UNIVERSIDADE FEDERAL DE ITAJUBÁ – UNIFEI
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

APLICABILIDADE DE PLANEJADORES AUTOMÁTICOS EM JOGOS E ESTUDO DE CASO EM UM AMBIENTE DE SIMULAÇÃO

Christiano Henrique Rezende

Itajubá, julho de 2018

UNIVERSIDADE FEDERAL DE ITAJUBÁ – UNIFEI
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Christiano Henrique Rezende

APLICABILIDADE DE PLANEJADORES AUTOMÁTICOS EM JOGOS E ESTUDO DE CASO EM UM AMBIENTE DE SIMULAÇÃO

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica, como parte dos requisitos para obtenção do Título de Mestre em Engenharia Elétrica.

Área de Concentração: Automação e sistemas elétricos industriais.

Orientador: Prof. Dr. Luiz Edival de Souza

Julho de 2018
Itajubá, MG

UNIVERSIDADE FEDERAL DE ITAJUBÁ – UNIFEI
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

APLICABILIDADE DE PLANEJADORES AUTOMÁTICOS EM JOGOS E ESTUDO DE CASO EM UM AMBIENTE DE SIMULAÇÃO

Christiano Henrique Rezende

Dissertação aprovada por banca examinadora em 13 de Agosto de 2018, conferindo ao autor o título de **Mestre em Ciências em Engenharia Elétrica**.

Banca Examinadora:
Prof. Dr. Leonardo Rocha Olivi
Prof. Dr. Guilherme Sousa Bastos

Itajubá
2018

Christiano Henrique Rezende

APLICABILIDADE DE PLANEJADORES AUTOMÁTICOS EM JOGOS E ESTUDO DE CASO EM UM AMBIENTE DE SIMULAÇÃO

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica, como parte dos requisitos para obtenção do Título de Mestre em Engenharia Elétrica.

Trabalho aprovado. Itajubá, 13 de Agosto de 2018:

Prof. Dr. Luiz Edival de Souza
(Orientador)

Prof. Dr. Leonardo Rocha Olivi

Prof. Dr. Guilherme Sousa Bastos

Itajubá

13 de agosto de 2018

RESUMO

A indústria de jogos digitais está em constante crescimento e um dos desafios enfrentados pelos desenvolvedores é manter uma experiência atrativa para os jogadores. Para tanto, uma das ferramentas utilizadas é a inteligência artificial. Dentre as diversas áreas da inteligência artificial, o planejamento automático se destaca pela sua flexibilidade de resolução de problemas utilizando uma mesma codificação. Visando o estudo desta técnica, foi desenvolvido um ambiente de simulação nomeado de Labirinto do Minotauro e um agente inteligente utilizando técnicas de planejamento automático para atuar nele. Utilizando este ambiente, foram discutidos alguns casos pertinentes evidenciando suas peculiaridades. Em sequência foi avaliado a aplicabilidade do estudo desenvolvido em outras áreas, apresentando exemplos de problemas e os solucionando usando a abordagem desenvolvida neste trabalho.

PALAVRAS CHAVE: jogos digitais, inteligência artificial, planejamento automático, agente inteligente, labirintos.

ABSTRACT

The digital games industry is in a constant advance and keep an entertaining experience to the players is one of the developer's main challenge, to achieve this, one of the used tools is the artificial intelligence. Between the fields of the artificial intelligence, automated planning is recognized for their flexibility to solve diverse problems using the same codding. Seeking this technique's study, a simulation environment named Minotaur's Maze and an intelligent agent using automated planning techniques to solve it were developed. Using this environment, some pertinent cases were discussed where their particularity was shown. In sequel was evaluated the applicability of this study in other fields, presenting examples of problems and their solutions using this work's approach.

KEYWORDS: digital games, artificial intelligence, automated planning, intelligent agents, mazes.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Esse capítulo contextualiza a proposta, dando considerações gerais do tema, falando sobre o estado atual da indústria de jogos, o porquê de esta ser importante e relevante ao trabalho, a aplicação da IA nesta indústria e o contexto atual dos planejadores nos jogos. Por fim apresenta os objetivos e como este trabalho está organizado.

1.1. CONSIDERAÇÕES GERAIS

A indústria de jogos digitais é um setor em rápida ascensão. Em 2014 o estudo realizado por Fleury *et al.* (2014) contabilizou 148 empresas desenvolvedoras de jogos distribuídas pelo Brasil[1] e em 2018 são computadas pelo menos 215 empresas dedicadas a produzir jogos autorais[2].

Umas das ferramentas utilizadas pelos desenvolvedores para fornecer uma experiência atrativa aos jogadores, é a inteligência artificial. Um dos primeiros jogos que as pessoas têm lembrança de possuir uma inteligência artificial atrativa é o *Pac-Man*[3]. A inteligência de *Pac-Man* é baseada em somente uma máquina de estados onde cada um dos fantasmas ou está caçando o jogador ou fugindo dele, sendo que em cada junção do mapa os mesmos têm uma chance parcialmente randômica de ou manter a função em que estão ou escolher uma rota alternativa de perseguição (MILLINGTON e FUNGE, 2009).

Desde então, a inteligência artificial em jogos passou a ser mais explorada, e hoje existe uma grande diversidade destes que a utiliza, usando desde técnicas simples, como a do *Pac-Man*, a técnicas mais complexas, como o jogo *Creatures*[4].

Em *Creatures*, cada criatura possui redes neurais artificiais que controlam os processos sensório-motores e de aprendizado, sistemas bioquímicos artificiais para controle do metabolismo energético e regulação hormonal do comportamento. Tanto as redes neurais, quanto os sistemas bioquímicos são geneticamente especificados para possibilitar a adaptação evolutiva através da reprodução sexuada (GRAND, *et al.*, 1997).

---

[1] Dados disponíveis em: http://mapadaindustriadejogos.com.br/ (Acesso em 18 de junho de 2018)
[2] Dados disponíveis em: http://www.braziliangamecompanies.com/ (Acesso em 18 de junho de 2018)
[3] Midway Games West, Inc., 1979.
[4] Cyberlife Technology Ltd., 1997.

Existem duas conferências que abordam o assunto de inteligência artificial em jogos, a *IEEE Conference on Computational Intelligence and Games* (CIG) e a *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (AIIDE), além da revista *IEEE Transactions on Computational Intelligence and AI in Games* (T-CIAIG).

A escolha da técnica de inteligência artificial a ser utilizada no jogo depende da experiência que o desenvolvedor quer passar para os jogadores. O desenvolvedor precisa escolher uma heurística que melhor se adeque às suas necessidades, sendo que a inteligência pode ter uma maior reatividade, que é a capacidade de perceber o ambiente e selecionar ações apropriadas ao momento, ou um maior controle autoral, que é a capacidade do desenvolvedor ajustar a inteligência para garantir a experiência desejada (DILL, 2014).

Uma escala comparando diversas técnicas de inteligência artificial e a compensação entre seu controle autoral e sua reatividade está ilustrada na Figura 1.

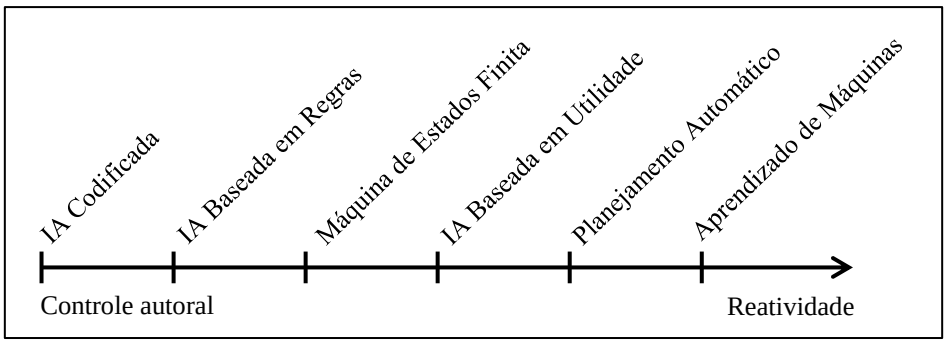


Figura 1 – Compensação entre controle autoral e reatividade.
Adaptado de (DILL, 2014).

A pesquisa desenvolvida por Yannakakis e Togelius (2015) fez um panorama sobre o estudo da inteligência artificial em jogos, explicitando dez campos desse estudo e a relação desses campos com seis áreas principais da inteligência artificial. Ainda, nesta pesquisa, foi apresentado que a área de Busca e Planejamento abrange 6 dessas 9 áreas de estudos sobre inteligência artificial em jogos (uma das áreas foi deliberadamente ignorada por todas as técnicas serem aplicáveis), tornando-as um setor de grande interesse para se realizar pesquisas.

Um marco para o estudo de planejamento em jogos foi o trabalho desenvolvido por Orkin (2006), que foi reconhecido como o primeiro caso bem-sucedido de aplicação

de técnicas de planejamento automático em um jogo comercial. Alguns jogos comerciais que utilizam técnicas de planejamento são: *F.E.A.R.*[5], *Dirty Harry*[6], *Tomb Raider*[7], *Middle-earth: Shadow of Mordor*[8], *Killzone 2*[9] e *3*[10], *Transformers: Fall of Cybertron*[11], *Dying Light*[12] e *PlannedAssault*, que é um gerador de missão *online* para *ARMA II*[13] (NEUFELD, *et al.*, 2017).

A razão do planejamento automático ser utilizado em alguns jogos comerciais, além de aumentar a performance do jogo, é também, reduzir a complexidade do código da inteligência artificial e aumentar a flexibilidade de sua utilização (ČERNÝ, *et al.*, 2016). Essa flexibilidade faz com que a inteligência resolva problemas diversos e que agentes diferentes utilizem a mesma codificação com poucas alterações.

1.2. JUSTIFICATIVA

Como demonstrado por Orkin (2006), a utilização de técnicas de inteligência artificial em ambiente de jogos torna a experiência do jogador mais atrativa. Em especial o planejamento automático faz com que os agentes inteligentes tomem ações racionais, criando a ilusão da inteligência, ao mesmo tempo que podem surpreender os jogadores com estratégias inesperadas.

Pelo lado do desenvolvedor, utilizar planejadores automáticos na tomada de decisões dos seus agentes alcança uma maior modularidade do código, possibilitando o reuso em agentes com características distintas e com poucas modificações, como demonstrado nos trabalhos de Orkin (2003, 2004, 2005) e Soemers e Winands (2016).

Na literatura existem alguns trabalhos que ponderam a utilização de planejadores automáticos em jogos. Neufeld *et al.* (2017) apresentam uma pesquisa sobre sistemas de planejamento utilizado em jogos comerciais, ponderando as propriedades dos sistemas desenvolvidos em etapas e apontando futuras oportunidades de tópicos para serem

[5] Sierra Entertainment, 2005.
[6] Warner Bros. Interactive, cancelado.
[7] Square Enix, 2013.
[8] Warner Bros. Interactive, 2014.
[9] Sony Computer Entertainment, 2009.
[10] Sony Computer Entertainment, 2011.
[11] Activision, 2012.
[12] Warner Bros. Interactive, 2015.
[13] Bohemia Interactive, 2009.

pesquisadas. Černý *et al.* (2016) comparam alguns planejadores que obtiveram boas classificações em competições anteriormente com agentes utilizando técnicas de inteligência reativas.

Algumas pesquisas desenvolvidas sobre inteligência artificial aplicada a jogos abordam a utilização das técnicas de planejamento automático e árvores de decisões para aumentar a ilusão de vida dos personagens não controlados pelo jogador (NPC – *Non-Player Character*). Alguns trabalhos envolvendo esta área foram apresentados por Burke *et al.* (2001), Isla e Blumberg (2002), Mateas e Stern (2002), Tencé *et al.* (2010), Cianciulli e Vassos (2013) e Mahmound *et al.* (2014).

Competições de inteligência artificial aplicada a jogos geraram trabalhos como os agentes desenvolvidos por Björnsson e Finnsson (2009) para resolver diversos tipos de jogos, os agentes de Togelius *et al.* (2010) e Bojarski e Congdon (2010) para jogar *Mario Bros.*[14] e o agente desenvolvido por Du *et al.* (2015) para jogar *Angry Birds*[15].

1.3. OBJETIVOS

O objetivo deste trabalho é estudar e utilizar um planejador automático para prover um agente com inteligência baseada em conhecimentos. Em conjunto com o objetivo anterior, é preciso estruturar e desenvolver um ambiente de simulação para que seja realizado testes com este agente, a fim de verificar os diversos problemas que esta inteligência consegue resolver, além do ambiente ser disponibilizado para utilização como material de intervenção pedagógica. Um objetivo secundário é verificar a aplicabilidade da técnica utilizada neste trabalho nas áreas da indústria 4.0 e em robôs de resgate e segurança pública.

1.4. ORGANIZAÇÃO DO TRABALHO

Este trabalho está disposto na seguinte maneira. No Capítulo 2 é exposto uma fundamentação teórica dos conceitos básicos sobre inteligência artificial, agentes inteligentes, em especial os agentes baseados em objetivos e os baseados em conhecimento. Além de apresentar os conceitos de planejamento automático e descrever

[14] Nintendo, 1985.
[15] Rovio, 2009.

o planejador utilizado neste trabalho, o *FF Planner*. No Capítulo 3 é apresentado o ambiente do Labirinto do Minotauro, feito a sua análise e explicado o modo em que foi desenvolvido o ambiente de simulação. Ainda neste capítulo, é realizado um detalhamento sobre a construção da inteligência artificial do agente, sua integração com a simulação e o funcionamento do programa como um todo. No Capítulo 4 são feitas análises de casos específicos envolvendo o labirinto do minotauro, evidenciando alguns resultados. No Capítulo 5 são analisadas as aplicabilidades das técnicas utilizadas no trabalho nas áreas da indústria 4.0 e em robôs de resgate e segurança. Por fim, no Capítulo 6, algumas considerações finais sobre o trabalho são feitas, e são apresentadas perspectivas para o desenvolvimento de futuros trabalhos.

2. FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo será apresentada a teoria necessária para o entendimento do trabalho, bem como referenciados os materiais adicionais a serem utilizados para uma consulta mais aprofundada sobre o assunto.

2.1. INTELIGÊNCIA ARTIFICIAL

A Inteligência Artificial, ou somente IA, é um campo de estudo que abrange diversas disciplinas, como filosofia, matemática, economia, neurociência, psicologia, engenharia de computadores, teoria de controle e cibernética e linguística. Além de buscar compreender o processo de inteligência, esta área busca também construir entidades inteligentes (RUSSELL e NORVIG, 2013).

A IA possui diversas definições, que podem ser dispostas ao longo de duas dimensões. A primeira estuda os processos comportamentais, ou como agir, e os processos de pensamento e raciocínio. A segunda avalia o sucesso do sistema como sendo fiel à performance humana ou possuindo uma performance ideal, ou racional. Um sistema é racional se ele toma a melhor decisão, dado o que sabe (RUSSELL e NORVIG, 2013).

A combinação dessas duas dimensões gera quatro abordagens para a IA:

- Agindo de forma humana: a abordagem do teste de Turing;
- Pensando de forma humana: a estratégia de modelagem cognitiva;
- Pensando racionalmente: a abordagem das "leis do pensamento" ou da lógica;
- Agindo racionalmente: a abordagem de agente racional.

A abordagem do agente racional tem como objeto de estudo os agentes.

Segundo Russell e Norvig (2013), "um agente é simplesmente algo que age". Mais do que somente agir, esta abordagem espera que os agentes tenham capacidade de resolver problemas com pouca ou nenhuma influência humana, consiga perceber o ambiente de trabalho, adapte-se a mudanças e consiga perseguir metas. Já o agente racional é um agente que realiza as ações de uma maneira a obter o melhor resultado, ou o melhor resultado esperado, no caso do agente não conhecer todas as informações do ambiente.

2.1.1. Agentes Inteligentes

Um agente pode ser definido como uma entidade capaz de perceber o ambiente usando sensores e de agir sobre o mesmo utilizando-se de atuadores (RUSSELL e NORVIG, 2013). Uma representação deste conceito pode ser visualizada na Figura 2.

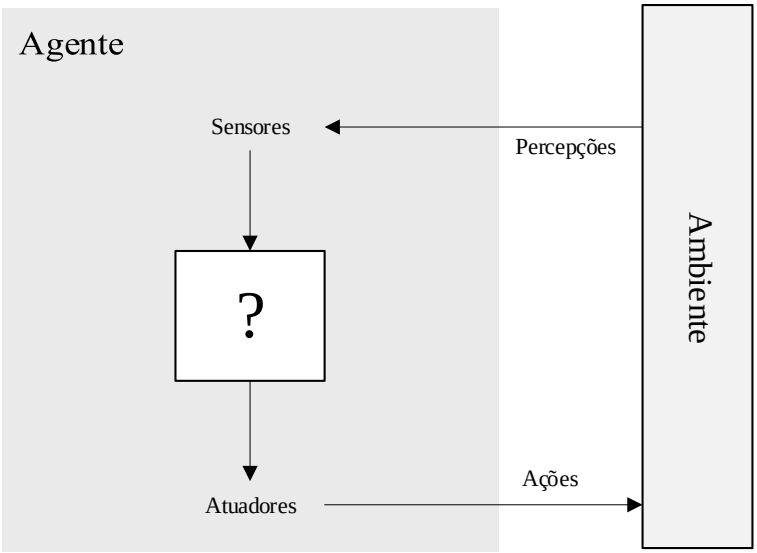


Figura 2 – Estrutura básica de um agente inteligente.
Adaptado de (RUSSELL e NORVIG, 2013).

Um agente robótico pode perceber o ambiente utilizando sensores térmicos, ultrassônicos e câmeras, e atuar sobre o mesmo utilizando-se de braços robóticos. Um agente de *software* pode receber suas percepções de dados de arquivos, entradas de *mouse* e teclado ou de áudio de microfone, e atuar no ambiente ao exibir mensagens no monitor, tocar uma faixa de áudio ou gravar dados em arquivos externos.

