Universidade Federal de Santa Catarina

Centro Tecnológico

Departamento de Automação e Sistemas

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Trabalho 1: Planejamento Automático

Inteligência Artificial

Guilherme Henrique Ludwig
Juliano Ricardo da Silva
Prof. Dr. Jomi Fred Hübner

Florianópolis - SC

Dezembro de 2021

Introdução

Considere o problema dos missionários e canibais visto nas aulas de busca, faça:

- a modelagem desse problema em PDDL;
- utilize um *solver* para obter a solução do problema.

A partir da solução do problema original (3 missionários e 3 canibais), avalie como esse problema pode se tornar "maior"e como o desempenho do *solver* é afetado pelo tamanho do problema.

Conteúdo

1 Modelagem do Problema

O problema dos missionários e canibais é um desafio clássico de lógica muito utilizado dentro do universo de estudo da inteligência artificial, onde o objetivo a ser alcançado, isto é, atravessar todos os canibais e missionários que estão em um lado do rio para o outro lado, deve obedecer à restrição de jamais ter em momento algum, em ambos os lados do rio, um número de canibais superior ao número de missionários. Do contrário, os canibais devorarão o(s) missionário(s) que está(ão) em menor número. A figura 1 ilustra este popular jogo:

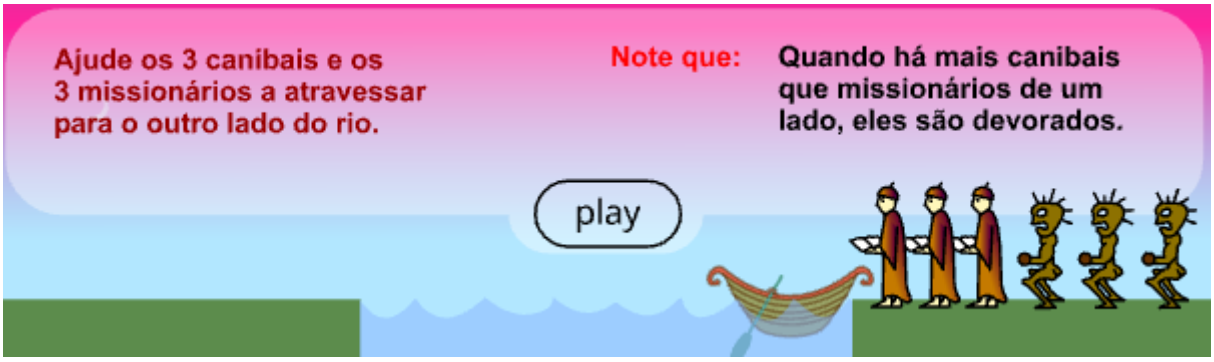

Figura 1: Jogo missionários e canibais. [1]

Para resolver este problema através da técnica de planejamento automático, é necessário definir nos termos da linguagem PDDL o problema que será solucionado. Primeiramente, deve-se definir quais são as condições iniciais do problema, quais são os objetivos e em qual domínio dar-se-á o desenvolvimento da solução. Além disso, os objetos também deverão ser definidos.

Sabe-se que, em planejamento automático, os objetos representam coisas no mundo que nos interessam. Aqui, o barco e os lados do rio serão considerados os objetos, pois através destes que os objetivos serão cumpridos. As condições iniciais representam os estados iniciais do 'mundo' onde os objetos existem, particularmente, a quantidade de

canibais/missionários presentes em cada lado do rio e, também, em qual lado do rio o barco está atracado e, não menos importante, se o barco está vazio ou ocupado.

Os objetivos são os 'fatos'/coisas que deverão ser verdade ao final do planejamento, ou seja, são os estados que deseja-se alcançar. No caso atual, o objetivo é passar todos os missionários e canibais para o outro lado do rio e não ter ninguém no barco.

Partindo agora para a programação na linguagem PDDL, o arquivo que reúne todos estes tópicos abordados anteriormente chama-se *problem.pddl* e é possível conferí-lo integralmente no apêndice 1.

2 Solução do Problema

Para solucionar o problema proposto é necessário definir em um outro arquivo, chamado *domain.pddl*, as ações e os predicados. Da literatura, sabe-se que os predicados são as propriedades dos objetos do mundo que interessam, podendo ser verdadeiras ou falsas. Já as ações são maneiras de se alterar os estados do mundo em questão [2].

Desta forma, a partir do conhecimento dos significados destes termos, fica mais fácil entender o código que foi programado em PDDL, cuja característica típica de linguagem declarativa permite fazer essas representações de maneira simples e direta.

Antes de entrar na parte de programação, é possível desde já abstrair os predicados e ações que deverão ser declarados. Portanto, com relação ao objeto **barco**, os seguintes predicados serão criados:

- **Termos**: free/busy ; **Significado**: o barco está livre ou ocupado;
- **Termos**: PA/NPA; **Significado**: barco pode ou não pode atraves-

sar para o outro lado do rio.

