os estados? (detalhe como cada operação irá alterar os estados e quais as condições para cada operação ser executada)
- Qual a estimativa do tamanho do espaço de busca (número de estados possíveis)?
- Que algoritmo de busca pode ser utilizado para resolver este problema considerando que a solução apresentada precisa ser ótima?

Sugestão de codificação do mapa

Considere a definição abaixo como mapa para a sua implementação:

```
@staticmethod
def createArea():

    Mapa = {
        'a': [(3, 'b'), (6, 'c')],
        'b': [(3, 'a'), (3, 'h'), (3, 'k')],
        'c': [(6, 'a'), (2, 'g'), (3, 'd'), (2, 'o'), (2, 'p')],
        'd': [(3, 'c'), (1, 'f'), (1, 'e')],
        'e': [(2, 'i'), (1, 'f'), (1, 'd'), (14, 'm')],
        'f': [(1, 'g'), (1, 'e'), (1, 'd')],
        'g': [(2, 'c'), (1, 'f'), (2, 'h')],
        'h': [(2, 'i'), (2, 'g'), (3, 'b'), (4, 'k')],
        'i': [(2, 'e'), (2, 'h')],
        'l': [(1, 'k')],
        'k': [(1, 'l'), (3, 'n'), (4, 'h'), (3, 'b')],
        'm': [(2, 'n'), (1, 'x'), (14, 'e')],
        'n': [(2, 'm'), (3, 'k')],
        'o': [(2, 'c')],
        'p': [(2, 'c')],
        'x': [(1, 'm')]
    }
```

Implementações

Implemente um agente, usando o algoritmo de busca em largura, para encontrar um caminho entre a cidade *i* e *o*.

Perguntas:

- Qual foi o tempo de processamento até a implementação encontrar uma solução?
- A árvore de busca gerada é "inteligente"?
- A solução encontrada é ótima?

Usando a mesma implementação, encontre um caminho entre a cidade *b* e *o*.

Perguntas:

- Qual foi o tempo de processamento até a implementação encontrar uma solução?
- A árvore de busca gerada é "inteligente"?
- A solução encontrada é ótima?

Usando o algoritmo de custo uniforme

Utilize o algoritmo de custo uniforme para encontrar uma solução para os problemas abaixo:

- da cidade *i* para a cidade *o*
- da cidade *b* para a cidade *o*
- da cidade *i* para a cidade *x*

Perguntas:

- Qual foi o tempo de processamento até a implementação encontrar uma solução?
- A árvore de busca gerada é "inteligente"?
- A solução encontrada é ótima?

Entendendo a busca que o algoritmo faz

Para entendermos melhor o que está acontecendo, segue a implementação da **Busca de Custo Uniforme**:

```
class BuscaCustoUniforme (SearchAlgorithm):

    def search (self, initialState, trace=False):
        open = []
        new_n = Node(initialState, None)
        open.append((new_n, new_n.g))
        while (len(open) > 0):
            #list sorted by g()
            open.sort(key = sortFunction, reverse = True)
            n = open.pop()[0]
            if trace: print(f'Estado = {n.state.env()} com custo = {n.g}')
            if (n.state.is_goal()):
                return n
            for i in n.state.sucessors():
                new_n = Node(i,n)
                open.append((new_n,new_n.g))
        return None
```

Ao executar:

```
state = Map('i', 0, 'i', 'x')
algorithm = BuscaCustoUniforme()
ts = time.time()
result = algorithm.search(state, trace=True)
tf = time.time()
```

O algoritmo abriu muitos nodos de forma desnecessária?

Usando heurísticas para otimizar a busca na árvore

O que é uma heurística?

O que é uma heurística?

Qual é a utilidade de uma heurística?

Qual é a utilidade de uma heurística?

Que heurística podemos usar no problema das cidades?

Que heurística podemos usar no problema das cidades?

Para responder as perguntas acima considere o slide 32 em diante:

 September 9, 2024

6.3.2 Continuação: menor caminho entre duas cidades

Usando busca em profundidade iterativo

Encontrar um caminho entre a cidade *i* e *o*.

Perguntas:

- Qual foi o tempo de processamento até a implementação encontrar uma solução?
- Quantos nodos o algoritmo de busca abriu até encontrar a solução?
- A solução encontrada é ótima?

Encontre um caminho entre a cidade *b* e *o*.

Perguntas:

- Qual foi o tempo de processamento até a implementação encontrar uma solução?
- Quantos nodos o algoritmo de busca abriu até encontrar a solução?
- A solução encontrada é ótima?

Usando busca de custo uniforme

Utilize o algoritmo de custo uniforme para encontrar uma solução para os problemas abaixo:

- da cidade *i* para a cidade *o*
- da cidade *b* para a cidade *o*
- da cidade *i* para a cidade *x*

Perguntas:

- Qual foi o tempo de processamento até a implementação encontrar uma solução?
- Quantos nodos o algoritmo de busca abriu até encontrar a solução?
- A solução encontrada é ótima?

Usando busca gananciosa

Considerar o slide 32 em diante:

- Que heurística podemos utilizar para este problema? Crie uma heurística e utilize o algoritmo `from aigyminsper.search.SearchAlgorithms import BuscaGananciosa` para encontrar as soluções para:
- da cidade *i* para a cidade *o*
- da cidade *b* para a cidade *o*
- da cidade *i* para a cidade *x*

O algoritmo de busca `BuscaGananciosa` precisa de um método que define a heurística. Este método na implementação do pacote `aigyminsper` é definido como:

```
def h(self):
    return <<valor numérico>>
```

Para esta parte do exercício considere também o arquivo CSV com a definição das heurísticas: [MapHeurísticas.csv](https://insper.github.io/ia_bcc/).

Perguntas:

- Qual foi o tempo de processamento até a implementação encontrar uma solução?
- Quantos nodos o algoritmo de busca abriu até encontrar a solução?
- A solução encontrada é ótima?

Usando busca A*

E o algoritmo A-Estrela?

Comparando algoritmos

Execute todos os objetivos listados abaixo para os algoritmos de busca também listados na tabela abaixo.

Objetivo	Algoritmo	Solução	Tempo de processamento
$i \rightarrow o$	Custo Uniforme		
$b \rightarrow o$	Custo Uniforme		
$i \rightarrow x$	Custo Uniforme		
$i \rightarrow o$	Ganancioso		
$b \rightarrow o$	Ganancioso		
$i \rightarrow x$	Ganancioso		
$i \rightarrow o$	A-Estrela		
$b \rightarrow o$	A-Estrela		
$i \rightarrow x$	A-Estrela		
$i \rightarrow o$	A-Estrela ($h(N) = h(N) * 100$)		
$b \rightarrow o$	A-Estrela ($h(N) = h(N) * 100$)		
$i \rightarrow x$	A-Estrela ($h(N) = h(N) * 100$)		
$i \rightarrow o$	A-Estrela ($h(N) == 1$)		
$b \rightarrow o$	A-Estrela ($h(N) == 1$)		
$i \rightarrow x$	A-Estrela ($h(N) == 1$)		

Anote na tabela acima o tempo de processamento e a solução encontrada e discuta os resultados obtidos:

- Qual foi o tempo de processamento até a implementação encontrar uma solução?
- Por que o tempo de processamento foi diferente para cada algoritmo?
- Por que a solução encontrada foi diferente em cada algoritmo?
- Por que a solução encontrada foi diferente em cada versão do A-Estrela?

🕒 March 9, 2025

6.3.3 Heurística admissível

No problema apresentado na [aula passada](#) teve alguma situação onde o **algoritmo A*** não encontrou a solução ótima?

Os arquivos: [src/MapLivro.py](#) e [data/MapHeuristics.csv](#) fazem parte de uma possível solução para o problema.

Resposta

Sim, quando a heurística não é admissível.