Percepção é o termo utilizado para referenciar as entradas de informações que o agente recebe em um determinado instante. Um agente pode decidir a ação a ser executada usando as percepções de um instante qualquer dado ou pode depender de uma sequência inteira de percepções que o agente já pôde coletar (RUSSELL e NORVIG, 2013).

Russell e Norvig (2013) apresentam cinco estruturas básicas de programa de agentes: Agentes reativos simples; Agentes reativos baseados em modelos; Agentes baseados em objetivos; Agentes baseados na utilidade e Agentes com aprendizagem.

Neste trabalho será tratado apenas os agentes baseados em objetivos.

2.1.2. Agentes baseados em objetivos

Um agente conhecer o estado atual do mundo e possuir o seu modelo, nem sempre é o suficiente para que ele possa escolher uma ação. Em situações que mais de uma ação é aplicável, o agente não saberá decidir qual delas realizar se não possuir uma regra específica para a situação e se não possuir informações sobre objetivos.

Um agente baseado em objetivos é aquele que combina o conhecimento do estado atual, o modelo que diz como o mundo evolui naturalmente, como suas ações modificam o ambiente e objetivos pré-definidos para decidir as ações que serão executadas (RUSSELL e NORVIG, 2013). A Figura 3 ilustra a estrutura desse tipo de agente.

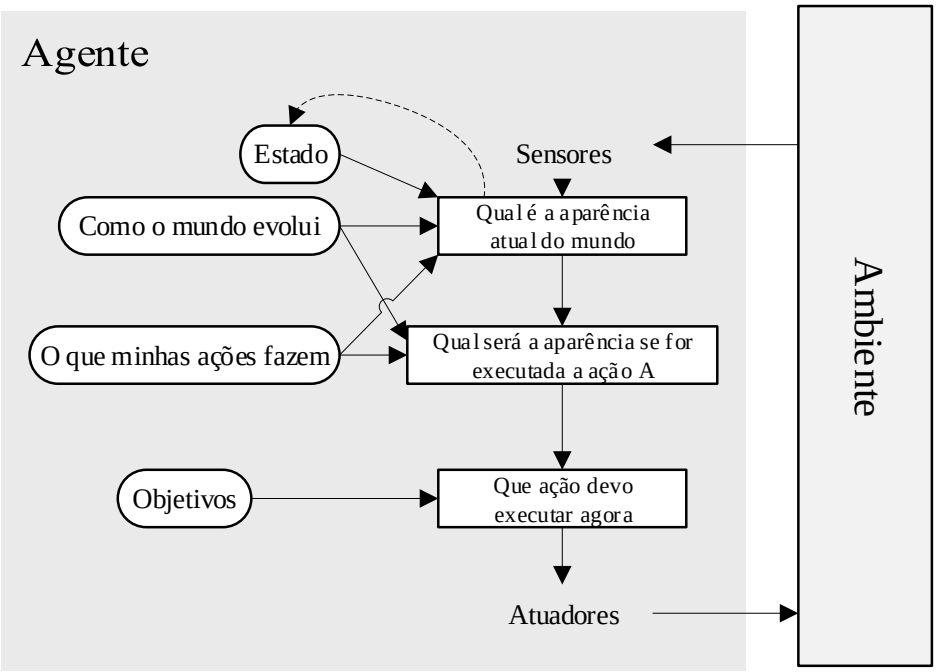


Figura 3 – Estrutura de uma agente baseado em objetivos.
Adaptado de (RUSSELL e NORVIG, 2013).

Por vezes, a seleção da ação utilizando objetivos é direta (ou seja, uma única ação resolve um objetivo), em outras, um objetivo não consegue ser atingido de imediato e é preciso considerar uma longa sequência de ações para alcança-lo. Busca e planejamento são subáreas da IA que se dedicam a encontrar sequências de ações para alcançar os objetivos do agente (RUSSELL e NORVIG, 2013).

O agente baseado em objetivos é mais flexível, comparado aos agentes reativos, pois o conhecimento no qual se apoia para decidir uma ação é representado explicitamente e pode ser modificado. Em situações que o ambiente se modifica durante a execução, o

agente baseado em objetivos pode atualizar os seus conhecimentos e todos os comportamentos relevantes ao conhecimento modificado serão alterados automaticamente. Já para o agente reativo, seria necessário reescrever diversas regras de condição-ação para abarcar a modificação do ambiente.

Para que um agente baseado em objetivos cumpra uma tarefa distinta, basta indicar essa tarefa nos seus objetivos. Um agente reativo é programado para cumprir apenas uma tarefa específica, caso queria que ele realize uma diferente, seria necessário substituir as regras de condição-ação para atender ao novo objetivo.

Um tipo de agente baseado em objetivos é o Agente baseado em conhecimento.

2.1.3. Agentes baseados em conhecimento

O agente baseado em conhecimento tem como componente central a sua base de conhecimento, ou KDB (*Knowledge Database*). Em geral, a KDB é um conjunto de sentenças, onde cada sentença representa um fato sobre o mundo. Quando uma sentença for representada sem ser derivada de outras sentenças, ela pode ser nomeada como axioma (RUSSELL e NORVIG, 2013).

Uma KDB precisa de meios para que seja inserida novas sentenças e para que elas possam ser consultadas. Para tal, utiliza-se operações que perguntam (*ASKs*) à KDB sobre as sentenças, e cujas respostas são baseadas nas sentenças que já foram informadas (*TELLed*) a ela. Os nomes padrões são ASK e TELL, respectivamente. Essas operações podem inferir novas sentenças a partir de antigas.

Um agente baseado em conhecimento é similar aos agentes que conhecem o estado atual do mundo, porém ele não é um programa arbitrário para calcular ações. Este agente consegue adaptar-se a uma descrição no nível de conhecimento, sendo necessário apenas especificar o que o agente sabe e quais as suas metas para corrigir o seu comportamento.

2.2. PLANEJAMENTO AUTOMÁTICO

Planejamento é o aspecto racional da ação. Para chegar em um objetivo pré-estabelecido, um agente precisa antecipar os efeitos que as suas ações terão no meio e deliberar quais as melhores ações a serem realizadas. Planejamento Automático é o

campo da IA que estuda esse processo de deliberação por meio de métodos computacionais (GHALLAB, *et al.*, 2004).

O uso do planejamento automático possui duas motivações básicas: A primeira, e mais prática, é criar ferramentas que forneçam planos factíveis e com uso econômico de recursos. A segunda, e mais teórica, é estudar os aspectos computacionais da inteligência para entender a característica racional da ação. O desafio não é somente estudar um processo abstrato isolado, mas os processos de deliberações envolvidos na escolha das ações como um sistema integrado.

Um agente que utiliza planejamento automático possui uma estrutura básica similar ao da Figura 4. A inteligência desse tipo de agente leva em consideração os objetivos a serem concluídos e as percepções do ambiente, para gerar planos de ações que serão convertidos em comandos para atuar no ambiente em questão.

Um agente puramente de planejamento considera apenas o estado inicial para gerar uma sequência de ações, porém um agente que tem integrado processos de replanejamento observa os sinais do ambiente e consulta as ações realizadas para averiguar se o plano está sendo concluído com êxito. Caso contrário, um novo plano é estabelecido com base no estado atual.

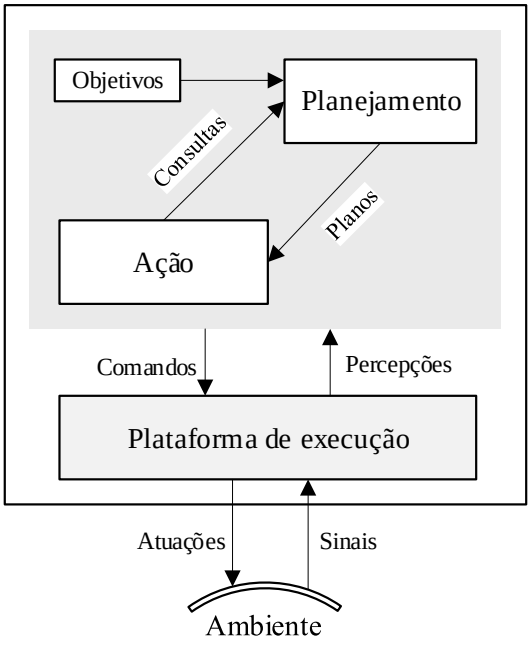


Figura 4 – Modelo de agente utilizando planejamento e ação.
Adaptado de (GHALLAB, *et al.*, 2016).

Um dos interesses do planejamento automático é utilizar abordagens genéricas e independentes do domínio. Uma abordagem dependente do domínio utiliza representações e técnicas adaptadas para o problema. O desenvolvedor prevê modelos para os tipos de ações que serão planejadas e para os estados do modelo em que elas terão efeito, utilizando, por exemplo, geometria, cinemática, e dinâmicas para planejamento de movimento ou manipuladores. O uso desta abordagem é bem justificada e tem um bom desempenho nas áreas específicas, porém o agente que utilizar este planejador terá a capacidade de deliberar limitada ao domínio em que o planejador é dependente (GHALLAB, *et al.*, 2004).

Quando um planejador independente do domínio vai resolver um determinado problema, ele recebe como entrada a especificação do mesmo e os conhecimentos sobre o domínio em que vai atuar, sendo que estes planejadores baseiam-se em modelos generalizados e abstratos de ações. O objetivo de utilizar uma abordagem independente do domínio é tornar a técnica de deliberação e raciocínio o mais abrangente e reutilizável possível. Essa metodologia não exclui o uso e pode ser utilizada em conjunto com ferramentas de planejamento dependentes de domínio (GHALLAB, *et al.*, 2004).

2.2.1. Planejamento clássico

Para utilizar a técnica de planejamento clássico, é preciso primeiro restringir o modelo à certas condições:

- Pressuposto 1: O domínio é finito.
- Pressuposto 2: O domínio é totalmente observável.
- Pressuposto 3: O domínio é determinístico.
- Pressuposto 4: Os objetivos são restritos.
- Pressuposto 5: O plano é uma sequência linear de ações.
- Pressuposto 6: As ações são discretas (não dependem ou representam o tempo).
- Pressuposto 7: O planejamento é realizado *offline*.

Considerando as condições acima, um domínio de planejamento pode ser representado por uma tripla $\Sigma = (S, A, \gamma)$, onde:

- $S = \{s_1, s_2, \cdots\}$ é um conjunto de estados;
- $A = \{a_1, a_2, \cdots\}$ é um conjunto de ações;

- $\gamma = S \times A \Rightarrow S$ é a função de transição de estados, sendo que:
  - $\gamma(s,a) = \left(s - \text{efeitos}^{-}(a)\right) \cup \text{efeitos}^{+}(a)$ se $a \in A$ for aplicável a $s \in S$, se não $\gamma(s,a)$ é indefinido.

A representação clássica é uma notação derivada da lógica de primeira ordem. Estados são representados como conjuntos de predicados que representam conhecimentos do mundo, e podem ser verdadeiros ou falso dependendo da interpretação. As ações são representadas por operadores de planejamentos que mudam a verdade desses conhecimentos (GHALLAB, *et al.*, 2004).

Para definir estados, operadores, ações, planos, problemas e soluções em planejamento clássico, será utilizado o problema do quebra-cabeças de oito peças como exemplo. Esse domínio de planejamento apresenta um tabuleiro onde existem nove espaços dispostos em uma matriz 3×3, e oito peças dispostas sobre esses espaços.

Estados

Estado é o conjunto de predicados que são verdadeiros em um determinado momento. Um predicado pode ser a representação de um conhecimento ou a sua negação. Um predicado e sua negação nunca podem ser verdadeiros em um mesmo estado.

Observando um estado do quebra-cabeças com a disposição da Figura 5, pode-se considerar as constantes do problema como os espaços, nomeados através de sua posição de linha e coluna (l1c1, l1c2, l1c3, l2c1, ..., l3c3), e as oito peças, nomeadas pelo seu nome (p1, p2, p3, …, p8). O conjunto de constantes resultante é então {l1c1, l1c2, l1c3, l2c1, ..., l3c3, p1, p2, p3, …, p8}.

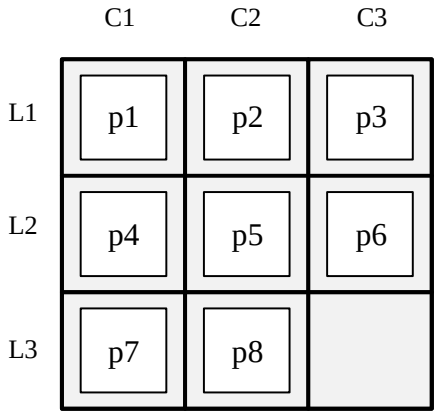


Figura 5 – Estado s0 do quebra cabeça.

Os conhecimentos relevantes para o estado são as posições que as peças ocupam no tabuleiro, qual espaço está vazio e as adjacências dos espaços. É possível representar esses conhecimentos como predicados em função de variáveis, ex., *pos(peça, espaço)*, *livre(espaço)* e *adj(espaço, espaço)*. A representação do estado *s0* do quebra-cabeças apresentado pode ser vista no Quadro 1.

```
s0={
   ;; Posição inicial dos números
      pos(p1, l1c1), pos(p2, l1c2), pos(p3, l1c3),
      pos(p4, l2c1), pos(p5, l2c2), pos(p6, l2c3),
      pos(p7, l3c1), pos(p8, l3c2), livre(l3c3),
   ;; Adjacência das colunas
      adj(l1c1, l1c2), adj(l1c2, l1c1), adj(l1c2, l1c3), adj(l1c3, l1c2),
      adj(l2c1, l2c2), adj(l2c2, l2c1), adj(l2c2, l2c3), adj(l2c3, l2c2),
      adj(l3c1, l3c2), adj(l3c2, l3c1), adj(l3c2, l3c3), adj(l3c3, l3c2),
   ;; Adjacência das linhas
      adj(l1c1, l2c1), adj(l2c1, l1c1), adj(l2c1, l3c1), adj(l3c1, l2c1),
      adj(l1c2, l2c2), adj(l2c2, l1c2), adj(l2c2, l3c2), adj(l3c2, l2c2),
      adj(l1c3, l2c3), adj(l2c3, l1c3), adj(l2c3, l3c3), adj(l3c3, l2c3)
}
```

Quadro 1 – Representação do estado s0 do quebra cabeça.

Operadores e ações

No planejamento clássico, um operador de planejamento é uma tripla $o = (nome(o), precondição(o), efeitos(o))$ cujos elementos são os seguintes:

- $nome(o)$, o nome do operador no formato de $n(x_1, \cdots, x_k)$, onde *n* é um símbolo único chamado de símbolo do operador, e $x_1, \cdots, x_k$ são todas as variáveis simbólicas que aparecem em qualquer lugar de *o*.
- $precondição(o)$, são os predicados que precisam ser verdadeiras para que a ação possa ser executada em um determinado estado.
- $efeitos(o)$, são os efeitos que a ação terá no sistema caso ela seja executada, os efeitos podem ser positivos ou negativos, adicionando ou removendo predicados de um estado.

Um exemplo de operador utilizado no quebra-cabeças de 8 peças é apresentado na Quadro 2.

```
mover(x, y, z)
   ;; Move peça x da posição y para a posição z
   precondição:   pos(x, y), adj(y, z), livre(z)
   efeitos:       pos(x, z), livre(y), ¬ pos(x, y), ¬ livre(z)
```

Quadro 2 – Exemplo do operador ‘mover’ do quebra cabeça.

Planos, Problemas e Soluções

Um problema de planejamento clássico é uma tripla $\mathrm{P} = (\Sigma, s_0, g)$, onde:

- s_0, é o estado inicial. O estado inicial pode ser qualquer estado de S;
- g, é o objetivo a ser alcançado. É um conjunto de predicados que precisam ser verdadeiros em um estado; e
- $S_g = \{s \in S \mid s \text{ satisfaz } g\}$.

A declaração de um problema de planejamento $\mathrm{P} = (\Sigma, s_0, g)$ é $\mathrm{P} = (\mathrm{O}, s_0, g)$, onde O é o conjunto de operadores do domínio Σ (GHALLAB, *et al.*, 2004).

Essa declaração é a especificação do problema que deve-se utilizar para, por exemplo, descrever *P* para um programa de computador.

Voltando para o exemplo do quebra-cabeças, considera-se o estado ilustrado na Figura 6 como o objetivo a ser alcançado.

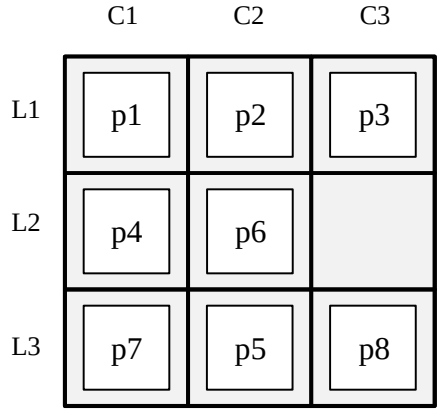


Figura 6 – Estado objetivo do quebra cabeça.

Ao declarar o objetivo, não é necessário informar as adjacências entre linhas e colunas, pois só é preciso garantir que as peças estejam nas posições especificadas. Considerando isso, o objetivo fica como o do Quadro 3.

```
g={
      pos(p1, l1c1), pos(p2, l1c2), pos(p3, l1c3),
      pos(p4, l2c1), pos(p6, l2c2), livre(l2c3),
      pos(p7, l3c1), pos(p5, l3c2), pos(p8, l3c3)
}
```
Quadro 3 – Representação do objetivo do quebra cabeça.