Já com relação ao lado do rio, tem-se que:

- **Termos**: mL ; **Significado**: muda de lado do rio (direita para esquerda);
- **Termos**: barcoNo ; **Significado**: qual lado do rio o barco está atualmente;

Os domínios destas variáveis são, respectivamente, os objetos **barco** e **lado**.

Em PDDL, os predicados, como supracitado, possuem dois valores possíveis (verdadeiro e falso) sendo, portanto, variáveis de estado booleano. Para representar variáveis de estado numérico é necessário utilizar as *functions*. No programa atual, foram criadas duas funções:

- **Termos**: mis; **Significado**: para contabilizar o número de missionários em um lado do rio;
- **Termos**: can; **Significado**: para contabilizar o número de canibais em um lado do rio;

O domínio destas funções é o objeto (tipo) **lado**.

Uma especificidade da linguagem é, primeiro, definir os *requirements*, isto é, requisitos necessários para o planejador (*solver*) considerar. Neste caso, foram selecionados os *requirements*: *typing*, para permitir a utilização da tipagem por nomes na declaração das variáveis, e também a *fluents*, que permite a utilização de variáveis de estado numéricas [3]. Por fim, as *types* que definem simplesmente os tipos dos objetos utilizados, no caso, lado e barco. Para conferir a declaração dos predicados e demais características da linguagem PDDL descritas até agora, basta ir direto ao apêndice 2.

Definido o problema e, posteriormente, os predicados que serão utilizados nas ações, está na hora de definir as ações em si. Agora, a solução será apresentada em duas partes: uma parte para as ações relacionadas às travessias dos missionários e a outra às travessias dos canibais.

Antes de começar com essas ações, é necessário, antes, explicar a lógica de funcionamento de duas ações complementares: **permiteEmbarque** e **atravessaRio_E_Desembarca**.

A ação **permiteEmbarque** possui como parâmetro o objeto barco e tem como **pré-condições** a conjunção de literais através da cláusula lógica *and* que representa a necessidade do barco estar livre e também de possuir permissão para atravessar. Ou seja, os dois predicados devem ser verdadeiros. O efeito desta ação é outra conjunção de literais, onde o barco passa a estar livre (*free* verdadeiro) e não mais ocupado (*busy* falso).

A ação **atravessaRio_E_Desembarca**, possui como parâmetro o objeto barco e como pré-condição o literal NPA (não pode atravessar) ser verdadeiro. Com isso, subentende-se que todos os passageiros foram embarcados e o barco pode atravessar o rio. O **efeito** é simplesmente a negação de NPA e afirmação de PA como verdade, possibilitando que, após atravessar o rio, a ação **permiteEmbarque** possa ser realizada de novo. Além disso, todos os passageiros são desembarcados toda vez que esta ação é cumprida. As duas ações podem ser conferidas no apêndice 2.

Agora, como primeira ação referente à travessia, temos uma ação de travessia híbrida, ou seja, que atravessa um missionário e um canibal. Isso é fundamental para a resolução do problema pois sabe-se de antemão que, em algum momento, essa ação deve ser obrigatoriamente realizada. Para que isso ocorra, é necessário que as seguintes condições

sejam verdadeiras:

- Número de canibais e missionários no lado atual ≥ 1;
- Número de missionários $\geq$ canibais no lado atual, após a partida do barco;
- Número de missionários $\geq$ canibais no outro lado do rio, contando com quem está no barco;

Matematicamente, os segundo e terceiro itens são representados pela soma (quando se quer contar com o passageiro que está no barco) e subtração (quando se quer comparar o número de miss./canibais sem contar com quem está no barco) com um. Com isso, impede-se que se tenha, após a ação, um número desbalanceado de missionários/canibais em ambas as margens.

Existem outras **pré-condições** que, apesar de não terem sido esmiuçadas ainda, serão presentes em todas as ações de embarque, sendo elas:

- Necessidade do barco estar no lado atual onde se está embarcando;
- O barco estar livre;
- A permissão para atravessar ser verdadeira;

Como **efeitos** das ações de embarque, comumente, serão feitos:

- O incremento (*increase*) no valor da função missionário/canibal que está atravessando para o outro lado do rio;
- O decremento (*decrease*) no valor da função missionário/canibal do lado atual do rio;
- A passagem do barco para o outro lado do rio;

- O predicado NPA passa a ser verdadeiro e o PA falso;
- Barco passa a estar ocupado e não livre;

.

Feitas as considerações, de agora em diante serão abordadas apenas as *pré-condições* que definirão efetivamente a lógica de travessia com respeito à restrição imposta.