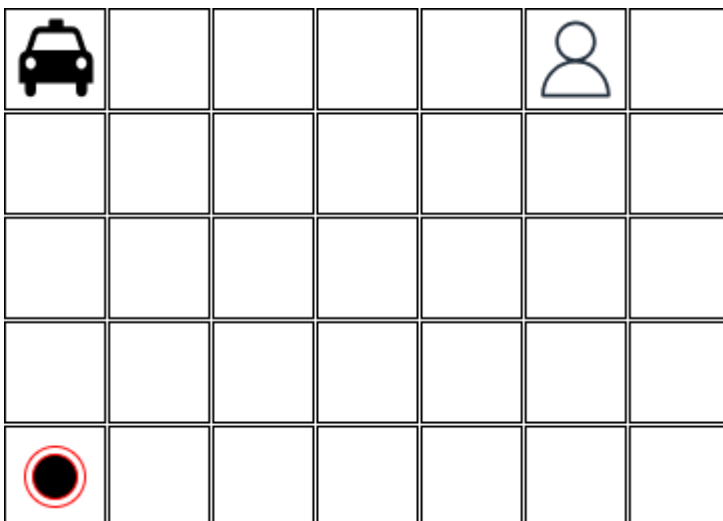
Dica

A heurística é admissível quando ela nunca superestima o custo para alcançar o objetivo.

Problemas

PROBLEMA DO TAXI DRIVER

Considere um agente que precisa levar um passageiro de um ponto A para um ponto B como apresentado no mapa abaixo:



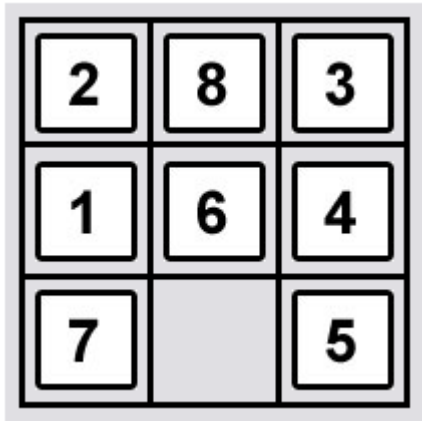
Neste mapa o taxi está na posição [0,0], o passageiro está na posição [0,5] e o destino do passageiro está na posição [4,0]. O agente pode se mover em qualquer direção (esquerda, direita, cima, baixo), mas apenas uma casa por vez. O custo para se mover de uma casa para outra é 1. O agente também tem como ação pegar e deixar o passageiro. O custo para pegar o passageiro é 2 e o custo para deixar o passageiro é 0. O objetivo do agente é levar o passageiro até o destino com o menor custo possível.

Para as dimensões apresentadas acima talvez algoritmos de busca cegos poderiam ser utilizados. No entanto, considerando um mapa maior algoritmos de busca cegos não seriam uma boa opção.

O que é necessário fazer para implementar uma solução que sempre fornece uma solução ótima para o problema? Independente das dimensões do mapa?

8-PUZZLE

Defina uma ou mais heurísticas admissíveis para o problema do [8-puzzle](#).



Esta mesma heurística pode ser utilizada para o problema do [15-puzzle](#)?

O que muda do problema do 8-puzzle para o 15-puzzle?

CAVALO E TABULEIRO DE XADREZ

Considerando um tabuleiro de xadrez (8×8) com um único cavalo, quais os movimentos que o cavalo deve fazer para percorrer todas as posições do tabuleiro uma única vez e retornar ao ponto de partida?

🕒 March 11, 2025

6.3.4 Problema do 8-Puzzle

Considere o problema dos 8-puzzle discutido em sala de aula. Implemente uma solução para este problema.

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

Implemente um agente autônomo que consegue resolver este problema. Você deve adotar uma matriz como forma de representação dos estados. Por exemplo, o estado inicial indicado acima deve ser representado da seguinte forma:

```
[[5, 4, 0], [6, 1, 8], [7, 3, 2]]
```

ou seja, o espaço em branco deve ser representado por um zero. O estado final apresentado acima deve ser representado da seguinte forma:

```
[[1, 2, 3], [8, 0, 4], [7, 6, 5]]
```

As operações que o agente sabe fazer são:

- **cima**: o agente move o espaço em branco para cima;
- **baixo**: o agente move o espaço em branco para baixo;
- **esquerda**: o agente move o espaço em branco para a esquerda, e;
- **direita**: o agente move o espaço em branco para a direita.

Considerando a forma como os estados são representados e as ações que o agente sabe executar, responda as seguintes perguntas:

- Qual a estimativa do tamanho do espaço de busca (número de estados possíveis)?
- Que algoritmo de busca pode ser utilizado para resolver este problema considerando que a solução apresentada precisa ser ótima?

A entrega da sua implementação deverá ter 1 arquivo chamado `Puzzle8.py` que implementa a interface `State` e que deve passar por TODOS os testes especificados em `test_8_puzzle.py`.

O seu agente deve ser capaz de identificar um plano para todos os estados iniciais descritos como fáceis e difíceis no arquivo de testes. Para os estados descritos como impossível o agente precisa retornar a mensagem *"Nao achou solucao"*. Deve-se considerar o estado *goal* em formato caracol, como apresentado abaixo:

1	2	3
8		4
7	6	5

Ao implementar o método `successors` do seu agente considere a seguinte ordem para adicionar os nodos em abertos: cima, baixo, esquerda e direita.

Formato de entrega

- Para a implementação e entrega deste exercício nós vamos utilizar o [Github Classroom](#).
- **Prazo para a entrega:** 23/03/2025 (domingo) até às 23:50 horas.
- Este trabalho é individual.

Exemplo de código para o método `show_path`

```
def show_path(self):
    algorithm = AEstrela()
    if not Puzzle8.tem_solucacao(self.tabuleiro):
        return 'Nao tem solucao'
    result = algorithm.search(self)
    if result != None:
        return result.show_path()
    else:
        return 'Nao achou solucao'
```

Detalhe importante sobre o método `tem_solucacao`: uma forma para cálculo se uma determinada configuração tem solução ou não é seguir a seguinte regra: *deve-se calcular a quantidade de inversões necessárias para ordenar certa sequência numérica, determinado por Possível a quantidade de inversões pares e Impossível a quantidade de inversões ímpares.*

 March 19, 2025

6.4 Programação por restrições

6.4.1 Atividade proposta

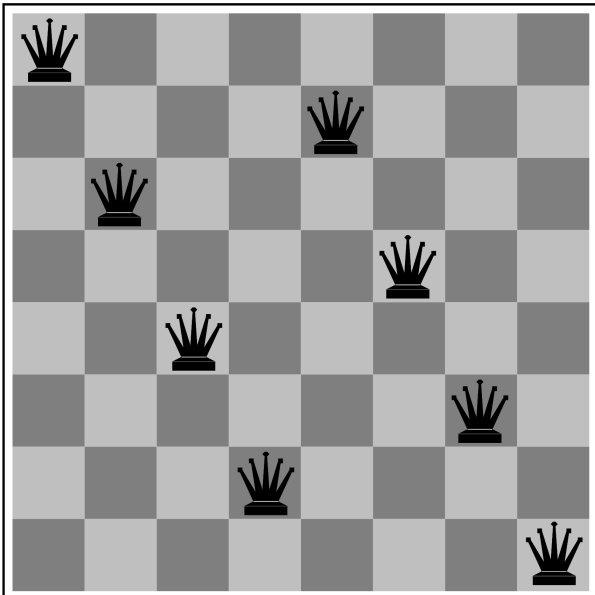
Contexto

Até agora vimos diversos problemas de planejamento. Problemas onde o agente precisava sair de um estado e chegar em outro, ou seja, sequenciar um conjunto de ações para alcançar um objetivo.