Com o estado inicial representado anteriormente, o objetivo definido e o conjunto de ações *O*, o problema de planejamento está totalmente definido. Ao informar esses parâmetros para o algoritmo de planejamento, ele irá procurar uma sequência de ações que, ao serem aplicadas, satisfaçam o objetivo estabelecido. Essa sequência de ações é denominada um plano.

Um plano π é solução para P se $\gamma(s_0, \pi)$ satisfaz g. No Quadro 4 têm-se dois planos, $\pi 1$ e $\pi 2$.

Um plano é considerado redundante se uma subsequência desse plano já satisfaz *g*, e é considerado mínimo se nenhuma solução possuir uma quantidade menor de ações. É possível visualizar que $\pi 2$ é redundante e que $\pi 1$ é o seu mínimo equivalente.

```
π1=<
      mover(p8, l3c2, l3c3)
      mover(p5, l2c2, l3c2)
      mover(p6, l2c3, l2c2)
>

π2=<
      mover(p8, l3c2, l3c3)
      mover(p5, l2c2, l3c2)
      mover(p6, l2c3, l2c2)
      mover(p3, l1c3, l2c3)
      mover(p3, l2c3, l1c3)
>
```

Quadro 4 – Representação dos planos π1 e π2 para o quebra-cabeças de oito peças.

2.2.1. FF Planner

O planejador *Fast-Forward*, abreviado como FF, obteve o melhor desempenho na quinta competição de sistemas de planejamento na AIPS'00 (*International Conference on Artificial Intelligence Planning and Scheduling*) (HOFFMANN, 2001).

A panorâmica da arquitetura do sistema é apresentada na Figura 7, onde é arranjado os princípios mais fundamentais do planejador FF.

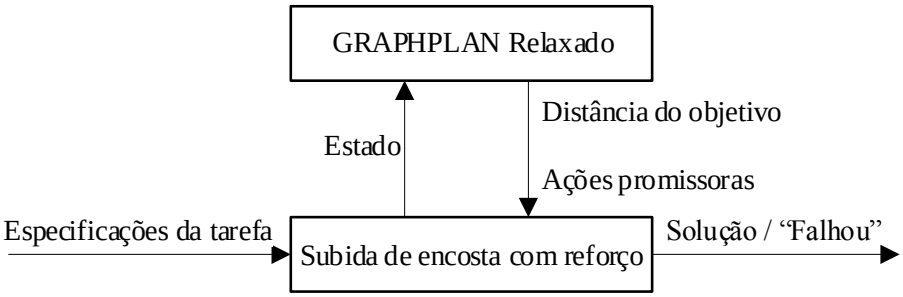


Figura 7 – Arquitetura do sistema base do planejador FF.
Adaptado de (HOFFMANN e NEBEL, 2001).

A heurística utilizada no planejador FF consiste de alguns passos:

Dado a especificação da tarefa, o planejador FF executa o algoritmo de Subida de Encosta com Reforço, que é o algoritmo de busca utilizado. Em cada estado que a subida de encosta com reforço encontra, ele aciona a técnica do GRAPHPLAN Relaxado. Essa técnica efetua um relaxamento do domínio para *Delete-Free* (ou DF), onde os efeitos de remoção de predicados dos operadores utilizados são ignorados, e executa o GRAPHPLAN.

GRAPHPLAN é um algoritmo utilizado em planejamento que considera o plano como se fosse um grafo com dois tipos de nós, os nós de fatos e os de ações, que se expandem em camadas intercaladas de fatos e ações até que todos os objetivos estejam presentes em uma camada e seja realizado uma busca de planejamento neste grafo (BLUM e FURST, 1997).

A resposta que o GRAPHPLAN relaxado fornece ao algoritmo de busca são as ações promissoras, que são ações com maior chance de alcançar os objetivos, e a distância até o objetivo, pois ao relaxar o domínio pode-se estimar uma distância do estado atual para um estado onde todos os objetivos tenham sido concluídos.

Obtendo a resposta do GRAPHPLAN relaxado, a busca por subida de encosta com reforço continua a busca em largura até encontrar um estado mais próximo do objetivo, caso não encontre um estado que se destaque, ele avança mais um estado e continua a busca. Ao encontrar um estado favorável, ele inicia uma nova busca tendo como estado inicial este estado encontrado. E, ao encontrar um estado que atenda aos objetivos, ele retorna à sequência de ações que levaram o estado inicial até ele.

Mais detalhes sobre esse planejador podem ser obtidos no trabalho desenvolvido por Hoffmann e Nebel (2001).

3. DESENVOLVIMENTO DO SIMULADOR

Neste capítulo será apresentado o ambiente do Labirinto do Minotauro, que serve de base para o estudo deste trabalho, realizado uma análise de suas propriedades e como o conhecimento deste ambiente foi modelado. Em sequência, o modo em que o ambiente da simulação foi desenvolvido é especificado, seguido pela metodologia utilizada para construir o agente inteligente e sua integração com a simulação.

3.1. AMBIENTE DO LABIRINTO DO MINOTAURO

O ambiente do Labirinto do Minotauro[16] teve como principal inspiração o Mundo do Wumpus, criado por Gregory Yob (1975) e apresentado como ambiente para teste de agentes por Russell e Norvig (2013).

No Labirinto do Minotauro, um agente está localizado em um local arbitrário de um labirinto, cujos espaços e suas vizinhanças são conhecidos. O agente possui somente uma bomba que explode em uma área de um espaço, além do local onde ela foi lançada. Dentro desse labirinto podem existir algumas chaves, que são necessárias para utilizar passagens secretas. O agente sabe que pode haver um minotauro dentro do labirinto, que incapacita o agente se ele entrar em contato, porém ele desconhece sua localização enquanto não perceber o seu cheiro, enquanto o minotauro tem conhecimento da localização do agente em todos os instantes. Um exemplo desse ambiente é apresentado na Figura 8.

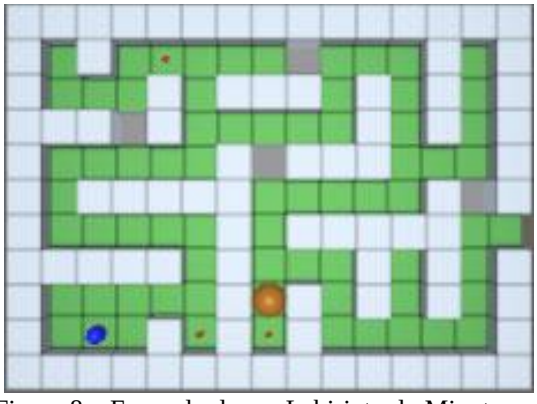


Figura 8 – Exemplo de um Labirinto do Minotauro.

[16] Disponível para download em: https://goo.gl/6MUriW (Link gerado em 4 de setembro de 2018)

A definição do ambiente da tarefa de acordo com a descrição de PEAS (*Performance*, *Environment*, *Actuators*, *Sensors*) utilizada por Russell e Norvig (2013) é dada a seguir:

Performance

A performance é dada pelo cumprimento ou não do objetivo do planejador, sendo que as ações possuem mesmo custo para serem realizadas. A minimização do plano foi deixada por conta do planejador utilizado.

Ambiente

O ambiente é composto de um labirinto em grade que, analogamente aos labirintos estudados por Funke *et al.* (2017), pode ser considerado uma matriz binária $n \times m$ onde os 0 são espaços acessíveis e os 1 são espaços bloqueados (ex., paredes).

Além do labirinto, é necessário que o ambiente contenha o agente, que será o detentor da inteligência artificial baseada em planejamento, e o minotauro, que possui um algoritmo de busca de Dijkstra para traçar uma rota pelo labirinto até o agente.

O ambiente pode possuir passagens secretas que são utilizáveis apenas pelo agente quando ele tiver posse de uma chave. Essas chaves podem estar sobre um dos espaços disponíveis para movimentação.

Atuadores

O agente pode usar a ação *moveTo* para andar para um espaço adjacente ao seu, desde que esse espaço não esteja ocupado pelo minotauro, pois isso resultaria em sua morte. A ação *pick* pode ser utilizada para recolher objetos que estejam no mesmo espaço em que o agente. Se o agente possuir uma chave, ele pode usar a ação *usePassage* para utilizar uma chave e liberar a passagem secreta. A ação *throwBomb* pode ser usada para lançar uma bomba em uma localidade a até dois espaços de distância do agente, sendo estes espaços ilustrados na Figura 9, essa ação só pode ser executada se o agente possuir uma bomba. A explosão da bomba tem a capacidade de matar o minotauro se o mesmo estiver na área de efeito da explosão, mas, se o agente estiver dentro da área, ele também é incapacitado. Uma vez morto, o minotauro e o agente não possuem a capacidade de retornar à vida.

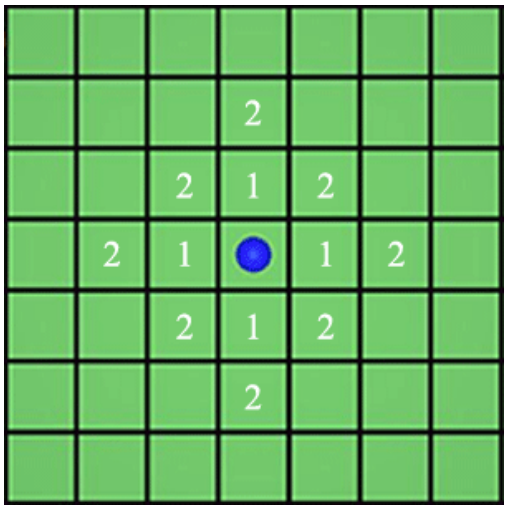


Figura 9 – Espaços onde o agente pode jogar a bomba.

Sensores

A área de percepção do agente pode ser configurada, sendo que tem como padrão três espaços de distância, sendo esta área ilustrada em azul na Figura 10. O agente tem somente uma percepção do ambiente, o *Cheiro* do minotauro, que alerta a presença do mesmo. O agente verifica constantemente a área de percepção, sendo que a mesma se move junto com o agente.

O minotauro possui uma área de cheiro que pode ser configurada, tendo como padrão um espaço de distância, isso é, somente o espaço em que ele está e os adjacentes possuem a informação de cheiro, esta área está ilustrada em laranja na Figura 10. Se o agente perceber um espaço que contenha esta informação, ele identifica a localização do minotauro.

Figura 10 – Área de percepção do agente e área de cheiro do minotauro.

3.2. ANÁLISE DO AMBIENTE

Nesta sessão, será realizado a análise das propriedades do ambiente de modo análogo ao apresentado por Russell e Norvig (2013). Em sequência são estruturados os conhecimentos relevantes que serão utilizados no ambiente, esta estrutura será a base para o domínio de planejamento utilizado.

3.2.1. Propriedades do ambiente de tarefas

Para decidir uma abordagem para o problema, primeiro se torna necessário analisar as suas propriedades. Esta análise será feita utilizando as definições feitas por Russell e Norvig (2013).

Totalmente observável x parcialmente observável

Um ambiente é dito totalmente observável se os sensores do agente dão acesso ao estado completo do ambiente em cada ponto do tempo. O labirinto do minotauro é um ambiente **parcialmente observável**, pois o agente tem o conhecimento da sua localidade, dos objetos dispostos pelo labirinto e as possíveis passagens existentes, porém o agente não tem conhecimento da localização do minotauro. Apenas quando o minotauro está próximo que o cheiro dele entra na área de percepção do agente e a localização dele é adquirida.

Agente único x multiagente

Apesar da distinção entre um ambiente possuir um único agente ou múltiplos agentes seja clara, é preciso avaliar quando uma entidade deve ou não ser considerado um agente. Se o comportamento de uma entidade B depende do comportamento do agente A para maximizar a sua performance, ele pode ser tratado como um agente por A.

Tendo isto em vista, pode-se considerar o labirinto do minotauro como um problema **multiagente**, já que o minotauro racionaliza em cima do ambiente, usando um método de resolução baseado em busca, com objetivo de encontrar menor caminho até o agente principal. Essa dinâmica torna o ambiente **competitivo**, pois os objetivos do agente e do minotauro são conflitantes, por impedirem que ambos sejam concluídos.

Determinístico x estocástico

Se o próximo estado do ambiente for completamente determinado pelo estado atual e pela ação realizada pelo agente, pode-se dizer que o ambiente é determinístico, caso contrário, ele é estocástico. Utilizando a definição de Russell e Norvig (2013), as incertezas que são provenientes unicamente de ações de outros agentes em um ambiente multiagente podem ser ignoradas. Pois um ambiente pode ser determinístico mesmo se cada agente não puder prever as ações do outro agente.

Segundo as definições apresentadas, o ambiente é considerado **determinístico**, tendo em vista que um estado futuro pode ser previsto usando apenas os efeitos das ações realizadas pelo agente como base, pois os efeitos das ações são conhecidos.

Episódico x sequencial

Em um ambiente episódico, a execução do agente é dividida em episódios atômicos. Em cada episódio o agente percebe o ambiente e, então, executa uma única ação correspondente às percepções e os episódios subsequentes não dependem das ações realizadas nos anteriores. Já em um ambiente sequencial, as ações realizadas anteriormente podem afetas as decisões futuras.

O problema em questão é considerado **sequencial**, mesmo o agente tomando decisões baseadas no estado atual do ambiente, ele precisa levar em consideração a sequência de ações que serão realizadas, pois o efeito de uma ação em um determinado momento pode influenciar na capacidade de realizar uma ação posteriormente.

Estático x dinâmico

Se o ambiente pode modificar-se enquanto o agente está deliberando, então o ambiente é dinâmico para o agente, caso contrário, ele é estático. Do modo em que está definido, o ambiente é **estático**, pois o ambiente do labirinto não sofre modificações com a passagem do tempo enquanto o agente delibera.

Discreto x contínuo

A distinção entre discreto e contínuo aplicam-se ao estado do ambiente, à maneira em que o tempo é tratado e às percepções e ações do agente. O ambiente é considerado

discreto, pois as ações realizadas pelo agente não sofrem influência do tempo e as percepções não carregam informações de estados anteriores, somente do atual. Os estados, assim como as ações, não têm influência do tempo e eles podem ser definidos apenas pela situação do ambiente atual.

Conhecido x desconhecido

Esta definição aplica-se ao conhecimento do agente sobre o ambiente e a sua dinâmica. Em um ambiente conhecido, o agente conhece todos os efeitos de suas ações no ambiente, no caso de um ambiente desconhecido, o agente precisa aprender qual a sua dinâmica antes de fazer boas decisões. Como sabe-se os resultados das ações possíveis de serem realizadas, o problema é dito como **conhecido**.

3.2.2. Definição do conhecimento

A fim de estruturar o ambiente para um agente baseado em conhecimento, é necessário identificar os conhecimentos relevantes e definir como eles serão representados.

As principais informações que o agente precisa ter para deliberar são: as localizações no labirinto, adjacência entre os espaços, posse de objetos, estados do agente e do minotauro e propriedades dos espaços.

As localizações serão representadas como espaços individuais, identificadas pela sua localização na matriz do labirinto, tendo como primeiro índice o 0 e início no canto inferior esquerdo e acrescendo até o canto superior direito. Um exemplo está demonstrado em (1):

$$\begin{array}{c} 3 \\ 2 \\ 1 \\ 0 \end{array} \begin{bmatrix} l3c0 & l3c1 & l3c2 & l3c3 \\ l2c0 & l2c1 & l2c2 & l2c3 \\ l1c0 & l1c1 & l1c2 & l1c3 \\ l0c0 & l0c1 & l0c2 & l0c3 \end{bmatrix} \quad \begin{array}{c} \\ \\ \\ \\ 0 \quad 1 \quad 2 \quad 3 \end{array} \tag{1}$$

A informação de que uma localidade está bloqueada ou não é indicada pela presença ou não da representação do espaço. Por exemplo, alguns espaços de (1) foram bloqueados a fim de deixar somente os espaços que formam o caminho presente em (2):

$$\begin{array}{c} 3 \\ 2 \\ 1 \\ 0 \end{array} \begin{bmatrix} - & - & - & l3c3 \\ - & - & - & l2c3 \\ l1c0 & l1c1 & l1c2 & l1c3 \\ l0c0 & - & - & - \end{bmatrix} \\ \quad\; 0 \quad 1 \quad 2 \quad 3 \tag{2}$$

Para representar as adjacências foi adotado o conceito de vizinhança, utilizando o predicado *neighbour(tile1, tile2)*, como um grafo orientado, isso é, indicando que o espaço *l0c0* é vizinho de *l1c0* (*neighbour(l0c0, l1c0)*), é possível mover o agente entre esses espaços, porém, se não for indicado que o espaço *l1c0* é vizinho de *l0c0* (*neighbour(l1c0, l0c0)*), o caminho inverso não é possível. Esse conceito de vizinhança abre possibilidades de criar situações como passagem secretas (ex., usando o predicado *passage(tile1, tile2)* para ligar os espaços vizinhos de uma parede), teletransportes ou similares, caso desejado, bastando indicar que um espaço é vizinho do outro, mesmo que não seja fisicamente.

O agente e o minotauro tem sua localização no labirinto representada pelo predicado *atTile(agent, tile)*, que simboliza que o *agent* (minotauro ou o agente, propriamente dito) está presente no espaço *tile*. A localização de objetos no mapa é feita de modo similar, utilizando o predicado *objectOnMap(object, tile)*.

Levando em consideração a configuração definida para o ambiente, somente o agente pode possuir objetos. A representação do conhecimento que o agente tem posse de um determinado objeto é dada pelo predicado *haveObject(object)*.

O agente e o minotauro possuem uma propriedade que indica se eles estão, ou não, vivos. Para representar esse conhecimento foi utilizado o predicado *isAlive(agent)*, onde *agent* pode ser o agente, propriamente dito, ou o minotauro.