A ação de embarca (e atravessar) um missionário está, pode-se dizer, fragmentada em duas partes: a primeira parte (**EmbarcaUmMissionário**) é a travessia de um missionário quando há mais de um missionário no lado atual do rio e a segunda parte (**EmbarcaUmMissionárioSozinho**) quando há apenas um missionário. Isso permite ao *planner* realizar a travessia do missionário em cenários diferentes sem que isso implique em violação da restrição. As **pré-condições** para a ação **EmbarcaUmMissionário** são:

- Número de missionários $\geq$ canibais no lado atual após a partida do barco para o outro lado do rio.
- Número de missionários contando com o que está no barco $\geq$ canibais no outro lado após o desembarque.

Para a ação **EmbarcaUmMissionarioSozinho**, tem-se que:

- Número de missionários no lado atual igual a um;
- Número de missionários contando com o que está no barco $\geq$ canibais no outro lado após o desembarque.

Para embarcar dois missionários, o mesmo 'fenômeno' é observado, com a diferença que o número de missionários utilizados para fazer as comparações é o 2, mas também se verifica se o número de missionário

após a partida do barco será maior que o número de canibais que tem do outro lado do rio e também que o número de missionários que 'ficaram' no lado atual seja maior igual ao número de canibais. As ações dos missionários está no apêncice 3.

Da mesma forma que as ações de embarque e travessia dos missionários, construiu-se a lógica de pré-condições para a movimentação dos canibais ao longo do rio. A ação (**EmbarcaUmCanibal**) além de necessitar que exista ao menos um missionário na margem de origem exige que o número da canibais $+\mathbf{1}$ na margem de destino seja sempre menor ou igual ao número de missionários no lado futuro. Obviamente, como nas ações previamente esmiuçadas, é necessário que o barco esteja na margem de origem e que este esteja vago para comportar o canibal que realizará o embarque. Já a ação (**EmbarcaUmCanibalSozinho**) contempla o cenário em que, na margem de destino, não existam missionários.

As ações de **EmbarcamDoisCanibais** e **EmbarcamDoisCanibaisSozinhos** possuem precondições construídas baseadas nos mesmos princípios que as anteriores, com a diferença de que, ao invés de embarcarmos apenas um canibal por vez, embarcamos dois, mudando assim as pré-condições numéricas.

Como efeito das ações de transporte de canibais, acrescentam-se 2 ou 1 canibal a margem de destino e, como consequência, retiram-se 2 ou 1 canibal da margem de origem. Ainda atualiza-se a ocupação do barco (**free ou busy**) e o lado em que se encontra além de atualizar as condições de desembarque após a travessia. Desta forma podemos manter o comportamento original do jogo e permitir sua evolução, uma vez que o número de canibais em cada lado será atualizado permitindo a continuação da busca pela solução pela inteligência empregada pelo

solver. As ações dos canibais está no apêndice 4.

3 Complexificação do Problema

Para tornar o problema mais complexo fizemos uma análise levando em consideração o que mais dificultaria a solução do *game* por nós enquanto jogadores. A maneira mais óbvia é aumentar o número de canibais e missionários, por exemplo, dobrar de 3 para 6 cada um deles. Outra maneira seria de criar novas regras como, por exemplo, apenas o missionário ser capaz de pilotar o barco, ou, ainda, adicionar um ponto intermediário na travessia, onde se manteriam as regras já estabelecidas. Enfim, a complexificação do problema poderia ser feita de diversas maneiras, mas o importante a ser ressaltado é se, com o programa atual, o *solver* seria capaz de encontrar um plano e, caso sim, qual plano seria este.

Para isso, consideramos a primeira proposição de mudança, alterando o número de missionários e canibais para 4 cada um. Então, executou-se novamente o código e, logo de cara, percebeu-se que para encontrar um plano para este problema, o programa levaria muito mais tempo, implicando em um custo computacional bem maior. A figura 2 mostra que, mesmo após mais de duas horas de execução, o programa foi incapaz de encontrar um plano viável. A causa disso? Bem, vai desde a ineficiência do *solver* utilizado para lidar com problemas de instâncias superiores, até a própria inadequação do programa em si que, apesar de funcionar muito bem para as condições iniciais do desafio, tem seu desempenho drasticamente reduzido quando estas condições se alteram.