No entanto, existem outras classes de problemas onde o objetivo não está em encontrar uma sequência de ações, mas sim em encontrar um ou mais estados que satisfazem um conjunto de restrições.

Problema das N rainhas

Um exemplo de problema desta classe é o das **N** rainhas: dado um tabuleiro $(N \times N)$, é possível encontrar uma configuração onde (N) rainhas no tabuleiro não conseguem atacar nenhuma das outras rainhas no mesmo tabuleiro?



Para tratar este tipo de problema existe uma outra classe de algoritmos. Nesta disciplina, vamos ver os seguintes algoritmos desta classe:

- subida da montanha, e;
- subida da montanha estocástico.

Para entender o funcionamento dos algoritmos, você pode ler [este conjunto de slides](#).

Atividade de implementação

Para esta atividade vamos utilizar a biblioteca [aigyminsper](#). Mais especificamente, os algoritmos disponíveis em `aigyminsper.search.CSPAlgorithms` e a solução para o problema das **N** Rainhas em [N_QueensProblem.py](#).

- O arquivo `N_QueensProblem.py` está incompleto. Falta implementar a **heurística** para o problema. Sendo assim, a sua primeira atividade é fazer isto.
- Depois de implementada a heurística, teste a solução com o algoritmo **SubidaMontanha** para todos os cenários, onde $(4 \leq N \leq 10)$. O algoritmo subida da montanha conseguiu encontrar solução para todos os casos?
- Teste a solução com o algoritmo **SubidaMontanhaEstocastico** também para todos os cenários. Qual foi a diferença no comportamento?

Discussão e fechamento da aula

- Você sabe explicar o motivo das respostas fornecidas pelos algoritmos **SubidaMontanha** e **SubidaMontanhaEstocastico**?
- Detalhes de implementação:
- Qual o motivo das ações do agente serem só *move up* e *move down*?



Mínimo ou máximo local

O que é mínimo ou máximo local e quais são as suas consequências?

🕒 October 11, 2023

6.5 Ambientes competitivos

6.5.1 Jogos de tabuleiro e busca competitiva

Como sugestão de atividade adicional, é proposto que você leia e execute os exercícios propostos em [Implementando um jogador para Liga4](#).

Este material apresenta conceitos e algoritmos relacionados com a implementação de agentes autônomos que atuam em ambiente competitivo de soma zero e sem variável aleatória.

Questionário

Depois de feito o tutorial acima, responda as seguintes perguntas:

1. O que é um ambiente competitivo de soma zero?
2. Qual o objetivo do algoritmo Min-Max? Em outras palavras, por que um agente autônomo que atua em um ambiente competitivo deve usar o algoritmo Min-Max?
3. O que é função de utilidade? Por que utilizar funções de utilidade?
4. Qual é a relação da profundidade da árvore de busca do Min-Max com o desempenho final do agente? Existe correlação? Justifique a sua resposta.

Objetivos deste tópico


Se ao final desta atividade você consegue:

- Entender e implementar o algoritmo Min-Max sem limite de profundidade;
- Alterar o algoritmo Min-Max para incluir um limite de profundidade, e;
- Entender o conceito de função de utilidade, sua aplicação e como usar em um ambiente competitivo.

Então você terá alcançado os principais objetivos deste tópico.

Referências

- [Slides utilizados em sala de aula](#)
- [Material adicional sobre busca competitiva](#)

 April 16, 2024

6.6 Aprendizagem por reforço

6.6.1 Algoritmo Q-Learning

Definições

Material utilizado para aula expositiva:

Implementação

Depois de uma breve explicação sobre definições e principais conceitos, nós vamos implementar o primeiro agente usando aprendizagem por reforço. Por favor, siga as instruções abaixo:

TRABALHE COM O ARQUIVO TAXIDRIVERGYM_INTRODUCTION.PY

- Leia a descrição do ambiente em https://gymnasium.farama.org/environments/toy_text/taxi/.
- Copie o arquivo `TaxiDriverGym_introduction.py`.
- Execute cada um dos comandos que estão no arquivo `TaxiDriverGym_introduction.py` em um interpretador python para entender o que é `environment`, `reward` e `action`. Além de entender detalhes do ambiente.
- Quantos espaços possíveis o ambiente Taxi-v3 possui?
- Quantas ações o agente que atua no ambiente Taxi-v3 possui?
- O que a variável `reward` retornada por `env.step(<number>)` significa?

ARQUIVO QLEARNING.PY

Este arquivo implementa o algoritmo **Q-Learning**. A classe implementada neste arquivo possui 4 métodos:

- `__init__` (construtor): que recebe todos os hiperparâmetros do algoritmo Q-Learning e inicializa a Q-table com base no número de ações e estados informados pelo parâmetro `env`. Alguns destes hiperparâmetros não serão utilizados neste momento, mas vamos mantê-los.
- `select_action`: dado um estado, este método seleciona uma ação que deve ser executada.
- `train`: método responsável por executar as simulações e popular a **Q-table**. Este método retorna uma q-table, mas também atualiza um arquivo CSV com os dados da q-table e uma imagem que é um plot da quantidade de ações executadas em cada episódio.
- `plotactions`: é um método que cria uma imagem com o plot da quantidade de ações executadas em cada episódio.

ATUALIZANDO OS VALORES DA Q-TABLE

Dentro do método `train` tem-se o seguinte trecho de código:

```
for i in range(1, self.episodes+1):
    state = self.env.reset()
    reward = 0
    done = False
    actions = 0

    while not done:
        action = self.select_action(state)
        next_state, reward, done, _ = self.env.step(action)

        # Adjust Q value for current state
        old_value = #pegar o valor na q-table para a combinacao action e state
        next_max = #pegar o valor maximo da q-table para o proximo estado
        new_value = #calcula o novo valor
        self.q_table[state, action] = new_value

        # atualiza para o novo estado
        state = next_state
```

que é responsável por executar N episódios e atualizar a variável `self.q_table`.

Este método é chamado pelo arquivo `TaxiDriverGym.py` nas linhas 11 e 12:

```
# only execute the following lines if you want to create a new q-table
qlearn = QLearning(env, alpha=0.1, gamma=0.6, epsilon=0.7, epsilon_min=0.05, epsilon_dec=0.99, episodes=100000)
q_table = qlearn.train('data/q-table-taxi-driver.csv', 'results/actions_taxidriver')
#q_table = loadtxt('data/q-table-taxi-driver.csv', delimiter=',')
```

Atividades:

- Complete o código do método `train` em `QLearning.py`.
- Execute o arquivo `TaxiDriverGym.py` com o comando:

```
python TaxiDriverGym.py
```

Lembre-se que nesta execução o programa irá criar toda a Q-table e armazenar no arquivo `data/q-table-taxi-driver.csv`. Depois de calcular os valores para a Q-table o programa irá resolver um dos possíveis cenários considerando um estado inicial qualquer. Além disso, o programa irá gerar um plot no diretório `results` que descreve a quantidade de ações executadas em cada época.

- Abra o arquivo `results/action_taxidriver.jpg` e faça uma análise do mesmo. O que este gráfico representa?
- Agora faça o algoritmo `TaxiDriverGym.py` ler a Q-table a partir do arquivo gerado anteriormente e veja qual é o comportamento. Execute diversas vezes.
- Qual é o comportamento do agente? Ele sempre consegue encontrar uma solução? As soluções parecem ser ótimas?

Considerando os valores informados nos parâmetros do método `train`, se a sua implementação do `q-learning` estiver correta então o agente `TaxiDriverGym.py` deve encontrar a solução para todos os casos apresentados. Se por algum motivo a sua solução não estiver convergindo então significa que tem algum *bug* na atualização da q-table.

Uma vez que você confirmou que a sua implementação não tem *bugs* então você pode ajustar alguns dos hiperparâmetros. Por exemplo, diminuindo a quantidade de episódios e analisando a Q-table gerada.