Os espaços possuem duas propriedades auxiliares, uma indicando que o agente já passou pelo espaço (*visited(tile)*), e outra indicando que o espaço é seguro, isso é, que não possui o *cheiro* do minotauro (*tileSafe(tile)*).

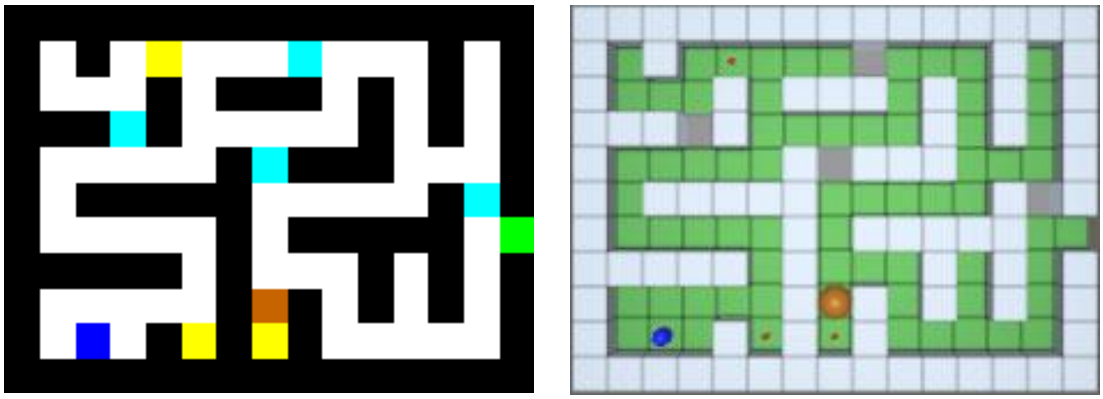
3.3. DESENVOLVIMENTO DA CENA DO JOGO

O ambiente de simulação foi desenvolvido utilizando o Unity3d[17], que é um *game engine* comumente utilizada para desenvolvimento de jogos e aplicativos para diversas plataformas.

Para criar a cena inicial, o programa faz a leitura dos pixels de uma imagem e cria os objetos do ambiente baseado na posição do pixel e na cor que ele possui, podem existir os seguintes elementos de cena:

- Parede – representado pela cor preta;
- Espaço – representado pela cor branca;
- Saída do labirinto – representada pela cor verde;
- Chave – representada pela cor amarela, a chave sempre é criada sobre um espaço;
- Passagem – representada pela cor ciano, a passagem é considerada uma parede e seu funcionamento será explicado posteriormente;
- Agente – representado pela cor azul, é o local onde o agente é criado inicialmente, o agente sempre é criado sobre um espaço;
- Minotauro – representado pela cor laranja, assim como o agente, é o local onde o minotauro é criado inicialmente, e o mesmo sempre é criado sobre um espaço.

Um exemplo está ilustrado na Figura 11.



(a) (b)

Figura 11 – (a) Mapa de pixels utilizado pelo programa, (b) Ambiente criado baseado no mapa de pixels.

[17] Disponível em: https://unity3d.com/pt/ (Acesso em 6 de junho de 2018)

Em conjunto à criação do labirinto, é criado um grafo contendo os espaços como vértices e as arestas interligando os espaços adjacentes, conforme ilustra a Figura 12. Esse grafo é utilizado para informar conhecimentos relacionados à vizinhança para a base de conhecimento e como base para movimentação do agente e do minotauro.

Para realizar o movimento do minotauro, foi implementado um algoritmo de busca de Dijkstra utilizando o grafo como espaço de busca, tendo como objetivo fixo encontrar a menor rota para o agente.

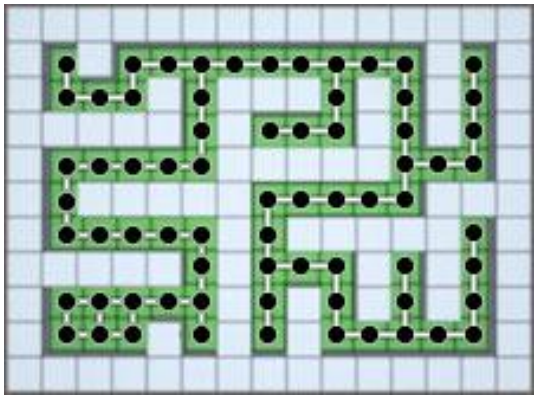


Figura 12 – Grafo criado com as vizinhanças dos espaços do labirinto.

Para utilizar uma passagem do labirinto, o programa da simulação precisa realizar algumas etapas, sendo elas representadas na Figura 13:

a. Anteriormente à utilização da passagem, o agente precisa recolher uma das chaves presentes no labirinto. Essas chaves são representadas como esferas vermelhas;

b. Uma passagem pode ser atravessada em qualquer sentido, mas para isso o agente precisa estar em uma de suas vizinhanças;

c. Ao indicar para a simulação a intenção de utilizar a passagem, e que o agente possui a chave, o grafo cria uma aresta entre os espaços vizinhos, possibilitando a movimentação do agente entre esses espaços;

d. Com essa aresta criada, o agente move-se para o espaço localizado no outro estremo da passagem;

e. Finalizando o movimento, o agente descarta uma chave, pois seu uso é único, e o grafo remove a aresta interligando os espaços. Indicando que a passagem está intransponível até que o agente possua uma nova chave e requeira o seu uso.

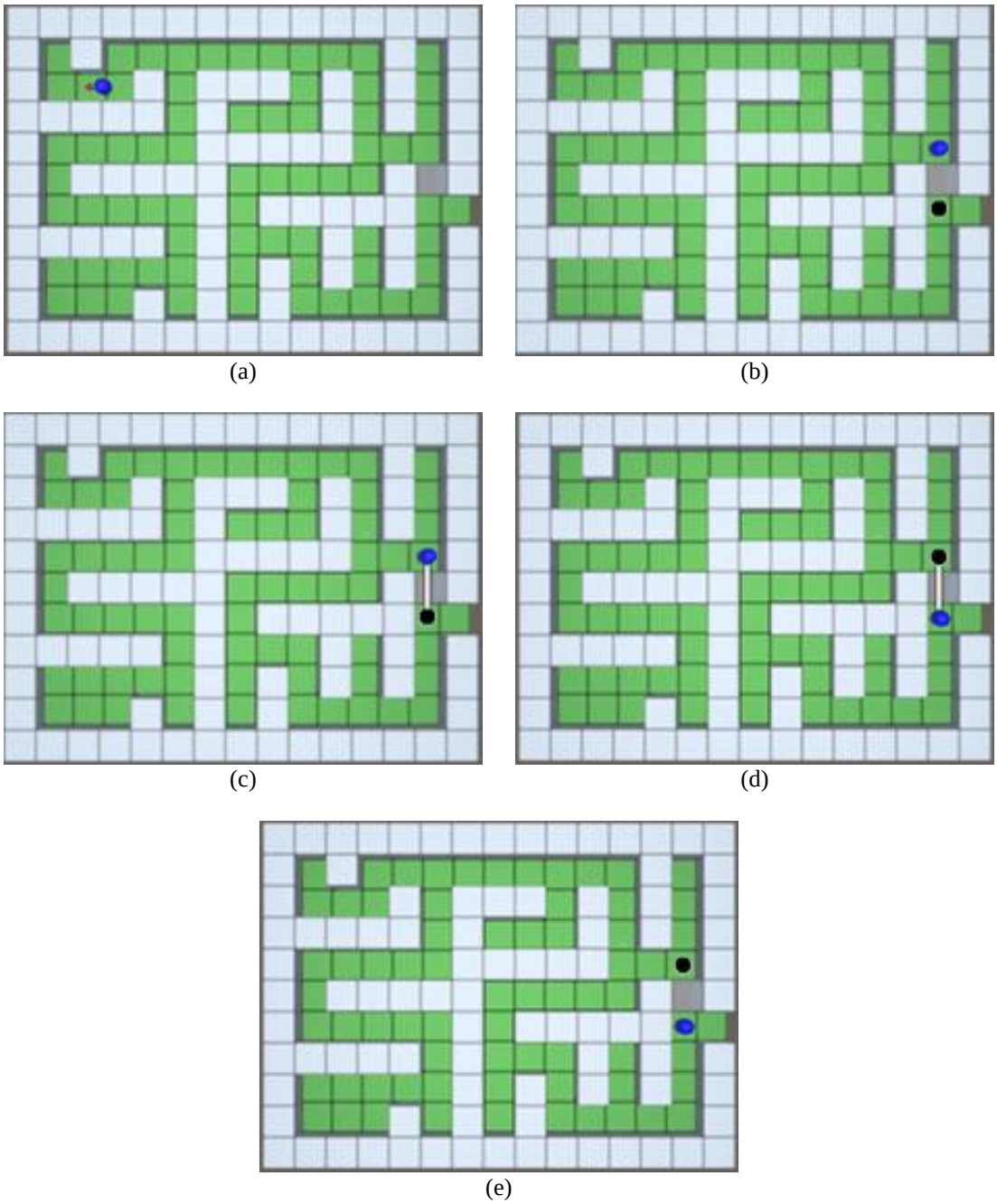
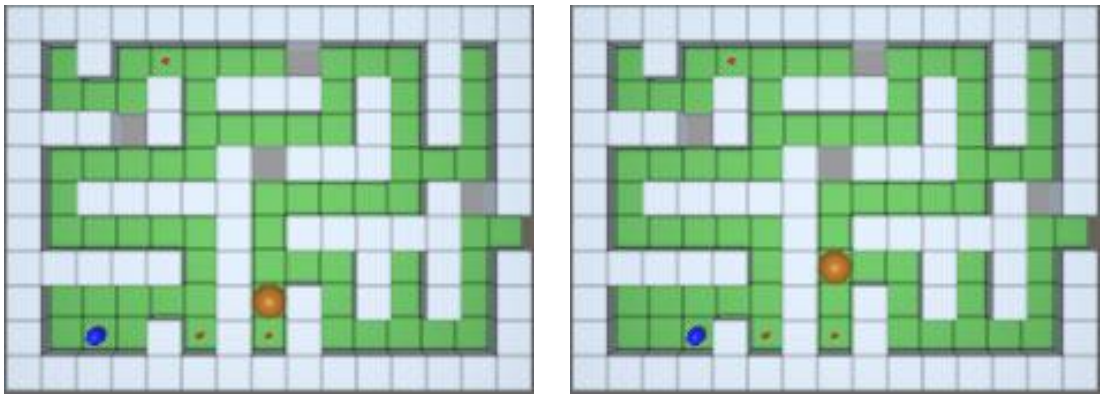


Figura 13 – Passos para utilizar uma passagem. (a) Possuir uma chave, (b) estar na adjacência da passagem, (c) criar uma aresta no grafo, (d) mover para o outro lado e (e) remover a aresta do grafo.

A execução da simulação é realizada em turnos com ações simultâneas. Isso é, em um mesmo instante, o agente e o minotauro realizam uma ação. Como exemplo, na Figura 14, o agente está inicialmente localizado no espaço *l1c2*, e o minotauro no espaço *l2c7*. Ao realizar a ação *moveTo(l1c3)*, o agente move para o espaço designado e o minotauro simultaneamente move-se para o espaço *l3c7*, pois é o próximo passo da rota que a busca de Dijkstra traçou para ele.



(a) (b)

Figura 14 – (a) No instante t1, o agente está no espaço *l1c2* e o minotauro no espaço *l2c7*. (b) No instante t2, o agente foi para o espaço *l1c3* e o minotauro para o *l3c7* simultaneamente.

No início do problema o agente possui uma bomba para ser usada como defesa. O agente pode lançar ela a até dois espaços de distância da sua localidade, e a bomba tem uma área de explosão de um espaço, além do local aonde foi lançada. No exemplo da Figura 15, a bomba foi lançada no espaço indicado pela letra 'B' e sua área de explosão foi realçada. Como o minotauro encontrava-se dentro da área, ele foi destruído, porém, se o agente fosse atingido pela explosão, ele também seria incapacitado.

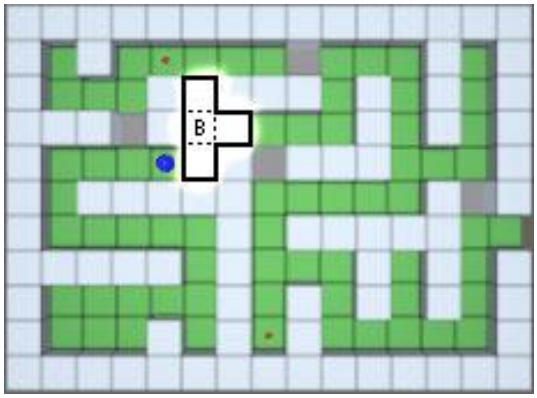


Figura 15 – Efeito da bomba no labirinto.

3.4. DESENVOLVIMENTO DA INTELIGÊNCIA ARTIFICIAL DO AGENTE

O agente desenvolvido tem como princípio básico os agentes racionais descritos por Russell e Norvig (2013), em especial os agentes baseados em objetivos e conhecimentos.

3.4.1. Integração Simulação/Planejador

O programa é composto por estruturas que interagem entre si durante a execução do programa. Essas estruturas são: O labirinto, propriamente dito, o agente, a base de conhecimento (que será nomeado apenas de KDB – *Knowledge DataBase*) e o planejador. O labirinto, o agente e a KDB foram codificados dentro da simulação, enquanto o planejador automático é executado externamente à simulação. Suas interações estão ilustradas na Figura 16.

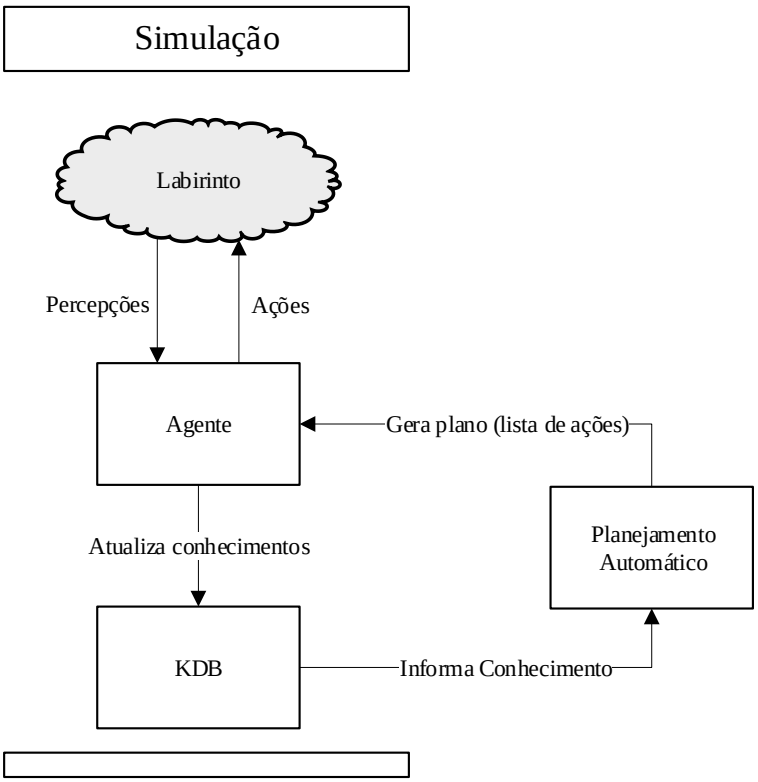


Figura 16 – Interações entre as estruturas que compõem o programa.

O labirinto já foi descrito na sessão 3.1 Ambiente do Labirinto do Minotauro, e tem como objetivo principal ser o ambiente de trabalho do agente.

O agente é o detentor da habilidade de interagir com o ambiente, percebendo os seus arredores e podendo alterá-lo ao realizar ações, ex., ao realizar a ação *pick(agent, key, l3c4)*, o agente retira uma chave do ambiente, alterando o estado atual em relação ao estado do início da simulação. Com as percepções adquiridas, o agente verifica incoerências nos conhecimentos presentes na KDB e os altera caso necessário.

A KDB será explicada em detalhes no tópico 3.4.2 Base de conhecimento, e seu papel é armazenar todo o conhecimento que o agente possui sobre o ambiente da simulação e fornecê-los para o planejador quando requisitado.

O planejamento automático é realizado externamente ao programa da simulação, podendo ele utilizar qualquer planejador que tenha como entradas um arquivo de domínio e outro de problema, ambos em PDDL (*Planning Domain Definition Language*), e forneçam um arquivo com os passos de execução do plano para ser lido pela simulação. O domínio desenvolvido para este trabalho está disponível na íntegra no APÊNDICE A.

O PDDL, surgiu de uma necessidade de generalizar as linguagens de planejamento. Foi criada com objetivo primário de tornar possível a realização da Competição Internacional de Planejamento (IPC) de 1998/2000, sua primeira definição foi apresentada por McDermott *et al.* (1998). Desde então esta linguagem foi sendo aperfeiçoada e encontra-se na versão PDDL3.0, que tem sua definição atualizada por Gerevini e Long (2005) e é a versão utilizada neste trabalho.

3.4.2. Base de conhecimento (KDB)

A base de conhecimento (KDB) utilizada foi adaptada do código presente no C# Unity3D HTN-Planner (CUHP)[18], um planejador criado por Pieterjan van Gastel. O CUHP possui uma estrutura que tem como papel armazenar estados do ambiente para que seja realizado uma busca. Na simulação essa estrutura é utilizada como a KDB, tendo o papel de armazenar o conhecimento que o agente possui sobre o ambiente, como as vizinhanças dos espaços, sua localidade, a localização das chaves, passagens do labirinto e a localidade do minotauro (somente ao ter a percepção do *cheiro* do mesmo).

A KDB é formada por dois dicionários que representam os conhecimentos como predicados de primeira ordem, ex., *haveObject(key)* e *neighbour(l3c4, l2c4)*. Essa estrutura de armazenamento dos conhecimentos pode ser melhor visualizada na Figura 17.