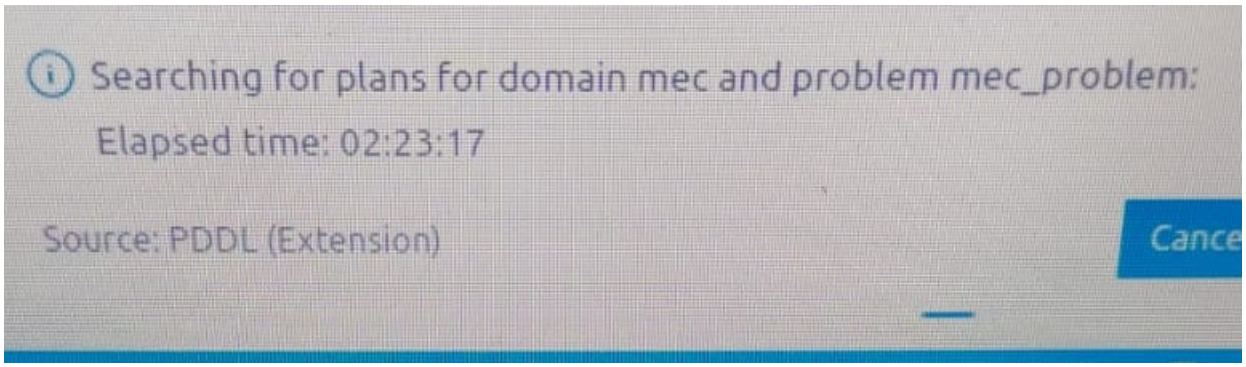

Figura 2: Programa em execução.

4 Avaliação do *solver*

O solver **optic-clp** foi utilizado pois fora o único encontrado que implementava as precondições numéricas, ferramental fundamental para a solução do problema dos canibais e missionários. Constatou-se que os solvers apresentados em aula possuíam uma série de limitações que impossibilitavam o conhecimento de um plano que seguisse o domínio implementado por nós. Chegamos a propor soluções utilizando o *editor.planning.domains* mas, pelo fato de esta ferramenta parecer não implementar as precondições idealizadas neste projeto, deixamo-o de lado. As soluções encontradas na ferramenta abandonada não consideravam o número de passageiros do barco (missionários e canibais), apenas os que já haviam desembarcado em uma das margens, o que torna o problema muito distante do real proposto pelo *game* clássico.

Apesar de termos encontrado uma solução para o jogo com o solver escolhido, ainda assim conseguimos encontrar algumas limitações neste. Uma delas é que o *optic* não aceita precondições do tipo *or* e *not*, o que nos fez repensar o desenvolvimento de nossa solução. Exemplo: Sabíamos que o número de canibais deveria ser sempre menor ou igual ao número de missionários nas margens do rio. Mas, caso o número de missionários fosse zero em uma das margens poderíamos ter

1,2,10,50,500 ou até 1000 canibais nesta. Isso é um caso clássico do uso do *or* para solucionar o problema das travessias. Entretanto, como não podíamos utilizar o condicional *or*, precisamos criar duas ações para uma mesma ação (embarcar um missionário e embarcar um missionário sozinho, por exemplo).

Apesar do problema encontrado, conseguimos contornar a situação. Ainda, constatamos que, de todos os solvers pesquisados e utilizados em nosso desenvolvimento, este foi sem dúvidas o que nos proporcionou maior ferramental para o alcance de nossos objetivos.

Constatamos ainda que, para um número de 6 indivíduos (sendo 3 canibais e 3 missionários), o solver acha um plano com relativa facilidade. Entretanto, conformes aumentamos o número de participantes do *game*, o encontro do plano de resolução torna-se demasiadamente custoso do ponto de vista temporal.

O plano encontrado pelo solver para a condição onde há 3 missionários e 3 canibais pode ser visto a seguir:

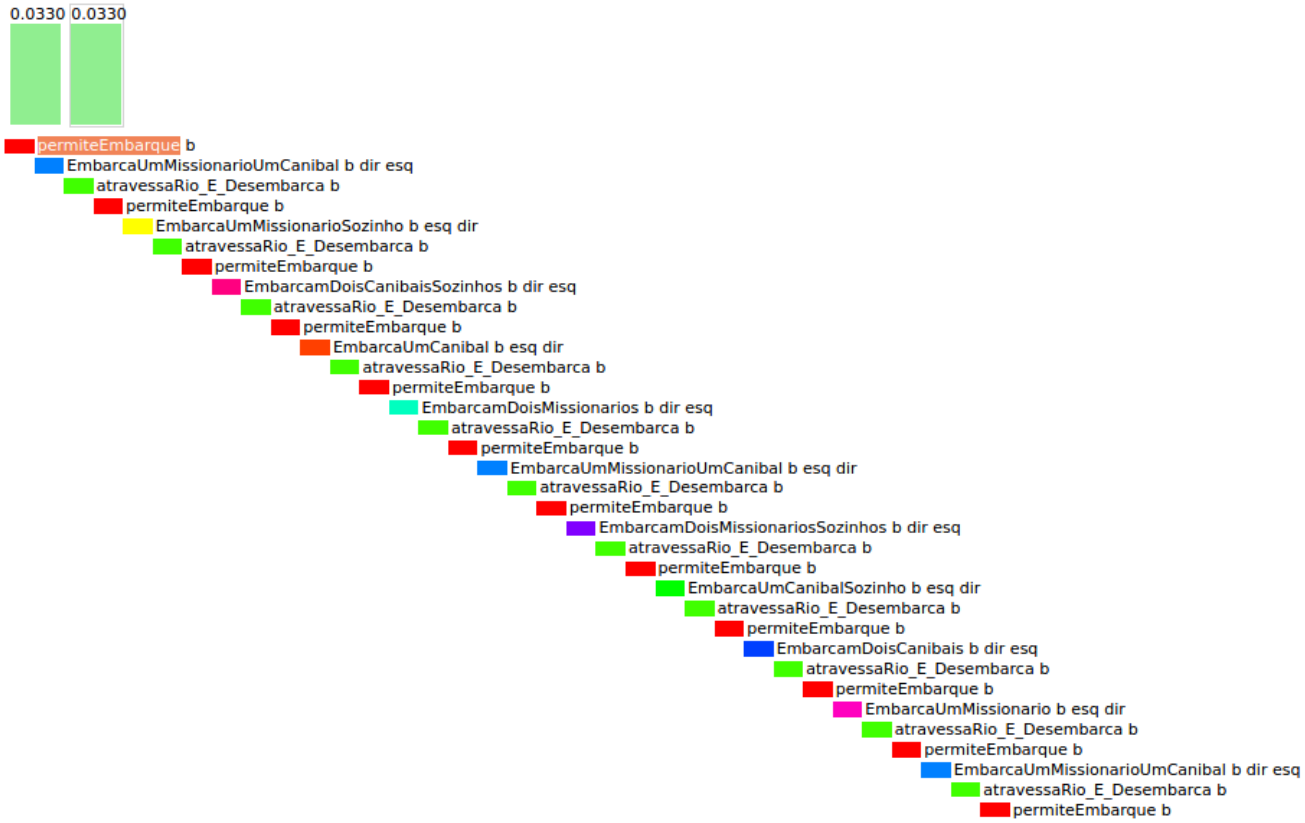

Figura 3: Plano encontrado pelo solver.

Para ilustrar os resultados, segue o passo a passo no próprio jogo:

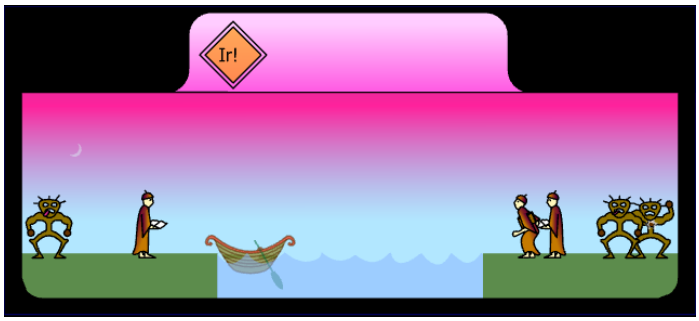

Figura 4: Passo 1.

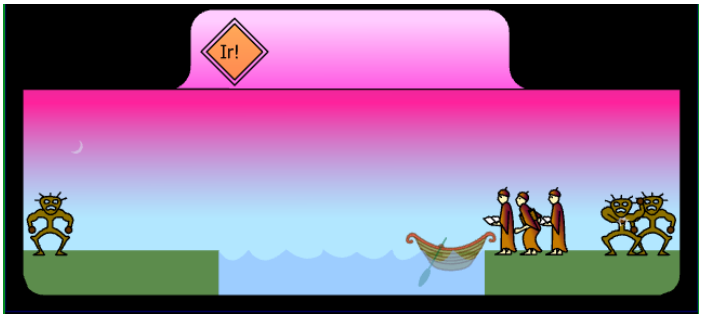

Figura 5: Passo 2.

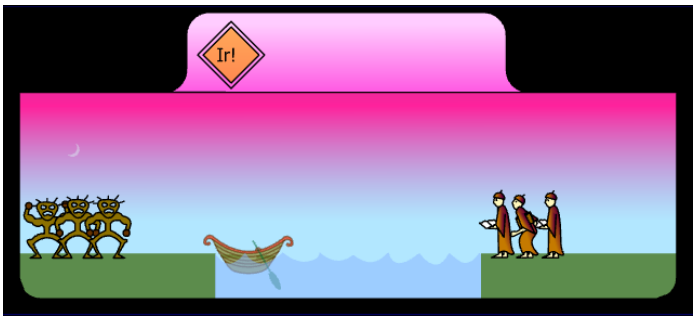

Figura 6: Passo 3.