- O arquivo `results/action_taxidriver.jpg` é um plot da quantidade de episódios versus a quantidade de atividades. Teria alguma outra forma de visualizar a evolução do agente? E se usarmos `rewards` ao invés da quantidade de atividades? A visualização fica melhor?

Q-table pronta

Como resultado da etapa de treinamento, o agente irá ter uma *Q-table* que mostra a melhor ação que ele deve executar em cada estado. Para usar este dado é muito simples:

```
(state, _) = env.reset()
done = False

while not done:
    print(state)
    action = np.argmax(q_table[state])
    state, reward, done, truncated, info = env.step(action)
```

Neste momento não temos mais nenhuma ação aleatória. O agente sempre irá selecionar a ação com o valor mais alto para cada estado.

6.6.2 Hiperparâmetros no Q-Learning

Ainda considerando o exemplo a implementação do `TaxiDriver`, responda as perguntas abaixo.

Para responder as questões abaixo utilize as implementações do `TaxiDriverGym.py` e o arquivo `QLearning.py` que você implementou na atividade anterior.

Manipulando α e γ

- Se α for um valor muito próximo de zero? Explique o comportamento encontrado.
- Se γ for zero? Explique o comportamento encontrado.

Para fundamentar a sua resposta, use os plots gerados na pasta `results` depois do treinamento.

Considerando uma escolha de ação sempre aleatória

O que acontece se a escolha das ações em cada estado for sempre aleatória? Ou seja, se a função `select_action` ao invés de ser definida como abaixo:

```
def select_action(self, state):
    rv = random.uniform(0, 1)
    if rv < self.epsilon:
        return self.env.action_space.sample() # Explore action space
    return np.argmax(self.q_table[state]) # Exploit learned values
```

É definida assim:

```
def select_action(self, state):
    return self.env.action_space.sample() # Explore action space
```

Qual o comportamento do agente? **Novamente:** use os plots gerados na pasta `results` depois do treinamento para fundamentar a sua resposta.

Considerando um agente que nunca explora novas ações

O que acontece se a escolha das ações em cada estado for sempre buscando a melhor ação? Ou seja:

```
def select_action(self, state):
    return np.argmax(self.q_table[state]) # Exploit learned values
```

Sumarizando os resultados através de imagens

Como podemos sumarizar os diferentes resultados através de imagens?

Neste momento, você já deve ter percebido que uma ferramenta muito útil para visualizar e sumarizar o aprendizado do agente são gráficos que mostram a evolução de alguma métrica ao longo dos diversos episódios.

- Quais foram as métricas utilizadas no caso do `TaxiDriver`?
- Quais foram os hiperparâmetros utilizados?
- O aprendizado dos agentes implementados para este caso **convergem** rapidamente?
- O **desempenho** do agente se mantém ao longo dos episódios?

Atividade

Você deve entregar um arquivo `README.md` que responde as seguintes perguntas:

- Qual é o impacto de α ? Faça um gráfico com a curva de aprendizado do agente fixando γ e variando α . A sugestão é utilizar três (3) valores para α : um próximo de zero, um intermediário e um próximo de um (1). Utilize um único gráfico com as três curvas de aprendizado.
- Qual é o impacto de γ no aprendizado do agente no ambiente `TaxiDriver`? Faça um gráfico com a curva de aprendizado do agente fixando α e variando γ . A sugestão é utilizar três (3) valores para γ : um próximo de zero, um intermediário e um próximo de um (1). Utilize um único gráfico com as três curvas de aprendizado.
- Qual é o impacto da forma como a ação é escolhida durante o treinamento? Faça um gráfico com a curva de aprendizado do agente usando três curvas: (i) onde o agente apenas explora o ambiente, ou seja, seleciona de forma aleatória todas as ações durante o treinamento; (ii) onde o agente apenas se utiliza do seu conhecimento sobre o ambiente, ou seja, na maioria das vezes faz o *exploitation* ao invés do *exploration*. Para este caso, crie uma função de escolha da ação onde o agente tem probabilidade de 90% para escolher uma ação de acordo com a *q-table* e os outros 10% são aleatórios; (iii) onde o agente faz uso de uma função de *exploration and exploitation* com um decaimento do ϵ .

Apresente os 3 gráficos no arquivo de `README.md`. Submeta este arquivo no repositório do projeto junto com os outros artefatos (scripts, dados e figuras). Este projeto é em grupo com até 3 pessoas e deve ser submetido neste link: <https://classroom.github.com/a/4rdGfqhx>.

EXEMPLO DE GRÁFICO

Um exemplo de imagem ¹ que sumariza dados ou apresenta resultados de experimentos é apresentada abaixo:

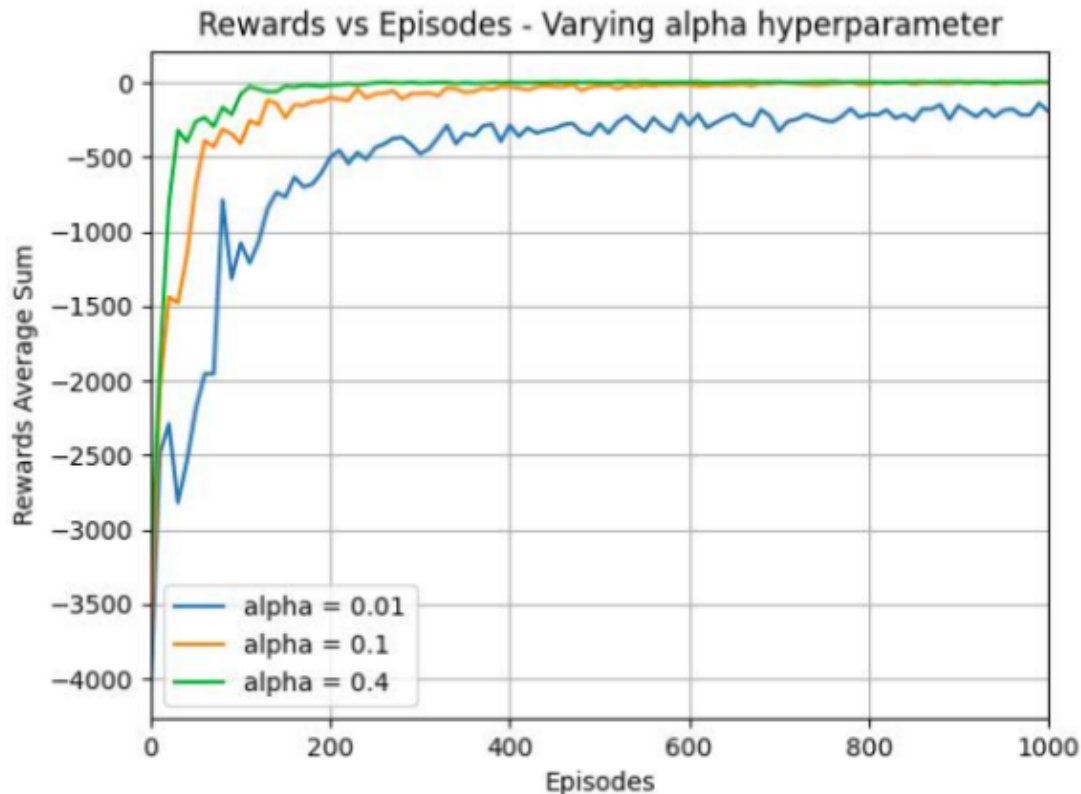
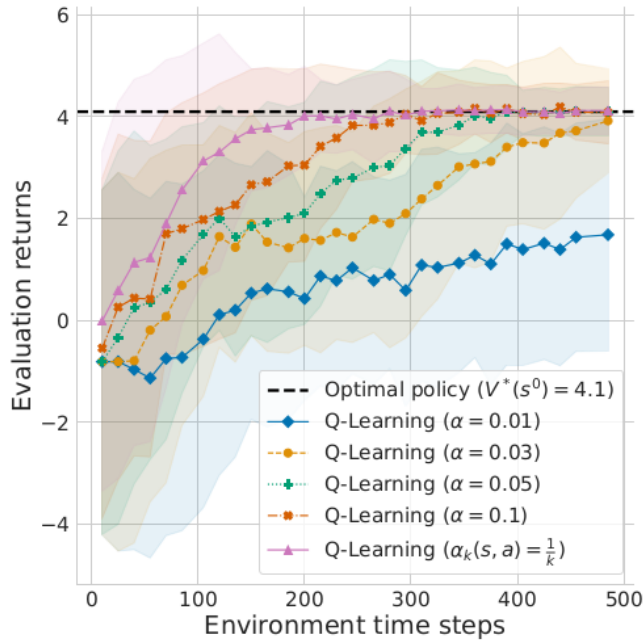