Um predicado pode possuir vários valores associados a ele, ex., *haveObject(key)* e *haveObject(bomb)*. Caso o predicado possua uma relação de valores, um primeiro valor

[18] Disponível em: http://pjvgastel-programming.blogspot.com.br/ (Acesso em 11 de maio de 2018)

pode associar-se a diversos valores secundários, ex., *neighbour(l1c1, l1c2)* e *neighbour(l1c1, l2c1)*. Cada interconexão entre predicado e valores representa um conhecimento do agente.

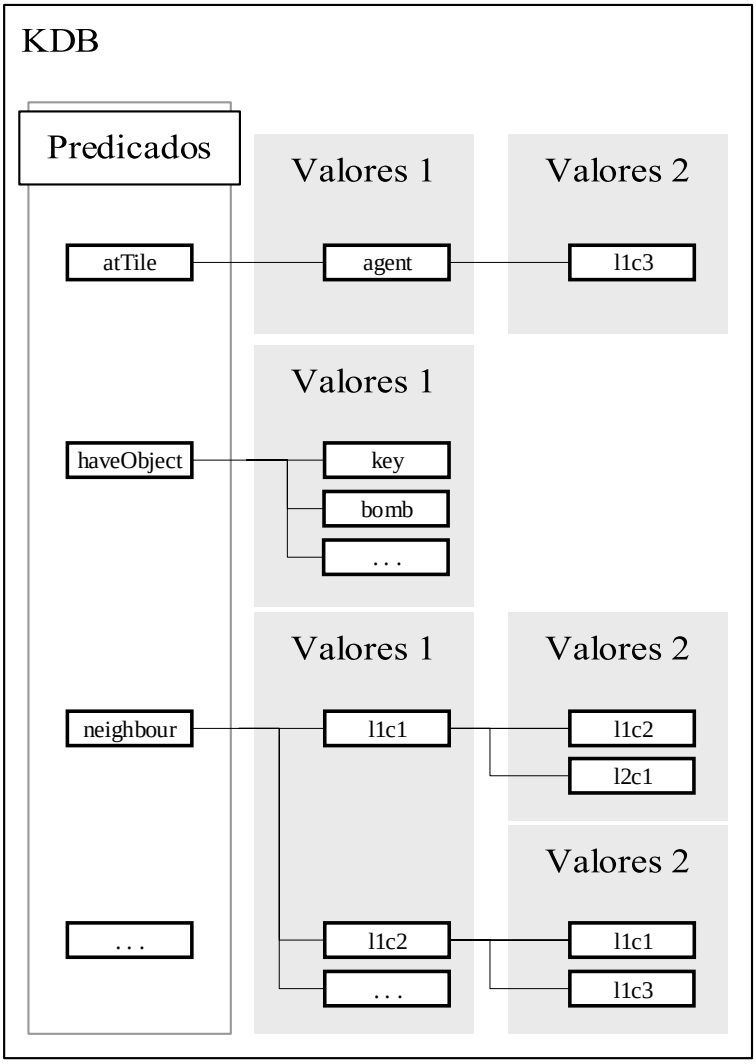


Figura 17 – Base de conhecimento do agente.

Durante a simulação, a KDB é alterada em tempo de execução, refletindo as alterações de conhecimento do agente em relação ao ambiente diante a cada ação realizada. Essas alterações refletem a busca de estados realizada pelo planejador em cada passo dado, por exemplo:

Em um determinando instante, o agente precisa de uma chave para dar continuidade ao plano, e ele possui os seguintes conhecimentos (dentre outros não ilustrados):

atTile(agent, l3c4) e *objectOnMap(key, l3c4)*

Para recolher a chave, o planejador indica a ação *pick(agent, key, l3c4)* para ser realizada. A ação *pick* está descrita no Quadro 5:

```
(:action pick
  ;; player pega o objeto que está no espaço location
  :parameters    (?player - player ?object - object ?location - tile)
  :precondition  (and  (atTile ?player ?location)
                       (objectOnMap ?object ?location)
                 )
  :effect        (and  (not (objectOnMap ?object ?location))
                       (haveObject ?object)
                 )
)
```

Quadro 5 – Ação de pegar objeto, codificado em PDDL.

Antes de executar a ação, a simulação precisa verificar se as precondições foram todas satisfeitas, e após realiza-la, a KDB precisa atualizar seus conhecimentos para corresponder aos seus efeitos. Para que isso seja possível, a KDB disponibiliza funções que permitem consultar (ex., *CheckVar(predicado, valor)*) e atualizar os conhecimento (ex., *Add(predicado, valor)* e *Remove(predicado, valor)*).

Portanto, após executar a ação *pick(agent, key, l3c4)*, são utilizadas as funções *Remove(objectOnMap, key, l3c4)* e *Add(haveObject, key)*, para remover o conhecimento antigo e adicionar a informação *haveObject(key).*

Outras funções disponibilizadas pela KDB são as seguintes:

- *Add(predicado, valor)* ou *Add(predicado, valor1, valor2)*: adiciona um novo conhecimento à KDB;
- *Remove(predicado, valor)* ou *Remove(predicado, valor1, valor2)*: remove um conhecimento da KDB;
- *GetStateOfVar(predicado)* ou *GetStateOfRelation(predicado)*: retorna todos os valores associados ao predicado;
- *ContainsVar(predicado)* ou *ContainsRelation(predicado)*: retorna verdadeiro se existir algum valor para o predicado, falso se não tiver nenhum valor associado;
- *CheckVar(predicado, valor)* ou *CheckRelation(predicado, valor1, valor2)*: verifica se existe o valor, ou valores, para o predicado e retorna verdadeiro ou falso;

3.4.3. Execução da simulação

O processo de execução da simulação está representado no fluxograma da Figura 18 e será explicado em detalhes a seguir.

A simulação inicialmente cria o *layout* físico com os objetos da cena e em seguida ele instancia os agentes da simulação. Com o cenário totalmente criado, o ambiente de simulação adiciona todo o conhecimento relevante ao planejador na KDB, como as vizinhanças dos espaços em que os agentes podem mover, a posição inicial do agente e dos objetos que existem na cena, dentre outros.

Em sequência, o ambiente de simulação escreve o problema inicial a ser resolvido pelo planejador em PDDL, incluindo os objetivos escolhidos pelo usuário e todo o conhecimento disponibilizado pela KDB, e executa o algoritmo de planejamento externamente ao ambiente de simulação, que proverá um arquivo de plano com os passos necessários para que o objetivo seja alcançado pelo agente que o processará.

Com a finalização do algoritmo de planejamento, uma janela de comando indicará se um plano que satisfaça o problema foi encontrado ou não. Caso possua um plano, o ambiente de simulação realiza a leitura do arquivo de plano e coloca as ações em uma fila para serem realizadas. Caso não possua um plano válido, o ambiente de simulação deverá ser executado novamente com outras condições iniciais.

Antes de executar uma ação, o agente realiza duas verificações: A primeira, se a área de percepção está como o esperado, e a segunda, se pode realizar a ação designada. Ao verificar a área de percepção, o agente analisa se o minotauro está presente ou não em um dos espaços da área, e compara com o seu conhecimento. Caso tenha uma inconsistência de informações (ex., a KDB indica que o minotauro não está perto, mas a percepção indica que ele está, ou vice-versa; ou a KDB não tem conhecimento se o minotauro está vivo, mas verifica se ele está, ou vice-versa), é efetuado o processo de replanejamento, caso esteja como o esperado, o agente segue com a segunda verificação.

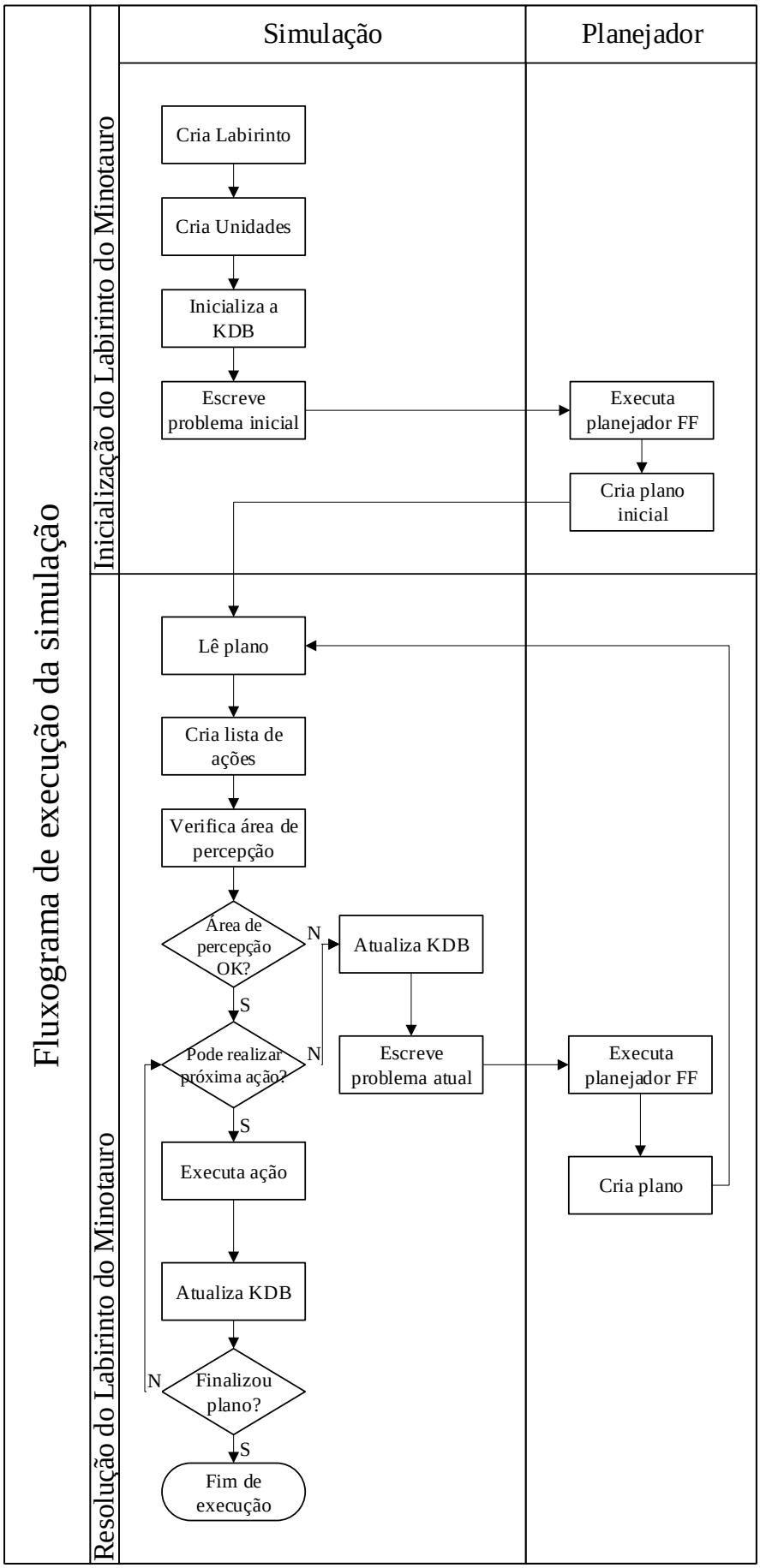


Figura 18 – Fluxograma de execução da simulação.

Para averiguar se as pré-condições para realizar a ação foram satisfeitas, são comparados os conhecimentos que correspondem às precondições presentes na KDB com as suas propriedades correspondentes, fornecidas pela simulação. Caso as informações sejam coerentes, a ação é realizada, caso contrário, a ação é cancelada e o processo de replanejamento é realizado.

Quando o processo de replanejamento é requisitado, a simulação atualiza a KDB com as informações do estado atual, ou seja, posição atual do agente, posição atual do minotauro, caso conhecida, posse de objetos pelo agente e objetos retirados do mapa. Com a KDB atualizada, a simulação escreve um novo arquivo de problema e executa novamente o algoritmo de planejamento. Caso o novo problema tenha solução, a simulação realiza a leitura do novo plano e cria uma nova lista de ações. Novamente, se não for encontrado um plano válido, será necessário reiniciar o ambiente de simulação com configurações iniciais diferentes.

Após uma ação ser realizada, a KDB atualiza os seus conhecimentos com os efeitos esperados da ação e verifica se o plano foi concluído. Caso ainda tenha ações na fila, a simulação volta para a etapa de verificação da ação, se não ele dá o plano como finalizado e a simulação é encerrada com um aviso de término.

Em situações em que o agente é incapacitado pelo minotauro ou pela explosão da bomba, a simulação é encerrada previamente com um aviso que o plano falhou porque o agente morreu.

4. ANÁLISES E RESULTADOS

Neste capítulo serão analisados alguns casos pertinentes utilizando o Labirinto do Minotauro, evidenciando suas peculiaridades, e é apresentado o simulador desenvolvido como ferramenta de ensino a ser utilizado futuramente em sala de aula.

4.1. ANÁLISES DE CASOS

Caso 1: Erro de modelagem acarretando em morte do agente

Nesse caso, têm-se um labirinto com o layout indicado na Figura 19, na qual o minotauro foi deliberadamente ocultado, com a posição inicial do agente no espaço *l3c2* e saída na *l0c2*. O objetivo do agente é somente sair do labirinto.

Inicialmente o planejador gera o plano indicado em (3) :

0: moveTo(player, l3c2, l2c2, minotaur)
1: moveTo(player, l2c2, l1c2, minotaur) (3)
2: moveTo(player, l1c2, exit, minotaur)

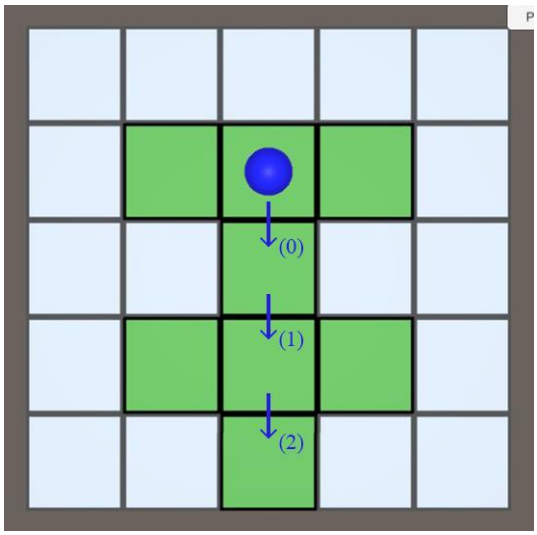
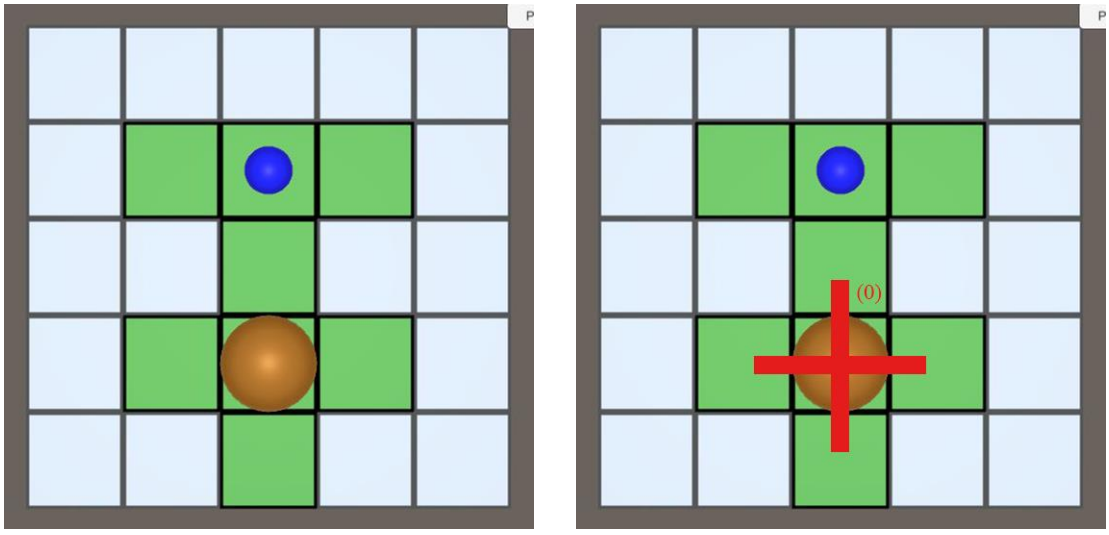


Figura 19 – Caso 1 com passos do plano inicial.

Devido ao modo em que a ação *throwBomb* foi modelada, o efeito da ação indica que o minotauro será morto, mas não prevê que o agente também será incapacitado caso esteja dentro da área de explosão. Analisando duas posições iniciais distintas do minotauro, obteve-se dois comportamentos diferentes durante o replanejamento por causa deste equívoco.

No caso da Figura 20, o minotauro inicialmente está no espaço *l1c2*, e o plano gerado após o replanejamento é apresentado em (4). Neste plano, o agente elimina o minotauro, jogando a bomba no espaço *l1c2* e continua executando o plano inicial.