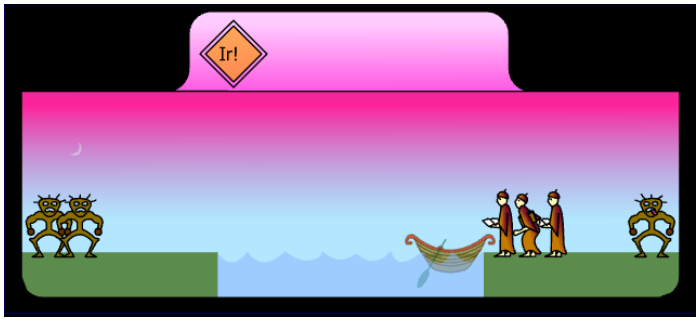

Figura 7: Passo 4.

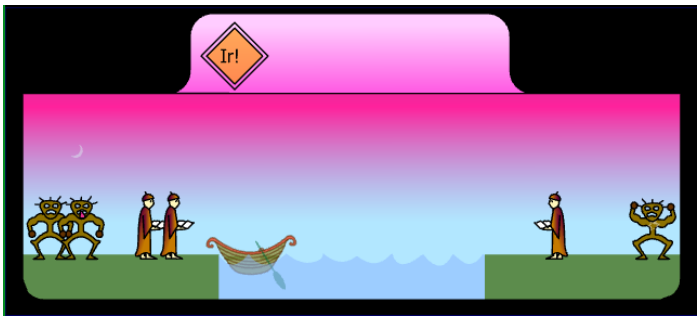

Figura 8: Passo 5.

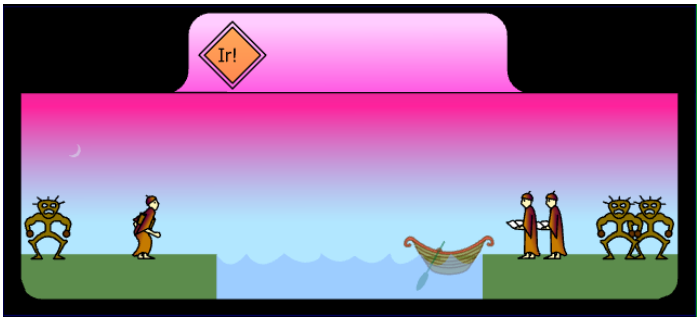

Figura 9: Passo 6.

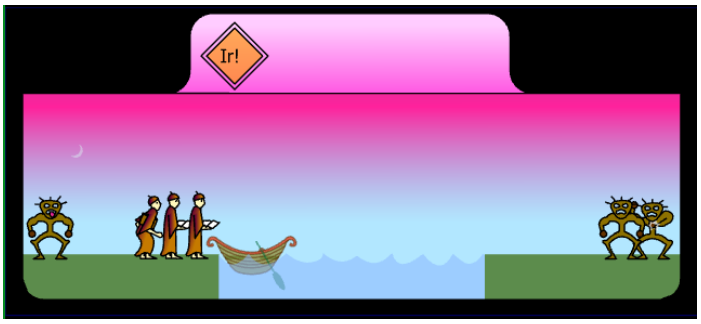

Figura 10: Passo 7.

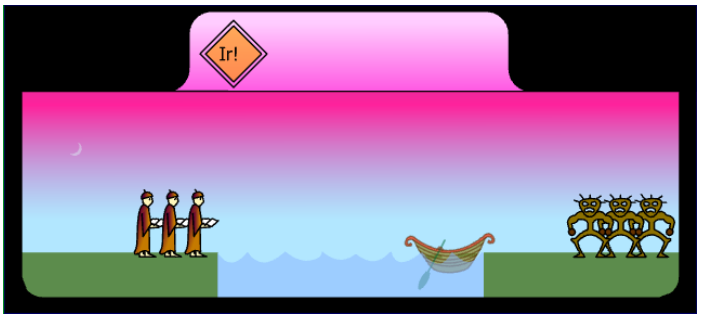

Figura 11: Passo 8.

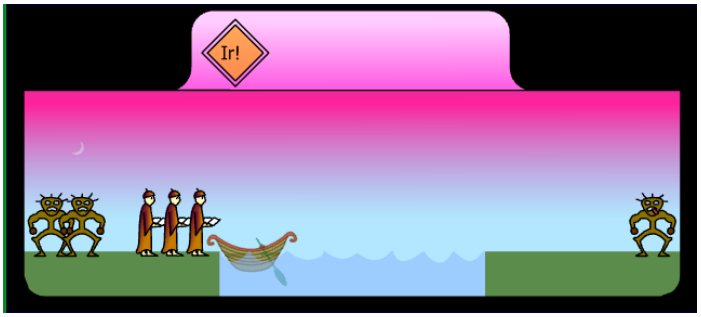

Figura 12: Passo 9.