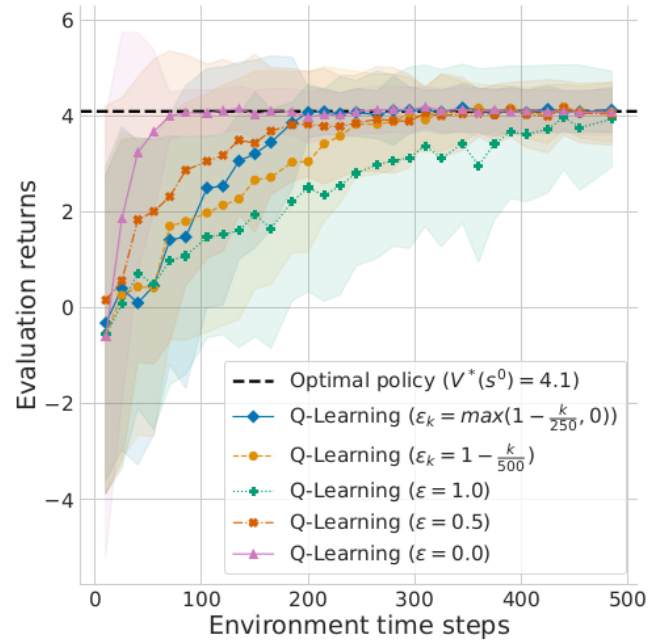
Figure 1: O parâmetro α representa a taxa de aprendizagem do agente, e seus valores variam em um intervalo de 0 a 1. De acordo com o algoritmo de treino do Q-learning, quando esse parâmetro assume valores mais próximos de 1, o novo aprendizado é um fator relevante durante o treino, já quando esse valor é próximo de 0 a importância do aprendizado se torna menos significativa. No gráfico, é possível observar o impacto da variação desse parâmetro ($\alpha = 0.01, 0.1$ e 0.4) durante os episódios, onde o treino com um menor α apresenta uma situação em que o algoritmo apresenta menos incentivo para aprender, o que gera uma curva mais lenta no aprendizado, custando mais para convergir que os valores mais altos de α . Nesse gráfico, os demais parâmetros do algoritmo, γ e ϵ , foram mantidos constantes, com valores iguais a 0.99 e 0.7, respectivamente.

Todas as informações relevantes para entender o resultado do treinamento precisam estar auto-contidas na imagem e na legenda da imagem.

Outro exemplo de gráfico muito bem feito é apresentado abaixo ²:



(c) Q-learning with different learning rates α (ϵ linearly decayed from 1 to 0)



(d) Q-learning with different exploration rates ϵ ($\alpha = 0.1$)

-
1. Este gráfico foi feito pela Letícia, aluna de Engenharia da Computação, durante a disciplina eletiva de *Reinforcement Learning*. ←
 2. Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. [Multi-Agent Reinforcement Learning: Foundations and Modern Approaches](#). MIT Press, 2024. ←

🕒 April 11, 2025

6.6.3 Algoritmo SARSA: abordagem on-policy

Nesta aula vamos utilizar os slides abaixo:

Definição e conceitos-chave

A atualização da *Q-table* no algoritmo **Q-Learning** é dada por:

$$Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_{A'} \{Q(s', A')\} - Q(s,a)]$$

A diferença entre a nova amostra e a estimativa antiga é usada para atualizar a estimativa antiga. O algoritmo **Q-Learning** considera o valor da nova amostra como o valor máximo possível de Q no estado s' :

$$\max_{A'} \{Q(s', A')\}$$

No entanto, a ação com o valor máximo possível de Q pode não ser a ação real que o agente tomará no futuro, porque com probabilidade ϵ o agente poderá executar uma ação aleatória. Em outras palavras, a ação usada para atualizar a política em **Q-Learning** é diferente da ação real que o agente tomará. Esta é a razão pela qual **Q-Learning** é chamado de algoritmo **off-policy**.

Por outro lado, o algoritmo **SARSA** é **on-policy** porque ele atualiza a *Q-table* com:

$$Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma Q(s', a) - Q(s,a)]$$

esta equação atualiza $Q(s,a)$ considerando a ação real executada pelo agente.

O algoritmo Sarsa é muito semelhante ao algoritmo Q-Learning:

```

function Sarsa(env,  $\alpha$ ,  $\gamma$ ,  $\epsilon$ ,  $\epsilon_{min}$ ,  $\epsilon_{dec}$ , episódios)
  inicializar os valores de  $Q(s, a)$  arbitrariamente
  for todos os episódios do
    inicializar s a partir de env
     $a \leftarrow \text{escolha}(s, \epsilon)$ 
    repeat
       $s', r \leftarrow \text{executar a ação } a \text{ no } env$ 
       $a' \leftarrow \text{escolha}(s', \epsilon)$ 
       $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$ 
       $s \leftarrow s'$ 
       $a \leftarrow a'$ 
    until s ser um estado final
    if  $\epsilon > \epsilon_{min}$  then  $\epsilon \leftarrow \epsilon \times \epsilon_{dec}$ 
  end for
  return Q

```

Implementação

O objetivo desta atividade é validar o entendimento do algoritmo **Sarsa** e compreender as diferenças práticas entre **Sarsa** e **Q-Learning**.

Siga os passos a seguir:

- **implementar o algoritmo Sarsa:** você pode começar a partir da sua implementação do Q-Learning, fazer uma cópia, mudar o nome 😊, e mudar o que for necessário.
- **aplicar a implementação do Sarsa para resolver o problema do taxi-driver:** compare os resultados do Sarsa com os resultados do Q-Learning. Compare a curva de treinamento e o comportamento do agente após o treinamento. Para facilitar esta comparação, você pode usar o mesmo valor de ϵ , α e γ para ambos os algoritmos. Sugere-se a seguinte configuração:

```

(
  alpha=0.1,
  gamma=0.99,
  epsilon=0.1,
  epsilon_min=0.1,
  epsilon_dec=1,
  episodes=5000
)

```

Sugere-se também fazer uma validação mais robusta após o treinamento, executando o agente treinado em 100 episódios e calculando a recompensa média. Como apresentado no código abaixo:

```
q_table = loadtxt('data/q-table-sarsa-cliff-walking.csv', delimiter=',')
#q_table = loadtxt('data/q-table-qlearning-cliff-walking.csv', delimiter=',')

list_actions = []
list_rewards = []

for i in range(100):
    (state, _) = env.reset()
    rewards = 0
    actions = 0
    done = False

    while not done:
        print(state)
        action = np.argmax(q_table[state])
        state, reward, done, truncated, info = env.step(action)

        rewards = rewards + reward
        actions = actions + 1

    list_actions.append(actions)
    list_rewards.append(rewards)

print("Average actions taken: {}".format(np.mean(list_actions)))
print("Standard deviation actions taken: {}".format(np.std(list_actions)))
print("Average rewards: {}".format(np.mean(list_rewards)))
print("Standard deviation rewards: {}".format(np.std(list_rewards)))
```

- a terceira atividade desta implementação é **aplicar os algoritmos Q-Learning e Sarsa em um ambiente diferente**: o [Cliff Walking](#). Este é um ambiente muito simples. No entanto, quando você aplica esses algoritmos nesse tipo de ambiente, você pode ver as diferenças entre esses algoritmos e identificar qualquer bug em sua implementação, se existir.