0: throwBomb(l1c2, bomb, minotaur, player, l3c2)
1: moveTo(player, l3c2, l2c2, minotaur)
2: moveTo(player, l2c2, l1c2, minotaur)
3: moveTo(player, l1c2, exit, minotaur) (4)

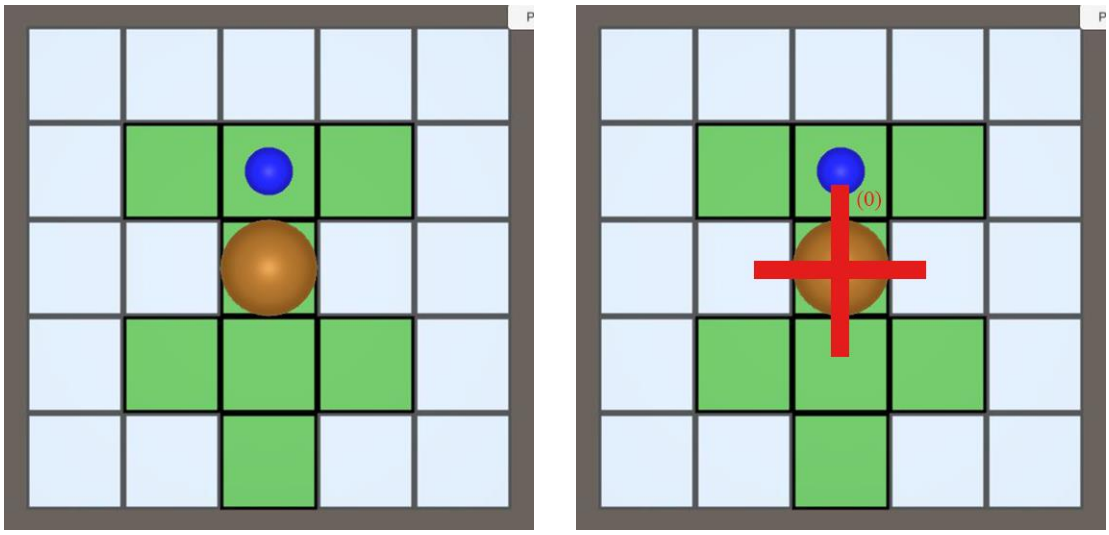


(a) (b)

Figura 20 – Caso 1a: (a) Estado de replanejamento, (b) Área de efeito da explosão.

Ao posicionar o minotauro no espaço *l2c2*, conforme mostra a Figura 21, o planejador gera o plano indicado em (5). Ao executar esse plano, o agente lança a bomba no espaço l2c2, fazendo com que ele fique dentro da área de efeito da explosão da bomba causando, assim, a sua morte.

0: throwBomb(l3c2, bomb, minotaur, player, l3c2)
1: moveTo(player, l3c2, l2c2, minotaur)
2: moveTo(player, l2c2, l1c2, minotaur)
3: moveTo(player, l1c2, exit, minotaur) (5)



(a) (b)

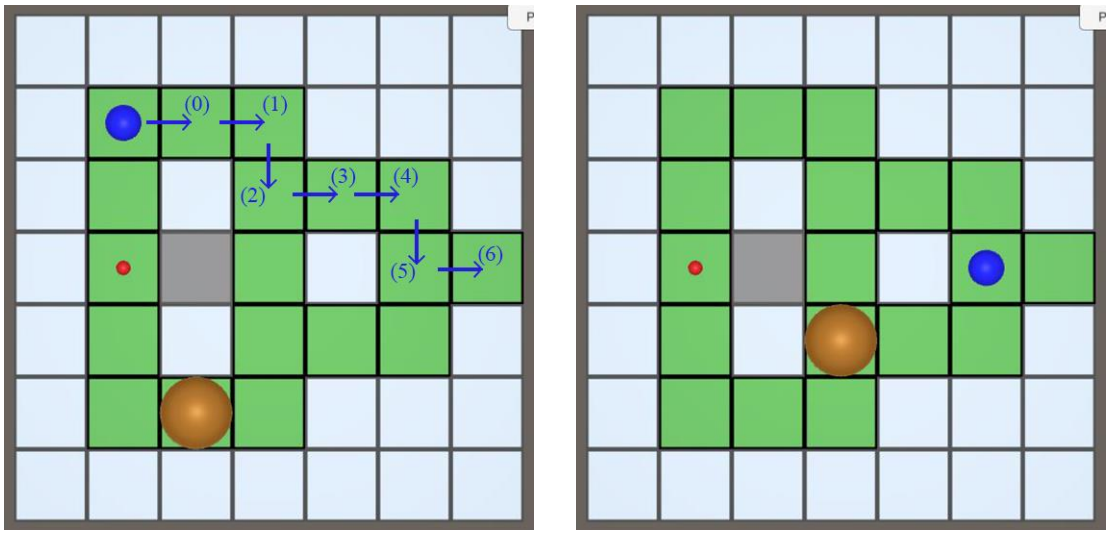
Figura 21 – Caso 1b: (a) Estado de replanejamento, (b) Área de efeito da explosão.

Esta situação exemplifica que a inteligência do agente depende do modo em que o domínio é modelado, afetando, assim, nos planos que o planejador fornece. Neste caso, o equívoco da modelagem torna aplicável o plano gerado em (5), sendo que o plano (4) atenderia aos objetivos de maneira mais segura.

Caso 2: Objetivo altera comportamento do agente

Nesse caso, o estado e *layout* do labirinto estão inicialmente como o ilustrado na Figura 22, sendo a posição inicial do agente o espaço *l5c1* e a saída em *l3c6*, e foi gerado o plano (6). Ao executar o passo 5 do plano, entrou-se no processo de replanejamento.

0: moveTo(player, l5c1, l5c2, minotaur)
1: moveTo(player, l5c2, l5c3, minotaur)
2: moveTo(player, l5c3, l4c3, minotaur)
3: moveTo(player, l4c3, l4c4, minotaur) (6)
4: moveTo(player, l4c4, l4c5, minotaur)
5: moveTo(player, l4c5, l3c5, minotaur)
6: moveTo(player, l3c5, exit, minotaur)



(a) (b)

Figura 22 – Caso 2: (a) Passos do plano inicial, (b) Estado de replanejamento.

Foram analisadas duas situações neste ponto: a primeira, caso o objetivo do agente fosse apenas sair do labirinto, a outra, caso o objetivo fosse sair do labirinto e garantir que o minotauro estivesse morto. Para a segunda situação, o objetivo *(not (isAlive minotaur))* foi acrescentado no arquivo de problema. Este objetivo só tem efeito quando o agente percebe o cheiro do minotauro, pois, até então, o agente não possui a informação de que o minotauro está vivo, considerando assim, este objetivo como concluído.

Na primeira situação, após o replanejamento, o agente ignora a ameaça do minotauro e segue para a saída, gerando o plano indicado em (7), que é o passo 6 do plano (6), e finalizando o seu objetivo, conforme mostra a Figura 23.

0: moveTo(player, l3c5, exit, minotaur) (7)

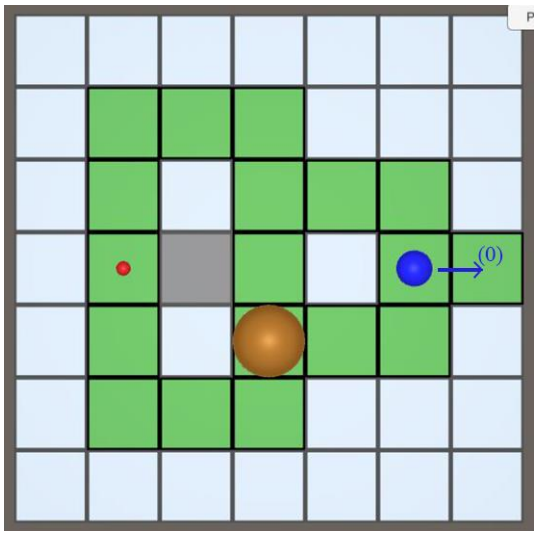


Figura 23 – Caso 2a: Agente ignorando o minotauro.

Na segunda situação, o agente precisa garantir que o minotauro está morto caso ele o perceba. Devido a essa situação, o planejador gera o plano (8) e o agente executa, jogando a bomba antes de sair do labirinto, conforme mostra a Figura 24.

0: throwBomb(l2c4, bomb, minotaur, player, l3c5)
1: moveTo(player, l3c5, exit, minotaur) (8)

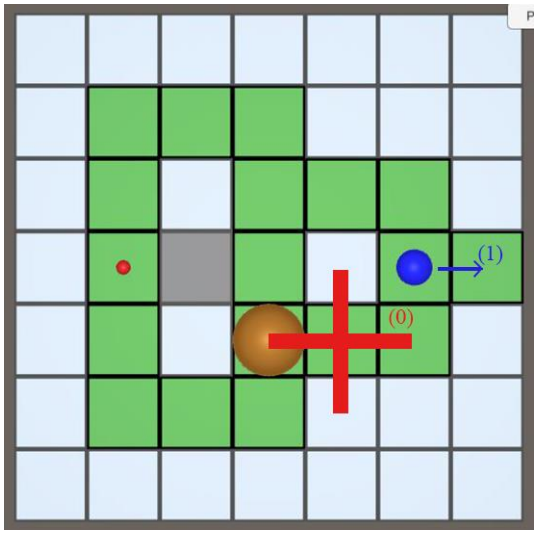


Figura 24 – Caso 2b: Agente eliminando o minotauro antes de sair do labirinto.

Este caso exemplifica a importância de o planejador possuir objetivos claros ao ser executado. Em situações similares, quando não explicitado que o minotauro deveria ser morto, o agente continuava com o mesmo plano enquanto o minotauro não impedisse o agente de executar a ação (ex., se o minotauro não impedisse a passagem do agente por um corredor).

4.2. COMO FERRAMENTA DE ENSINO

O simulador foi desenvolvido com objetivo de ser utilizada futuramente em sala de aula como ferramenta de ensino. A sua interface de utilização é apresentada a seguir.

4.2.1. Interface de utilização

Para utilizar o ambiente de simulação o usuário precisa inicialmente colocar nos espaços indicados em 1, na Figura 25, o número de Linhas e de Colunas desejados para o labirinto. Depois pressione o botão Editar Labirinto, indicado como número 2, e uma grade com as dimensões do labirinto é criada à direita.

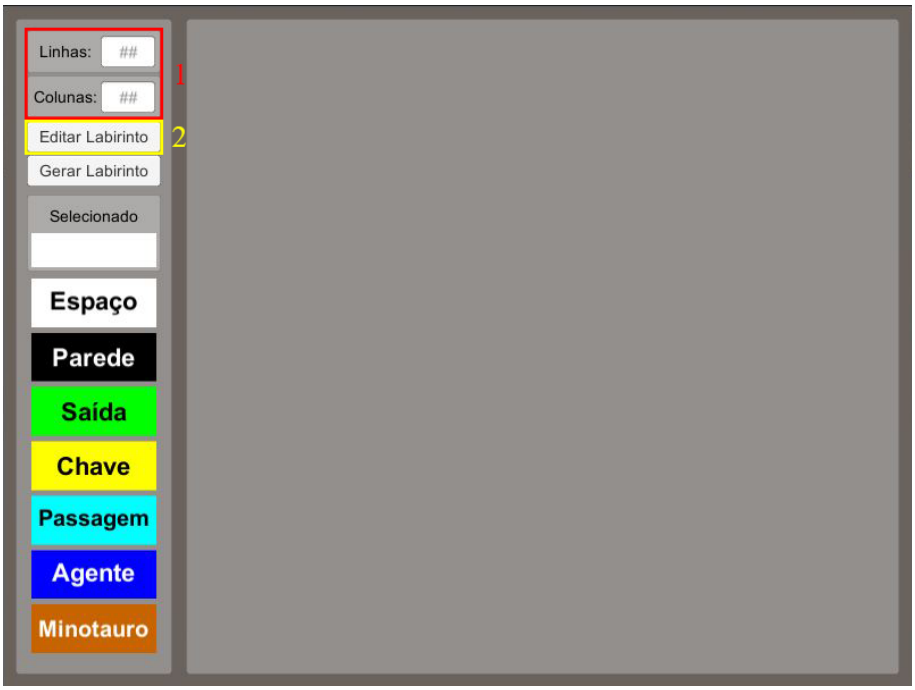


Figura 25 – Interface inicial.

Para editar o labirinto, basta clicar em uma das opções indicadas na Figura 26 como número 1, a seleção será confirmada com sua cor sendo apresentada no retângulo indicado como número 2. Com uma opção selecionada, pode-se clicar (podendo arrastar o mouse junto) nos quadrados da grade indicada como número 3, e desenhar o labirinto como for necessário.

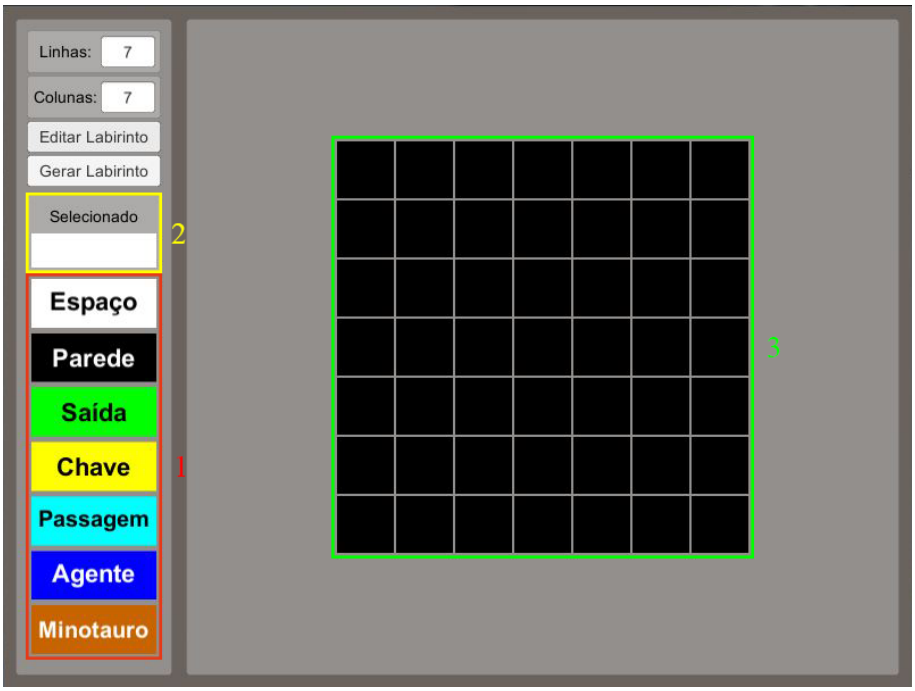


Figura 26 – Interface de edição da matriz do labirinto.

Assim que o *layout* do labirinto estiver concluído, basta clicar no botão Gerar Labirinto, indicado como 1 na Figura 27. O labirinto será gerado e uma tela de seleção de objetivos aparecerá.

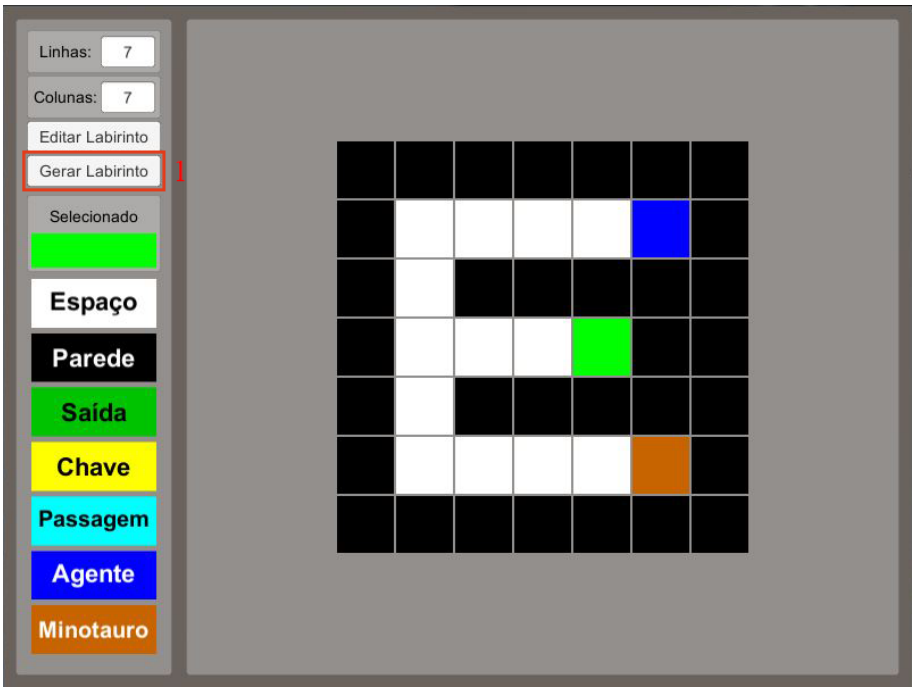


Figura 27 – Exemplo de matriz de geração do labirinto.

A tela de seleção de objetivos possui três objetivos por padrão que podem ser selecionados:

- (atTile player exit) – o objetivo do plano é que o agente esteja no espaço indicado como saída;
- (not (isAlive minotaur)) – o objetivo do plano é garantir que o minotauro não esteja vivo caso o agente o perceba;
- (forall (?tile – tile) (visited ?tile)) – o objetivo do plano é fazer o jogador mover para todos os espaços disponíveis do labirinto.

Para selecionar os objetivos, basta marcar as caixas de seleção indicadas como 1 na Figura 28. Os objetivos podem ser combinados para aumentar a complexidade do problema. Caso o usuário queria adicionar um objetivo que não esteja na lista, basta clicar no botão Novo Objetivo, indicado como número 2.