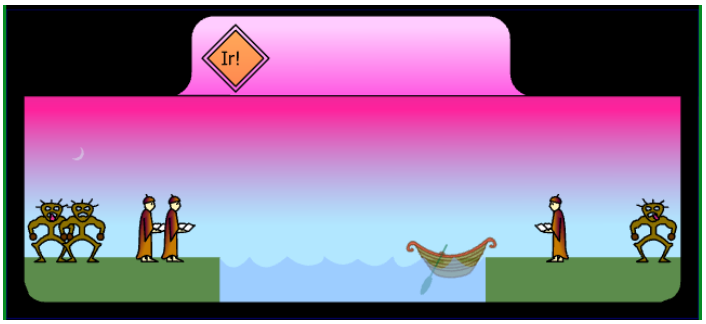

Figura 13: Passo 10.

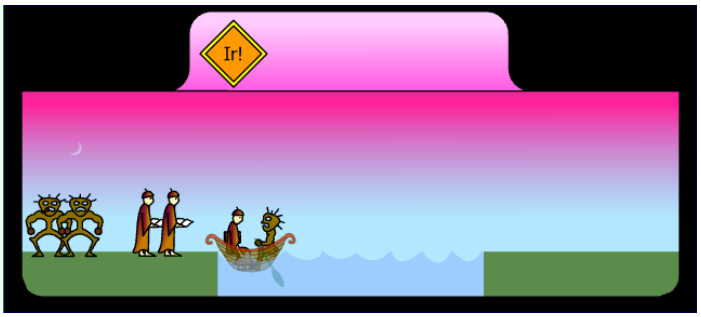

Figura 14: Passo 11.

Figura 15: Passo Final.

Parece claro que embora não pareça a solução ótima o solver cumpre de maneira bastante satisfatória o problema proposto, através do método A^*, nos levando a uma solução final que "zera"o jogo num tempo muito menor do que levaria qualquer jogador humano (0.03 segundos). Portanto a solução é bastante satisfatória.

Referências

1 360, J. *Missionários e Canibais.* Acesso em: Dezembro de 2021. Disponível em: <https://www.jogos360.com.br/missionarios_e_canibais.html>. Citado na página 4.

2 HELMERT, M. *An Introduction to PDDL.* [s.n.]. Acesso em: Dezembro de 2021. Disponível em: <https://www.cs.toronto.edu/~sheila/2542/s14/A1/introtopddl2.pdf>. Citado na página 5.

3 YOUNES, M. L. L. H. L. *PDDL1.0: An Extension to PDDL for Expressing Planning Domains with Probabilistic Effects.* Acesso em: Dezembro de 2021. Disponível em: <http://reports-archive.adm.cs.cmu.edu/anon/2004/CMU-CS-04-167.pdf>. Citado na página 6.

Apêndice

Listing 1: Problema em PDDL.

```
(define
(problem mec_problem)

(:domain mec)

(:objects
 b - barco dir esq - lado
)

(:init

        (barcoNo dir)
        (not(free b))
        (mL dir esq)
        (mL esq dir)
        (busy b)
        (pA b)
        (= (can dir) 3)
        (= (can esq) 0)
        (= (mis dir) 3)
        (= (mis esq) 0)

)

(:goal (and

    (barcoNo esq)
        (= (mis esq) 3)
        (= (can esq) 3)
    (free b))

)

)
```

Listing 2: Domínio em PDDL.

```
(define (domain mec)

(:requirements
 :typing
 :fluents
)
(:types
 lado
 barco
)

(:predicates
 (mL ?dir ?esq - lado)          ;mL = muda lado
 (free ?b      - barco)   ;lugar no barco vazio
 (busy ?b      - barco)   ;lugar no barco ocupado
 (PA ?b        - barco)  ;pode atravessar
 (NPA ?b       - barco)   ;nao pode atravessar
 (barcoNo ?m   - lado)    ;barco na margem
)

(:functions
 (mis ?m - lado) ;conta missionarios de um lado do rio
 (can ?m - lado) ;conta canibais de um lado do rio
)

(:action permiteEmbarque
 :parameters (?b - barco)
 :precondition (and(busy ?b)(pA ?b))
 :effect (and(free ?b)(not(busy ?b)))
)

(:action atravessaRio_E_Desembarca
 :parameters(?b - barco)
 :precondition(NPA ?b)
 :effect (and (not(NPA ?b))(PA ?b))
)
```