Para treinar seu agente para o problema do **Cliff Walking**, você deve configurar o ambiente assim:

```
env = gym.make("CliffWalking-v0").env
```

Depois da etapa de treinamento, para ver o comportamento, você deve codificar algo assim:

```
env = gym.make("CliffWalking-v0", render_mode="human").env

(state, _) = env.reset()
rewards = 0
actions = 0
done = False

while not done:
    print(state)
    action = np.argmax(q_table[state])
    state, reward, done, truncated, info = env.step(action)

    rewards = rewards + reward
    actions = actions + 1

print("\n")
print("Actions taken: {}".format(actions))
print("Rewards: {}".format(rewards))
```

Configure o `render_mode` para `human` para ver a animação do agente.

ENTREGA

Crie um arquivo `README.md` e responda as seguintes perguntas:

1. Qual algoritmo tem os melhores resultados para o ambiente do taxi-driver? A curva de aprendizado dos dois algoritmos é a mesma? O comportamento final do agente, depois de treinado, é ótimo em ambos os casos?
2. Qual algoritmo tem os melhores resultados para o ambiente do Cliff Walking? A curva de aprendizado dos dois algoritmos é a mesma? O comportamento final do agente, depois de treinado, é ótimo? Qual agente tem um comportamento mais conservador e qual tem um comportamento mais otimista?
3. De uma forma geral, qual seria a diferença entre os algoritmos **Q-Learning** e **Sarsa**? Os agentes treinados teriam o mesmo comportamento? As curvas de aprendizado também?


Rubrica de avaliação

Conceito	Descrição
A+	Responderam todas as perguntas e utilizaram as curvas de aprendizado e ilustrações do comportamento dos agentes treinados como base para as respostas.
B	Responderam todas as perguntas e utilizaram as curvas de aprendizado como base para as respostas.
C	A implementação dos algoritmos está correta, mas o estudante não entregou o arquivo README.md com as respostas.
D	A implementação dos algoritmos <i>Q-Learning</i> e <i>Sarsa</i> foi parcial.

Entrega

Esta atividade pode ser feita em grupos com até 2 integrantes.

Coloque todos os arquivos em um mesmo projeto e submeta-os para o <https://classroom.github.com/a/9Jt6D8XT>. Esta atividade é individual e o **prazo é 22/04/2025 23:30**.

 April 16, 2025

6.6.4 Ambientes não determinísticos

O ambiente **Frozen Lake** é um ambiente não determinístico onde um agente deve encontrar um caminho do lugar onde ele está para outro lugar passando por buracos. Se ele chegar no objetivo sem cair no buraco então ele termina a tarefa e tem 1 ponto de reward. Se ele cair em um dos buracos então ele termina a tarefa com 0 pontos de reward. Cada ação que não leva para um estado terminal tem reward igual a 0.

Neste ambiente o agente sabe executar 4 ações: ir para cima, ir para baixo, ir para esquerda e ir para direita. **Como o chão é de gelo, não necessariamente a ação de ir para baixo vai levar o agente para baixo**, por exemplo. Isto acontece com todas as quatro ações. Por isso que este ambiente é não determinístico.

Trabalhe com o arquivo `FrozenLake_introduction.py`

1. Este arquivo está disponível [aqui](#).
2. Leia a documentação do código fonte disponível em https://gymnasium.farama.org/environments/toy_text/frozen_lake/
3. Veja o que está codificado no arquivo `FrozenLake_introduction.py` e execute o mesmo.
4. Quantos estados e quantas ações o ambiente FrozenLake-v1 tem?
5. O que aconteceu com a execução das ações? O resultado foi o esperado? Descreva o que aconteceu.

Trabalhe com o arquivo `FrozenLakeQLearning.py`

- Este arquivo também está [aqui](#).
- Abra em um editor de texto e descomente as linhas 12 e 13 e comente a linha 14. O código deve ficar como abaixo:

```
# only execute the following lines if you want to create a new q-table
qlearn = QLearning(env, alpha=0.9, gamma=0.95, epsilon=0.8, epsilon_min=0.0001, epsilon_dec=0.9999, episodes=500000)
q_table = qlearn.train('data/q-table-frozen-lake.csv', 'results/actions_frozen_lake')
#q_table = loadtxt('data/q-table-frozen-lake.csv', delimiter=',')
```

- Execute o arquivo `FrozenLakeQLearning.py` com o comando:

```
python FrozenLakeQLearning.py
```

- Agora faça o algoritmo `FrozenLakeQLearning.py` ler a Q-table a partir do arquivo gerado anteriormente e veja qual é o comportamento. Execute diversas vezes. Ele consegue chegar ao objetivo sempre? Ele consegue chegar ao objetivo na maioria das vezes?
- E se executarmos 100 vezes? Quantas vezes o agente consegue atingir o objetivo?
- Como podemos melhorar o desempenho deste agente? Teste diferentes configurações de hiperparâmetros. Qual é o comportamento visto no gráfico de episódios versus rewards?

E o algoritmo Sarsa?

- Será que o algoritmo Sarsa tem um desempenho melhor para problemas não-determinísticos?
- [Aqui](#) tem um arquivo chamado `FrozenLakeSarsa.py` que você pode utilizar para responder esta pergunta.

Outro mapa

Existem dois mapas pré-configurados em https://gymnasium.farama.org/environments/toy_text/frozen_lake/. O mapa 4x4 e um mapa 8x8. E se mudarmos o mapa para 8x8?


```
import gymnasium as gym
env = gym.make("FrozenLake-v1", map_name="8x8", is_slippery=True).env
```

- O que muda? O problema se torna mais complexo? É necessário mudar algum dos hiperparâmetros? Qual é o melhor algoritmo? *Sarsa* ou *Q-Learning*?

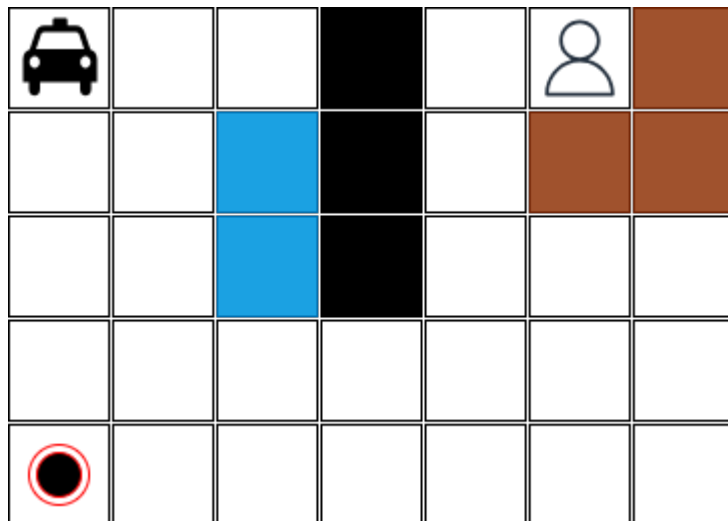
🕒 April 22, 2025

7. Projetos atuais

7.1 Taxi Driver

Neste exercício você deverá implementar um agente *taxi driver*. Um agente que é capaz de definir uma sequência de ações para pegar um passageiro em uma posição e deixá-lo em outra posição no mesmo mapa - um mapa que possui obstáculos, rios, asfalto e terra.