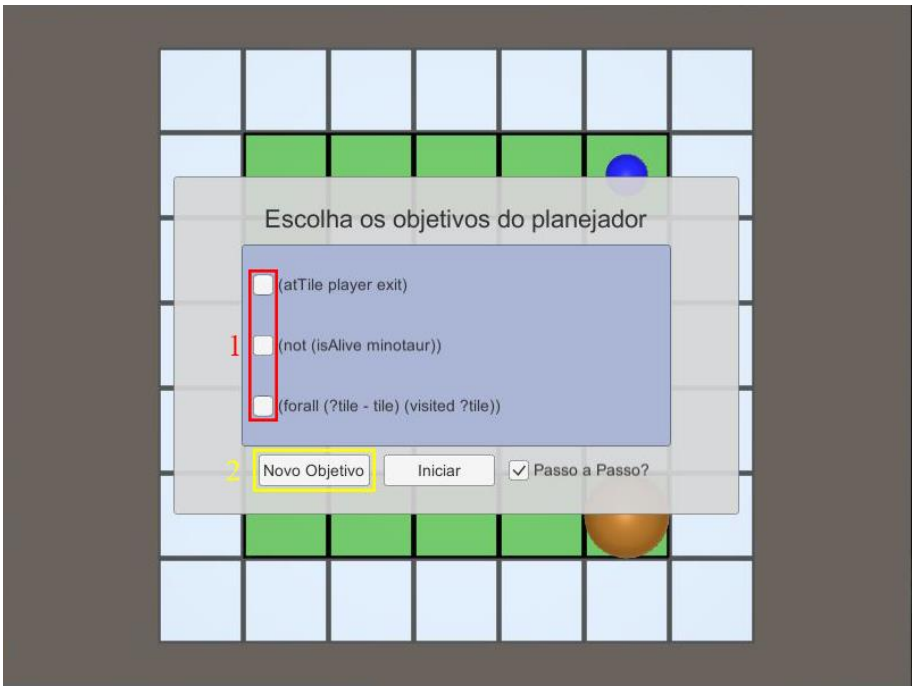


Figura 28 – Interface de escolha do objetivo.

Ao clicar no botão Novo Objetivo, uma nova caixa de seleção, com um campo de inserção de texto será criado, indicado como número 1 na Figura 29. Com isso, o usuário pode digitar no campo de texto o objetivo que deseja, em PDDL, e marcar a caixa de seleção correspondente.

Com os objetivos selecionados, pode-se escolher se se a execução será realizada passo a passo ou de modo contínuo. Para isso, basta marcar ou desmarcar a caixa de seleção indicada como número 2. Com objetivos e modo de execução selecionados, basta clicar no botão Iniciar, indicado como número 3.

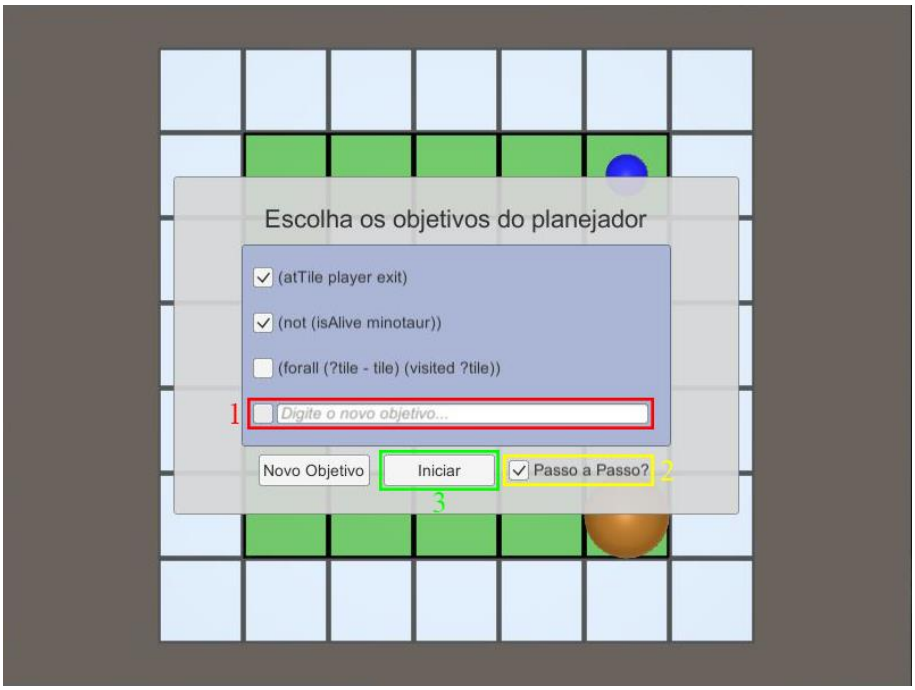


Figura 29 – Exemplo de inserção de novos objetivos.

No início da execução, e a cada replanejamento, uma caixa de comando (Figura 30) externamente à simulação será mostrada, para indicar o status do planejador, seu tempo de execução e se um plano válido foi encontrado. Na sua finalização, basta pressionar qualquer botão para continuar a execução.

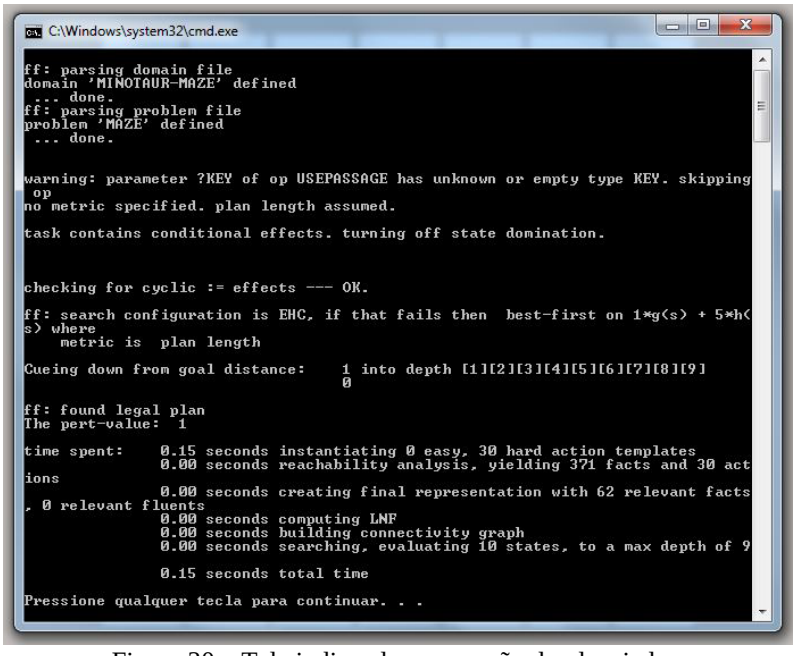
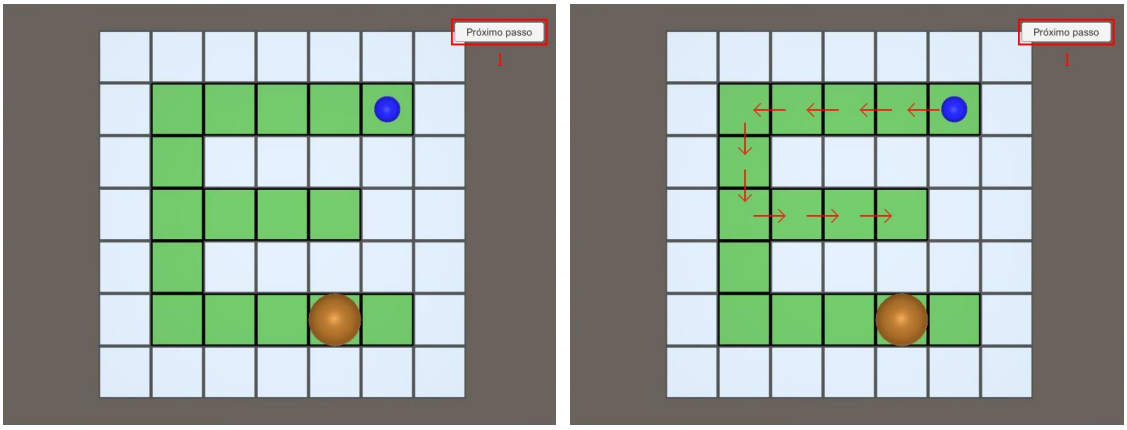


Figura 30 – Tela indicando a execução do planejador.

Caso o usuário tenha selecionado a execução passo a passo, o botão Próximo Passo, indicado como número 1 na Figura 31, estará habilitado. E ao pressioná-lo, o agente realizará a próxima ação. No exemplo da Figura 31 (b), será preciso pressionar o

botão nove vezes (considerando que o minotauro não se move e o agente não precise replanejar).



(a) (b)

Figura 31 – (a) Labirinto gerado, (b) Passos realizado pelo agente.

Quando o agente conclui o plano uma tela de finalização como a apresentada na Figura 32 aparece, dando a opção para que o usuário inicie um novo labirinto.

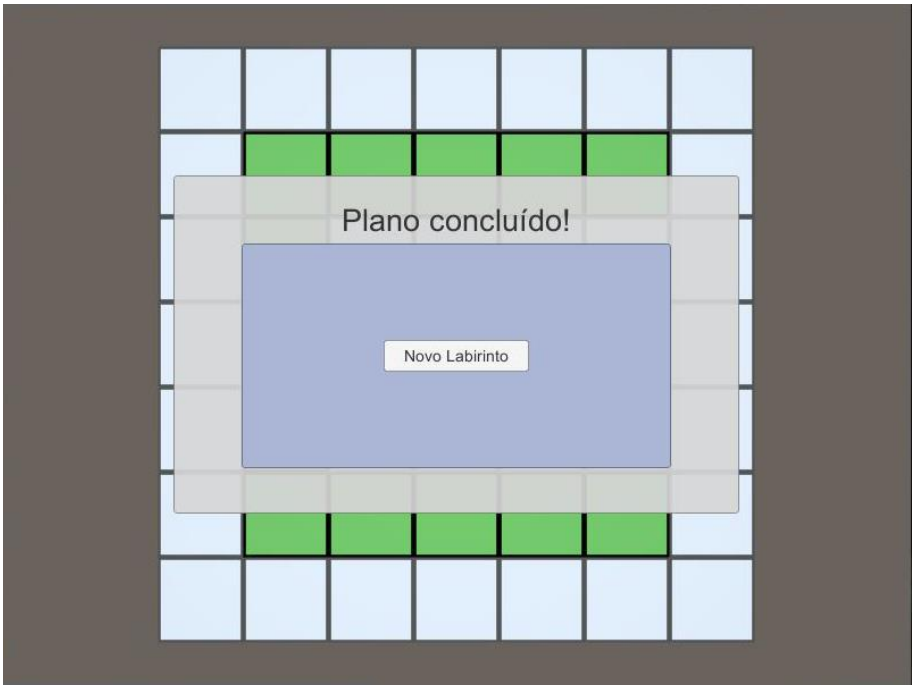


Figura 32 – Tela de finalização do plano.

5. APLICABILIDADE EM OUTRAS ÁREAS

Além de ser uma técnica interessante para ser utilizada em ambiente de jogos virtuais, o *AI planning*, em especial a técnica empregada nesse trabalho, pode ser aplicada à diversas situações reais.

A seguir serão apresentados alguns tópicos onde essa estratégia pode ser aplicada, fazendo sua contextualização, apresentando um problema, analisando e discriminando as alterações necessárias para adaptar esse trabalho aos ambientes propostos.

5.1.1. Indústria 4.0

A Industria 4.0 é um termo que envolve diversas tecnologias para automação e troca de dados e utiliza conceitos de sistemas cyber-físicos, internet das coisas e computação em nuvem.

Segundo Hermann *et al.* (2015), os princípios envolvidos em um projeto de Indústria 4.0 são:

- Interoperabilidade: habilidade dos sistemas cyber-físicos, humanos e fábricas inteligentes se conectarem e comunicarem através da internet e da computação em nuvem;
- Virtualização: capacidade dos espaços cyber-físicos de monitorar processos físicos, podendo associar os dados recebidos de sensores à simulações e modelos virtuais das fábricas;
- Descentralização: habilidade dos sistemas cyber-físicos tomarem decisões em fábricas inteligentes sem precisar de intervenção humana.
- Capacidade em Tempo Real: capacidade de coletar e analisar dados em tempo real, podendo reagir a falhas de máquinas e modificar rotas de produtos para outras máquinas.
- Orientação a Serviço: oferecimento dos serviços dos sistemas cyber-físicos, companhias e humanos através da internet das coisas.
- Modularidade: capacidade de adaptação flexível através da reposição ou expansão de módulos individuais.

Tendo em vista os princípios do projeto, pode-se aplicar as técnicas de *AI planning* para auxiliar na descentralização da indústria inteligente, capacidade de operação em tempo real e na sua modularidade.

Exemplo de problema

Considere uma indústria inteligente com *layout* físico fixo e com vários maquinários distribuídos. Um robô móvel autônomo pode trafegar livremente e utilizar os maquinários. Este robô recebe pedidos de peças no início de sua execução, e essas peças precisam passar por uma sequência de operações antes de serem entregues.

As máquinas dispostas pela indústria realizam operações específicas que podem, ou não, requisitar que uma operação anterior tenha sido realizada. Elas podem apresentar defeito durante a execução, impossibilitando o seu uso e alertando o seu estado para o usuário.

Os espaços da indústria são monitorados em tempo real para averiguar a localização do robô e se alguém adentrou o seu espaço de operação. Pessoas podem trafegar livremente pela fábrica, e o robô tem que tomar decisões para evitar colisões com elas.

Análise do ambiente

Fazendo uma breve análise, seguindo os moldes utilizados em 3.2, as propriedades do problema são:

- Completamente observável: o agente tem a percepção da sua localização, dos estados das máquinas e das localizações das pessoas no ambiente.
- Agente único: nesse cenário o robô opera sozinho no ambiente, as pessoas são consideradas parte do cenário e não como agentes.
- Determinístico: é considerado que as ações desenvolvidas pelo robô terão os seus efeitos realizados com certeza.
- Sequencial: as ações devem ser escolhidas de acordo com um histórico de percepções e ações.
- Estático: assim como no labirinto do minotauro, as alterações do mundo não ocorrem durante a deliberação do agente, e sim durante a sua execução.

- Discreto: para simplificar a análise desse cenário, o tempo de execução das tarefas foi desconsiderado, tendo como foco principal a ordem de execução das ações.
- Conhecido: todas as propriedades do domínio são conhecidas pelo agente, sem que haja necessidade de aprendizado.

Conhecendo as propriedades do ambiente, analisa-se os conhecimentos relevantes para o problema:

- Posição do agente, podendo ser representado em termos de coordenadas cartesianas.
- Localização das máquinas, no caso, representado apenas o terminal onde o robô irá operar, em termos de coordenadas cartesianas.
- Estado das máquinas, representado como um predicado indicando a máquina em questão e seu estado (livre, em operação e com defeito).
- Posição das pessoas, pode ser representado em termos de coordenadas cartesianas.
- Sequência de operações, o agente precisa saber a sequência de operações necessárias para obter um determinado produto, qual máquina realiza determinada operação.

Abordagens possíveis

Neste ambiente, o agente realiza um processo sequencial e cíclico de operações nas máquinas de acordo com os processos necessários para produzi-las, e finaliza a sua operação retornando para a posição inicial, onde aguarda por um novo pedido. Um possível fluxograma de execução deste processo é apresentado na Figura 33.

No fluxograma apresentado, a execução é separada em duas etapas, a inicialização do programa e o ciclo de operações principal. Na inicialização, é verificado a posição atual do agente e considerado como posição inicial. Em sequência o programa tem que receber as informações iniciais da fábrica: posição das máquinas e locais onde o agente vai recolher e entregar os produtos produzidos; operações necessárias para produzir determinado tipo de produto; quais operações as máquinas realizam; e outras informações relevantes. Por fim, o programa inicializa a KDB com essas informações iniciais e fica aguardando um pedido de produção de peças.

Ao receber um pedido de produção, o agente escreve o problema inicial envolvendo este pedido e envia para o planejador gerar o plano com a sequência de ações. Uma verificação similar ao do Labirinto do Minotauro pode ser realizado, onde ele verifica se a ação requisitava pode ser executada ou não.

O processo de replanejamento em decorrência dos defeitos das máquinas é realizado de maneira similar ao apresentado neste trabalho, sendo que o agente atualiza a KDB com as novas informações, escreve um plano atualizado e replaneja as ações. Um adicional a este processo é a emissão de um alarme para indicar que a máquina em questão necessita de manutenção. Caso seja verificado que um alarme já foi emitido, o agente interrompe a execução, retorna para a posição inicial e fica aguardando novos pedidos. Ao finalizar um plano com sucesso, o agente retorna para a posição inicial espera novos pedidos serem realizados.

Neste domínio, é necessário que o agente tenha ações como, por exemplo: *moverPara(de, para)*, *operarMáquina(máquina, operação)*, *emitirAlerta(máquina)* e *entregarProduto(local)*.