Listing 3: Domínio em PDDL.

```
  ;ol --> outro lado do rio
  ;la  --> lado atual do rio

(:action EmbarcaUmMissionarioUmCanibal
:parameters(?b - barco ?lA ?oL - lado)
:precondition(and(mL ?lA ?oL)(PA ?b)(barcoNo ?lA)(free ?b)(>=(mis ?lA)1)
            (>=(can ?lA)1)(>=(-(mis ?la)1)(-(can ?la)1))(>=(+(mis ?ol)1)
            (+(can ?ol)1)))
:effect(and(not(barcoNo ?lA))(barcoNo ?oL)(not(PA ?b))(NPA ?b)(not(free ?b))
       (busy ?b)(increase(can ?oL)1)(increase (mis ?oL)1)(decrease(can ?lA)1)
       (decrease (mis ?lA)1))
)

(:action EmbarcaUmMissionario
:parameters (?b - barco ?lA ?oL - lado)
:precondition (and(mL ?lA ?oL)(PA ?b)(barcoNo ?lA)(free ?b)(>=(mis ?la)1)
             (>= (-(mis ?lA) 1)(can ?lA))(>= (+ (mis ?oL) 1)(can ?oL)))
:effect(and(not(barcoNo ?lA))(barcoNo  ?oL)(not(PA ?b))(NPA ?b)(not(free ?b))
       (busy ?b) (increase (mis ?oL) 1)(decrease (mis ?lA)1))
)

(:action EmbarcaUmMissionarioSozinho
:parameters (?b - barco ?lA ?oL - lado )
:precondition(and(mL ?lA ?oL)(pA ?b)(barcoNo ?lA)(free ?b)(=(mis ?lA) 1)
            (>= (+ (mis ?oL) 1) (can ?oL)) )
:effect(and (not (barcoNo ?lA)) (barcoNo ?oL)
(not(PA ?b))(NPA ?b)(not(free ?b))
       (busy ?b)(increase (mis ?oL) 1) (decrease (mis ?lA) 1) )
)

(:action EmbarcamDoisMissionarios
:parameters (?b - barco ?lA ?oL - lado )
:precondition(and(mL ?lA ?oL)(pA ?b)(barcoNo ?lA) (free ?b)(>= (mis ?lA) 2)
             (>=(+(mis ?ol)2)(can ?ol))(>=(-(mis ?la)2)(can ?la)))
:effect(and(not (barcoNo  ?lA)) (barcoNo ?oL)(not(PA ?b))(NPA ?b)(not(free ?b))
       (busy ?b)(increase (mis ?oL) 2)(decrease (mis ?lA) 2))
)
```

Listing 4: Continuação domínio em PDDL.

```
(:action EmbarcamDoisMissionariosSozinhos
:parameters (?b - barco ?lA ?oL - lado)
:precondition(and(mL ?lA ?oL)(pA ?b) (barcoNo  ?lA) (free ?b)(=(mis ?lA) 2)
            (>= (+(mis ?oL) 2) (can ?oL)))
:effect(and (not(barcoNo ?lA))(barcoNo ?oL) (not(PA ?b))(NPA ?b)(not(free ?b))
       (busy ?b)(increase (mis ?oL) 2)(decrease (mis ?lA)2))
)

(:action EmbarcaUmCanibal
:parameters (?b - barco ?lA ?oL - lado)
:precondition(and(mL ?lA ?oL)(pA ?b)(barcoNo ?lA)(free ?b)(>=(can ?la)1)
            (>= (mis ?oL) (+ (can ?oL) 1)) )
:effect(and(not(barcoNo ?lA))(barcoNo  ?oL)(not(PA ?b))(NPA ?b)(not(free ?b))
       (busy ?b) (increase (can ?oL) 1)(decrease (can ?lA)1))
)

(:action EmbarcaUmCanibalSozinho
:parameters (?b - barco ?lA ?oL - lado)
:precondition(and(mL ?lA ?oL)(pA ?b)(barcoNo ?lA)(free ?b)(>= (can ?lA) 1)
            (= (mis ?oL) 0))
:effect(and (not (barcoNo  ?lA))(barcoNo ?oL)(not(PA ?b))(NPA ?b)(not(free ?b))
       (busy ?b)(increase (can ?oL) 1)(decrease (can ?lA) 1) )
)

(:action EmbarcamDoisCanibais
:parameters (?b - barco ?lA ?oL - lado)
:precondition(and (mL ?lA ?oL) (pA ?b)(barcoNo ?lA) (free ?b)(>= (can ?lA) 2)
            (<=(+(can ?ol)2)(mis ?ol)))
:effect(and (not (barcoNo ?lA))(barcoNo ?oL)(not(PA ?b))(NPA ?b)(not(free ?b))
       (busy ?b) (increase (can ?oL) 2) (decrease (can ?lA) 2) )
)

(:action EmbarcamDoisCanibaisSozinhos
:parameters (?b - barco ?lA ?oL - lado)
:precondition(and (mL ?lA ?oL)(pA ?b)(barcoNo ?lA)(free ?b)(>=(can ?lA)2)
            (= (mis ?oL)0))
:effect (and (not (barcoNo  ?lA))(barcoNo  ?oL)(not(PA ?b)) (NPA ?b)
        (not(free ?b))(busy ?b)(increase(can ?oL) 2)(decrease(can ?lA)2))
)

)
```