Por exemplo, considere o seguinte mapa:



Neste mapa:

- o táxi está na posição [0,0],
- o passageiro está na posição [0,5],
- o passageiro precisa ser levado para a posição [4,0],
- o mapa tem 5 linhas e 7 colunas,
- existem obstáculos no mapa que estão pintados em preto,
- existem rios no mapa que estão pintados em azul,
- existem áreas de terra que estão pintadas em marrom, e
- existem áreas de asfalto que estão pintadas em branco.

Sabemos que o táxi sabe executar as seguintes ações:

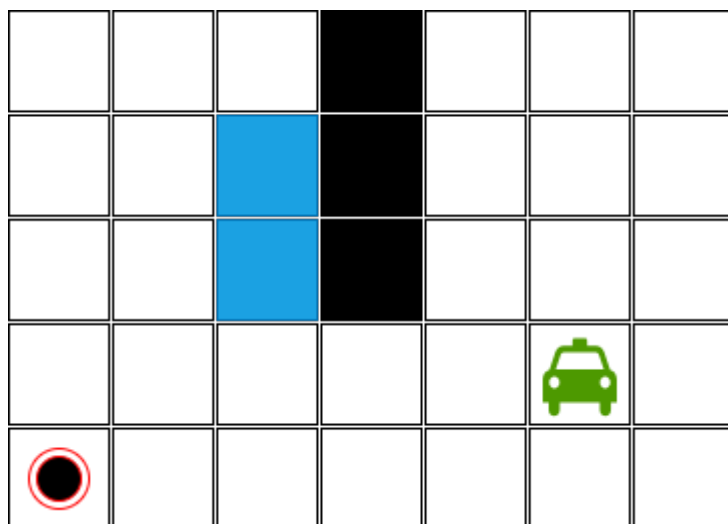
- ir para baixo;
- ir para cima;
- ir para esquerda;
- ir para direita;
- ir para a diagonal inferior esquerda;
- ir para a diagonal inferior direita;
- ir para a diagonal superior esquerda;
- ir para a diagonal superior direita;
- pegar o passageiro, e;
- liberar o passageiro.

Aspectos importantes sobre algumas ações:

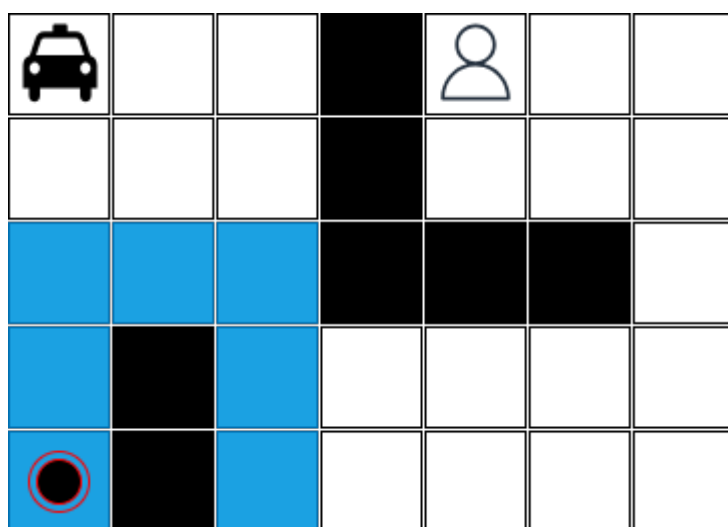
- a ação *pegar o passageiro* só pode ser executada se o táxi estiver vazio. Esta ação tem um custo de 5 unidades monetárias.
- a ação *pegar o passageiro* pode ser executada em qualquer posição do mapa. No entanto, só vai ter o efeito desejado se o táxi estiver na mesma posição que o passageiro.
- a ação *liberar o passageiro* só pode ser executada se o táxi estiver com o passageiro. Esta ação também tem custo de 5 unidades monetárias.
- a ação *liberar o passageiro* pode ser executada em qualquer posição do mapa. No entanto, só vai ter o efeito desejado se o táxi estiver na posição onde o passageiro precisa chegar.
- as ações *ir para baixo*, *ir para cima*, *ir para esquerda*, *ir para direita* e *ir para as diagonais* tem o mesmo custo e este custo é de 1 unidade monetária se o táxi estiver no asfalto.
- o táxi é anfíbio, ou seja, consegue andar na água, mas o custo de andar na água é de 10 unidades monetárias.
- o táxi consegue andar na terra, mas o custo de andar na terra é de 2 unidades monetárias.

Qual é a sequência de ações que o táxi precisa executar para pegar o passageiro e levar até o destino? Esta sequência de ações precisa ser ótima, ou seja, ter o menor custo.

A solução implementada precisa ser capaz de tratar diversas configurações com diversas dimensões. Por exemplo, a figura abaixo ilustra uma configuração possível onde o táxi está pintado de verde porque ele está com o passageiro:



A imagem abaixo é um mapa com as mesmas dimensões, mas com um número maior de obstáculos:



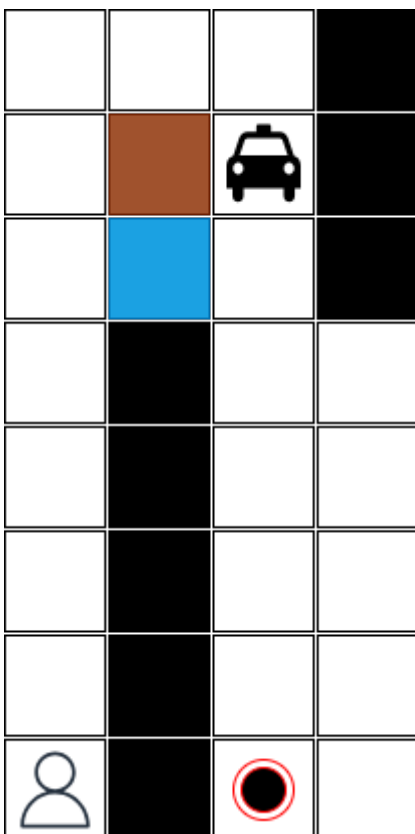
7.1.1 Entrega do exercício

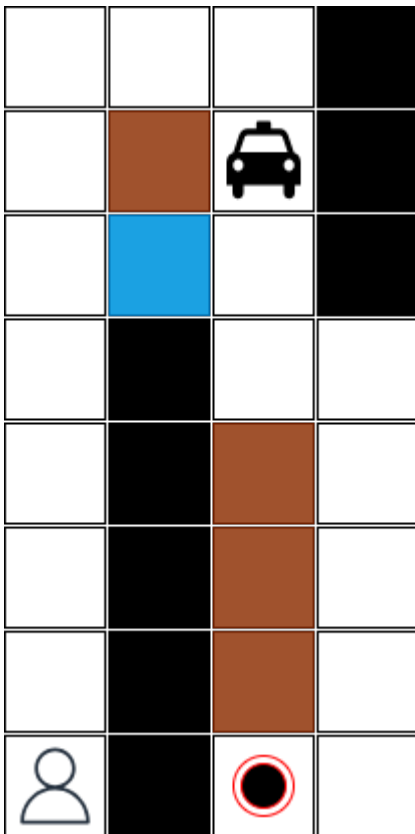
- Este exercício deverá ser feito por equipes com até 3 integrantes.
- O prazo máximo para entrega é 11/04/2023 (sexta-feira) até às 22:00 horas.
- A entrega deverá ser feita via *Github Classroom*. O link para a entrega é <https://classroom.github.com/a/I-L4Frwr>.
- Cada equipe deverá entregar a implementação de um *taxi driver* e um arquivo de documentação (`taxi_driver_readme.md`).
- A implementação deve ter um arquivo de testes usando `pytest` .
- O projeto deve ter um arquivo de `requirements.txt` que descreve os pacotes necessário para a execução.
- Não teremos tempo em sala de aula para discutir o projeto. Portanto, é importante que a equipe se organize para discutir o projeto fora do horário de aula.
- O horário de atendimento pode ser utilizado para tirar dúvidas sobre o projeto.

7.1.2 Requisitos da implementação do taxi driver

A equipe deverá implementar no mínimo dois arquivos: um arquivo python com a lógica do agente *taxi driver* e um arquivo de testes usando `pytest` . Qualquer entrega que não tenha os dois arquivos, o arquivo da solução e o arquivo de testes, terá nota **I**.

- A configuração do mapa deve ser fornecida via arquivo texto. Deve ser um dos parâmetros da implementação. A posição do passageiro, o destino final e a posição do táxi também podem ser fornecidas via arquivo texto ou via parâmetro de chamada da aplicação - este aspecto fica à critério da equipe.
- O arquivo de teste deve considerar os cenários ilustrados acima, mais os cenários ilustrados abaixo para que a equipe consiga um **C** como nota:

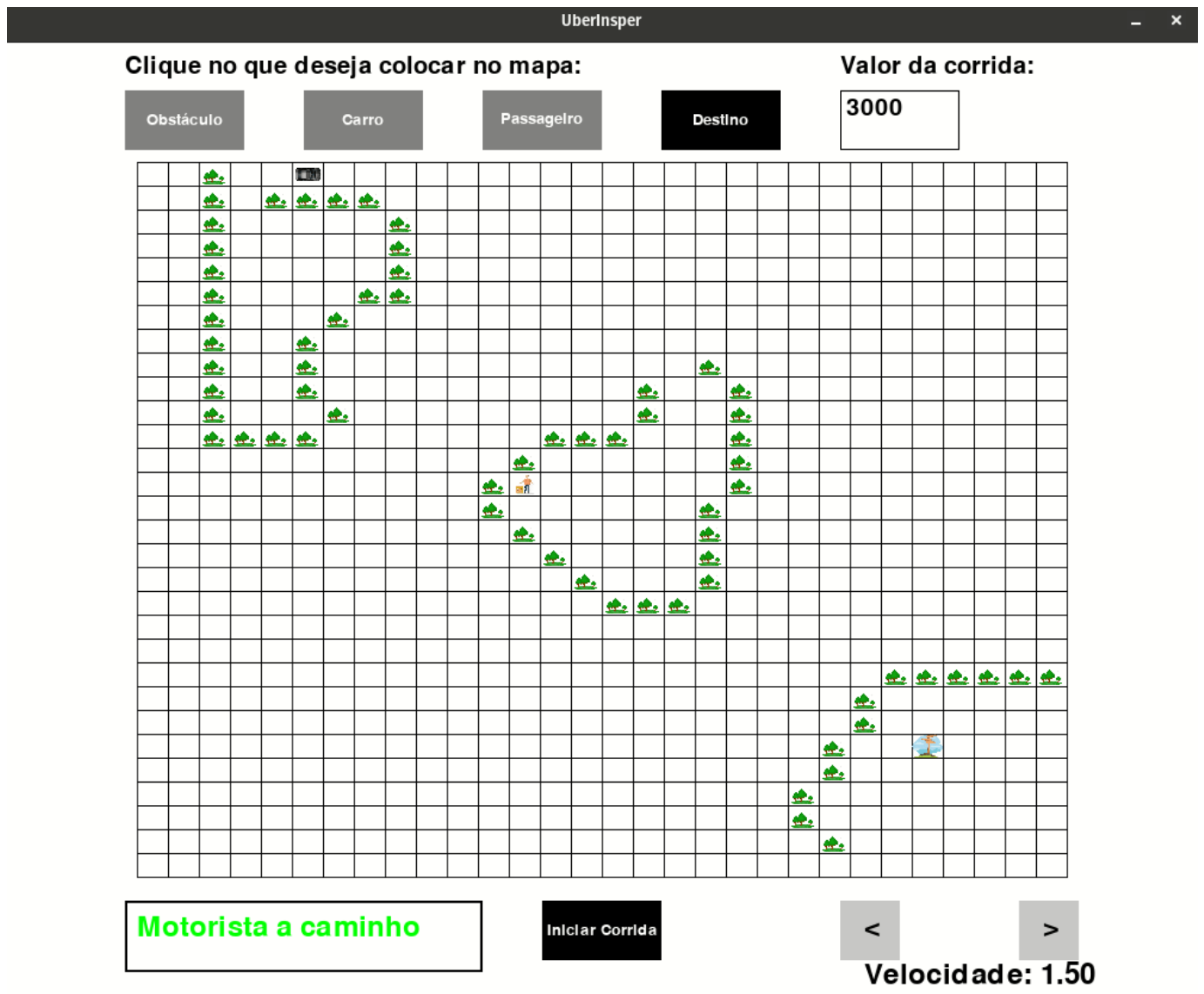




- Para os dois (2) primeiros cenários a solução precisa encontrar resposta ótima na ordem de segundos. Se isto acontecer, a equipe tem garantida nota **C**. Caso contrário, se a equipe entregou todos os itens solicitados, mas as soluções não são boas o suficiente então a nota será **D**.
- Se a solução encontrar resposta ótima na ordem de segundos para todos os cenários apresentados neste documento então a nota da equipe será **C**¹.
- O plano encontrado pelo agente *taxi driver* pode ser apresentado ao final da execução através de um print em modo texto. Neste caso a nota da equipe será **C**. Para alcançar uma nota **B** a equipe deverá implementar uma forma de visualização gráfica onde consegue ver o táxi em movimento e executando as ações. Recomenda-se o uso de `pygame` ou de alguma biblioteca gráfica para ambiente desktop.
- Para alcançar a nota **A** é necessário desenvolver um agente capaz de lidar com configurações mais complexas que as apresentadas aqui neste documento. Configurações com dimensões maiores e com um número maior de obstáculos, incluindo situações onde é impossível alcançar o objetivo. Além disso, a configuração do mapa pode ser feita de forma visual. Em outras palavras, o usuário poderá informar qual a dimensão do mapa, pintar os obstáculos, informar a origem do táxi, informar a origem do passageiro e o seu destino através de uma interface amigável.
- Digamos que este agente é um motorista de aplicativo de transporte. Ele conhece qual será o custo da viagem, ou seja, ir até o passageiro, pegar o mesmo e depois deixar em um ponto específico. Adicione um parâmetro na interface da sua solução que é o valor pago em unidades monetárias pela empresa do aplicativo de transporte. Se o valor do custo da viagem for menor que o valor pago pelo aplicativo de transporte então o agente executa a viagem, caso contrário ele não executa a viagem e informa que o custo da viagem é igual ou superior ao valor pago na interface da sua solução. Ao implementar esta funcionalidade e todas as outras anteriores a sua nota será **A+**.

7.1.3 Exemplo de interface gráfica esperada

Na imagem abaixo é apresentado um exemplo de interface gráfica esperada como resultado do projeto.



1. É importante lembrar que a equipe deve satisfazer também os requisitos de documentação para alcançar esta nota. ↩

🕒 April 1, 2025

8. Referências

8.1 Introdução à Inteligência Artificial

🕒 January 30, 2024

8.2 Algoritmos de Busca

🕒 February 10, 2023

8.3 Constraint Satisfaction Problems

🕒 March 8, 2023

8.4 Busca Competitiva

🕒 April 4, 2023

8.5 Jogos de tabuleiro e busca competitiva

[Implementando um jogador para Liga4.](#)

🕒 April 4, 2023

9. Exercícios e avaliações antigas

9.1 Exercícios adicionais a avaliações antigas

Neste documento você irá encontrar exercícios (problemas) adicionais e avaliações antigas.

9.1.1 Exercícios

- [Lista de exercícios sobre busca em espaço de estados](#)

9.1.2 Avaliações antigas

- [Avaliação intermediária do primeiro semestre de 2023](#)

🕒 March 17, 2024



None