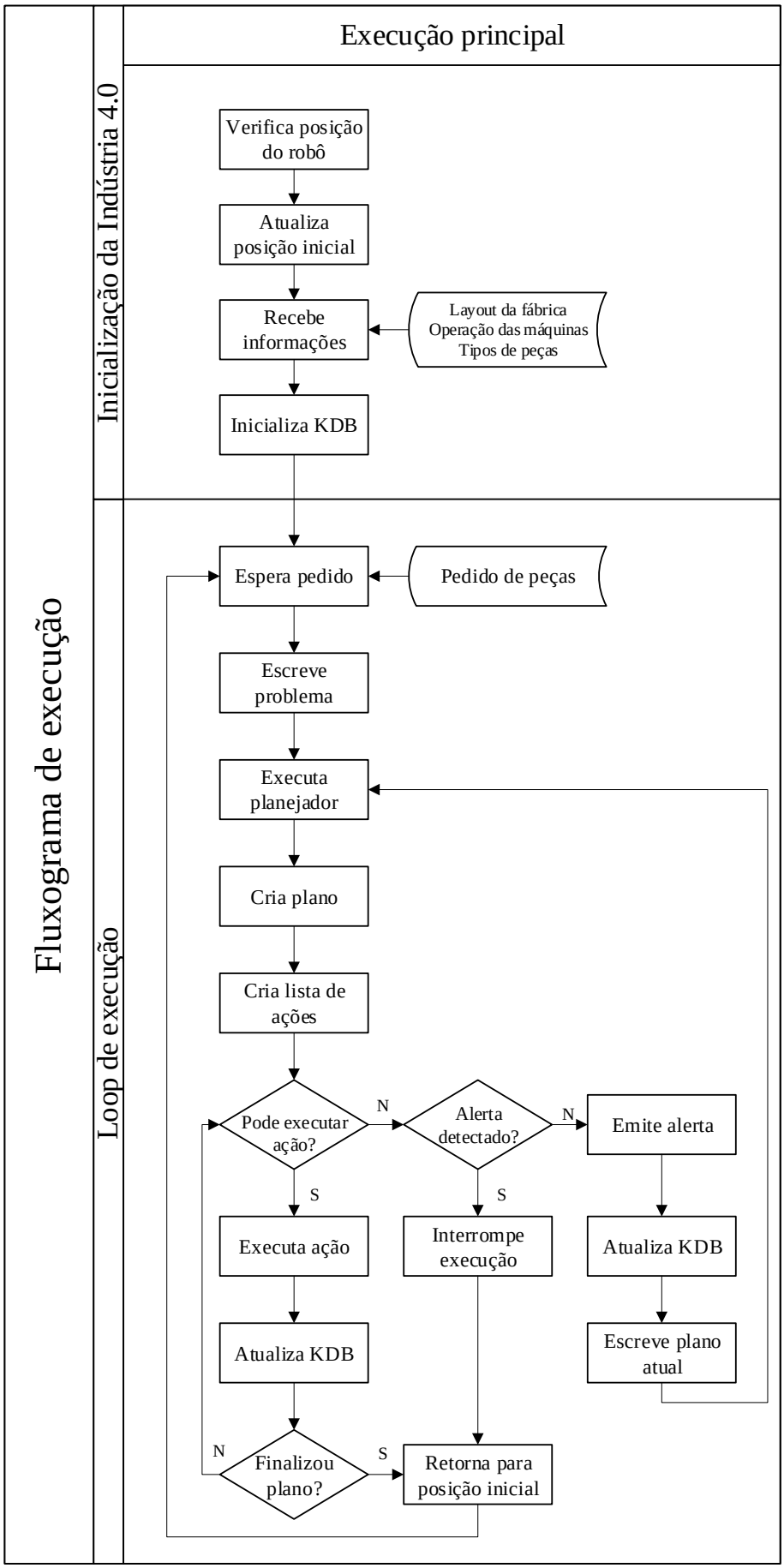


Figura 33 – Fluxograma de execução da Indústria 4.0.

5.1.2. Robôs de resgate e segurança pública

No ambiente de combate a incêndio, há um risco constante às vidas dos bombeiros, e equipes de resgate. Somente no ano de 2016 os bombeiros do estado de São Paulo atenderam mais de 40 mil casos de incêndios em edificações, vegetação e veículos[19]. Uma alternativa para diminuir os riscos dessas operações é utilizar robôs bombeiros e de resgate para realizar ou auxiliar os bombeiros no combate aos incêndios.

Alguns trabalhos envolvendo esse tópico são: o robô humanoide SAFFiR[20] (*Shipboard Autonomous Firefighting Robot*), que está sendo desenvolvido pelo U.S. Office of Naval Research; O robô Thermite[21], desenvolvido pela Howe and Howe Technologies Inc.; O veículo terrestre não tripulado Parosha Cheatah GOSAFER[22] e o MVF-5[23], produzido pela DOK-ING.

Também existe uma modalidade disputada desde 2006 na competição internacional de robótica, a RoboCup, com objetivo de desenvolver soluções robóticas para responderem a situações de emergência, a RoboCupRescue[24].

Exemplo de problema

Uma construção está em chamas e sabe-se que existem pessoas dentro dela e que a estrutura do edifício está comprometida.

Visando uma maior segurança e agilidade, é acionado um robô de resgate que possui balões de oxigênio e indicam a saída para as pessoas que encontra dentro do prédio, além de poder apagar alguns focos de incêndio para facilitar a saída delas.

Antes de adentrar a construção o robô tem como input a planta da construção e possíveis locais onde possa ter pessoas. Após adentrar o local, o robô trafega em direção

---

[19] Dados disponíveis em: http://www.ssp.sp.gov.br/Estatistica/CorpoBombeiro.aspx (Acesso em 15 de junho de 2018)
[20] Informações disponíveis em: https://www.onr.navy.mil/en/Media-Center/Press-Releases/2015/saffir-ship-firefighting-robot-prototype (Acesso em 05 de junho de 2018)
[21] Informações disponíveis em: http://www.roboticfirefighters.com/ (Acesso em 05 de junho de 2018)
[22] Informações disponíveis em: http://www.parosha-cheatah-gosafer.com/ (Acesso em 05 de junho de 2018)
[23] Informações disponíveis em: http://www.dok-ing.hr/products/firefighting/mvf_5 (Acesso em 05 de junho de 2018)
[24] Informações disponíveis em: http://wiki.robocup.org/Robot_League (Acesso em 30 de maio de 2018)

aos locais indicados, verificando em execução se o *layout* está condizente com o *input* recebido, atualizando o mapa caso haja alterações e decidindo a rota que será tomada.

Ao encontrar uma pessoa, ele entrega uma bolsa de oxigênio, escolta para a saída e continua a busca em outros locais, até que a varredura do mapa esteja completa e indique para o exterior que completou o resgate.

Análise do ambiente

Como em uma situação no mundo real é extremamente complexa, algumas propriedades do ambiente foram simplificadas para realizar essa análise. Obtendo, assim, as seguintes propriedades:

- Parcialmente observável: o agente não tem conhecimento das propriedades do ambiente até que ele entre em sua percepção, como focos de incêndios e pessoas para serem resgatadas.
- Agente único: o único agente é o robô de resgate, as pessoas a serem resgatadas são consideradas parte do domínio.
- Determinístico: uma ação realizada pelo agente terá seus efeitos aplicados no ambiente com sucesso.
- Sequencial: a sequência de ações realizadas pelo agente tem um impacto no ambiente, e elas devem ser escolhidas dado um histórico de percepções.
- Dinâmico: o ambiente pode alterar-se durante a deliberação do agente, uma parede pode desabar, ou uma pessoa a ser resgatada pode mover-se.
- Discreto: para efeito de simplificação, não serão considerados os tempos de execução das ações, somente a ordem de execução das mesmas.
- Conhecido: o agente possui o modelo do domínio, como suas ações impactam no ambiente e como ele é apresentado, portanto não necessita de aprender sobre o mesmo.

Os conhecimentos necessários para o agente resolver os problemas presentes nesse domínio são os seguintes:

- Planta da construção: um mapa discretizado do edifício contendo as vizinhanças entre cômodos ou, caso queria um nível maior de detalhes, espaços discretos

dentro de um mesmo ambiente, a fim de formar um grafo de navegação para o agente.

- Localização do agente: localização do agente dentro do grafo de navegação da planta da construção.
- Quantidade de bolsas de oxigênio: uma quantidade máxima de bolsa de oxigênio que o agente pode carregar em cada viagem, essa quantidade pode ser reposta ou não sempre que entregar uma pessoa socorrida.
- Localização das vítimas: uma entrada de informação que indica a possível localização, dentro do grafo de navegação, de vítimas do incêndio.

Abordagens possíveis

Nesse ambiente, o planejador traça inicialmente a rota para os possíveis pontos onde tem pessoas, porém o grafo criado inicialmente pode não condizer com a realidade, devido à desmoronamentos e a focos de incêndios. Ao encontrar essas inconsistências, o agente tem que ser capaz de atualizar os seus conhecimentos prévios e gerar uma nova rota até o seu objetivo.

Ao encontrar uma vítima, o agente adiciona o objetivo de entregar a bolsa de oxigênio e de escoltar a pessoa para a saída com prioridade, e somente após concluir esse objetivo, continuar a varredura anterior até os outros pontos.

Do trabalho desenvolvido, pode ser utilizado a estrutura de movimentação discreta, baseada em grafo de vizinhanças, a capacidade de replanejamento devido às incoerências com o conhecimento prévio e o planejamento de rotas e utilização de objetos.

Para ser possível resolver esse problema, é necessário implementar uma técnica de *scheduling* para definir prioridades de conclusão dos objetivos. No caso, dar prioridade para escoltar a vítima para fora, ao invés de continuar a busca.

6. CONCLUSÃO

Neste capítulo serão realizadas considerações gerais sobre o trabalho e em sequência serão propostos trabalhos futuros.

6.1. CONSIDERAÇÕES GERAIS

Visando a utilização dos planejadores automáticos em ambiente de jogos, foi desenvolvido o ambiente do Labirinto do Minotauro e implementada uma técnica de planejamento e ação, onde o agente averigua as ações realizadas e as replaneja, caso necessário. Esta técnica pode ser utilizada em diversos ambientes, não somente em jogos, porém, um fator determinante para o sucesso do planejador é o modo em que o ambiente é modelado pois, como apontado na sessão 4.1, a responsabilidade pela maior parte da inteligência do agente é do domínio de planejamento.

A implementação de um planejamento e ação tem que ser bem estruturada. Dependendo de como foi codificado o processo de replanejamento, o agente pode entrar em um processo de replanejamento infinito, sem que ele consiga realizar alguma ação. Uma solução possível é monitorar a quantidade de replanejamentos efetuadas em um instante e, caso verifique que o agente ficou preso neste processo, determinar o último plano encontrado como válido ou alterar os objetivos do planejador.

A falta de conhecimento prévio sobre alguns assuntos presentes nesse trabalho acarretou em problemas envolvendo a modelagem do domínio utilizado. Para atender à presença do minotauro no ambiente, ações simples como mover ficaram dependentes da existência do mesmo, impossibilitando testes com o agente em labirintos puros. Dificuldades enfrentadas ao codificar a interface entre o planejador e o ambiente de simulação acarretou em uma perda de flexibilidade a alterações, contrapondo à vantagem que os planejadores provêm.

6.2. TRABALHOS FUTUROS

Com o atual conhecimento sobre o assunto tratado neste trabalho, algumas perspectivas para novos trabalhos são sugeridas:

- Tornar o agente mais flexível, deixando uma “caixa preta” para ser preenchida por técnicas de IA distintas para que uma comparação de desempenho seja evidente;
- Implementar um ambiente de simulação que comporte mais de um agente e/ou minotauro e desenvolver mecanismos multiagente de cooperação/competição;
- Desenvolver cenários de simulações para as situações apresentadas em 5 e aplicar técnicas de planejamento e ação;
- Aplicar as técnicas estudadas de planejamento e ação em um robô para agir em um ambiente físico.

7. REFERÊNCIAS

BJÖRNSSON, Y.; FINNSSON, H. CadiaPlayer: A Simulation-Based General Game Player. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 1, n. 1, p. 4-15, 2009.

BLUM, A. L.; FURST, M. L. Fast planning through planning graph analysis. **Artificial Intelligence**, v. 90, n. 1-2, p. 281-300, 1997.

BOJARSKI, S.; CONGDON, C. B. REALM: A rule-based evolutionary computation agent that learns to play Mario. **Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games**, Dublin, 2010. 83-90.

BURKE, R. et al. Creature Smarts: The Art and Architecture of a Virtual Brain, 2001.

ČERNÝ, M. et al. To Plan or to Simply React? An Experimental Study of Action Planning in a Game Environment. **Computational Intelligence**, v. 32, n. 4, p. 668-710, 2016.

CIANCIULLI, S.; VASSOS, S. **Planning for Interactive Storytelling Processes**. Proceedings of the 3rd International Planning in Games Workshop. Rome: ICAPS. 2013. p. 23-26.

DILL, K. What Is Game AI? In: RABIN, S. **Game AI Pro:** Collected Wisdom of Game AI Professionals. Boca Raton: CRC Press, 2014. p. 3-9.

DU, R.; GAO, Z.; XU, Z. Deliberately Planning and Acting for Angry Birds with Refinement Methods, 2015.

FLEURY, A.; SAKUDA, L. O.; CORDEIRO, J. H. D. **I Censo da Indústria Brasileira de Jogos Digitais, com Vocabulário Técnico sobre a IBJD**. Pesquisa do GEDIGames, NPGT, Escola Politécnica, USP, para o BNDES. São Paulo. 2014.

FUNKE, S.; NUSSER, A.; STORANDT, S. The Simultaneous Maze Solving Problem. **AAAI Conference on Artificial Intelligence**, 2017. 808-814.

GEREVINI, A.; LONG, D. **Plan Constraints and Preferences in PDDL3**. Technical Report 2005-08-07, Department of Electronics for Automation, University of Brescia. Brescia, Italy. 2005.

GHALLAB, M.; NAU, D.; TRAVERSO, P. **Automated Planning:** theory and practice. San Francisco: Elsevier, 2004.

GHALLAB, M.; NAU, D.; TRAVERSO, P. **Automated Planning and Acting**. Nova York: Cambridge University Press, 2016.

GRAND, S.; CLIFF, D.; MALHOTRA, A. Creatures: artificial life autonomous software agents for home entertainment. **Proceedings of the first international conference on Autonomous agents**, Marina del Rey, 1997. 22-29.

HERMANN, M.; PENTEK, T.; OTTO, B. Design Principles for Industrie 4.0 Scenarios: A Literature Review. **Technische Universität Dortmund, working paper**, n. 01, 2015.

HOFFMANN, J. FF: The fast-forward planning system. **AI Magazine**, v. 22, n. 3, p. 57-62, 2001.

HOFFMANN, J.; NEBEL, B. The FF Planning System: Fast Plan Generation Through Heuristic Search. **Journal of Artificial Intelligence Research**, v. 14, p. 253-302, 2001.

ISLA, D.; BLUMBERG, B. New Challenges for Character-Based AI for Games. **AAAI Spring Symposium on AI and Interactive Entertainment**, Palo Alto, 2002.

MAHMOUND, I. M. et al. **Believable NPCs in Serious Games:** HTN Planning Approach Based on Visual Perception. 2014 IEEE Conference on Computational Intelligence and Games. Dortmund: IEEE. 2014. p. 1-8.

MATEAS, M.; STERN, A. A Behavior Language for Story-Based Believable Agents. **IEEE Intelligent Systems**, v. 17, n. 4, p. 39-47, 2002.

MCDERMOTT, D. et al. PDDL - The Planning Domain Definition Language, 1998.

MILLINGTON, I.; FUNGE, J. **Artificial Intelligence for Games**. 2ª. ed. Boca Raton: CRC Press, 2009.

NEUFELD, X. et al. Building a Planner: A Survey of Planning Systems Used in Commercial Video Games. **IEEE Transactions on Games**, 2017.

ORKIN, J. Applying goal-oriented action planning to games. **AI Game Programming Wisdom**, v. 2, p. 217-228, 2003.

ORKIN, J. Symbolic representation of game world state: Toward real-time planning in games. **Proceedings of the AAAI Workshop on Challenges in Game Artificial Intelligence**, 2004. 26-30.

ORKIN, J. Agent Architecture Considerations for Real-Time Planning in Games. **AIIDE**, 2005. 105-110.

ORKIN, J. Three States and a Plan: The A.I. of F.E.A.R. **Game Developers Conference**, 2006.

RUSSELL, S.; NORVIG, P. **Inteligência Artificial:** Tradução da 3a Edição. Rio de Janeiro: Elsevier Brasil, 2013.

SOEMERS, D. J. N. J.; WINANDS, M. H. M. Hierarchical Task Network Plan Reuse for video games. **2016 IEEE Conference on Computational Intelligence and Games (CIG)**, 2016. 1-8.

TENCÉ, F. et al. The Challenges of Believability in Video Games: Definitions, Agents' Models and Imitation Learning. **arXiv preprint arXiv:1009.0451**, 2010.

TOGELIUS, J.; KARAKOVSKIY, S.; BAUMGARTEN, R. The 2009 Mario AI Competition. **IEEE Congress on Evolutionary Computation**, Barcelona, 2010. 1-8.

YANNAKAKIS, G. N.; TOGELIUS, J. A Panorama of Artificial and Computacional Intelligence in Games. **IEEE Transactions on Computacional Intelligence and AI in Games**, v. 7, n. 4, p. 317 - 335, 2015.

APÊNDICE A – Domínio desenvolvido para o problema do labirinto do minotauro e codificado em PDDL

```
(define (domain minotaur-maze)
  (:requirements :adl)

  (:types
    tile - tile
    key - object
    bomb - object
    object
    player - agent
    minotaur - agent
    agent
  )

  (:predicates
    (atTile ?agent - agent ?tile - tile)
    (neighbour ?tile1 - tile ?tile2 - tile)
    (visited ?tile - tile)
    (tileSafe ?tile – tile)
    (objectOnMap ?object - object ?tile - tile)
    (haveObject ?object - object)
    (passage ?tile1 - tile ?tile2 - tile)
    (isAlive ?agent - agent)
  )

  (:action moveTo
    ;; Move o ?player do espaço de origem ?from para o de destino ?to se os
    ;; espaços forem vizinhos e o minotauro não esteja no destino
    :parameters    (?player - player ?from - tile ?to – tile ?minotaur -
                   minotaur)
    :precondition  (and  (atTile ?player ?from) (neighbor ?from ?to)
                         (or  (not (isAlive ?minotaur))
                              (not (atTile ?minotaur ?to))
                         )
                   )
    :effect        (and  (not (atTile ?player ?at)) (atTile ?player ?to)
                         (when (not (visited ?to)) (visited ?to))
                   )
  )

  (:action pick
    ;; ?player pega o ?objeto que está no espaço ?location
    :parameters    (?player - player ?object - object ?location - tile)
    :precondition  (and  (atTile ?player ?location)
                         (objectOnMap ?object ?location)
                   )
    :effect        (and  (not (objectOnMap ?object ?location))
                         (haveObject ?object)
                   )
  )
```

(continua na próxima página)

(continuação do código)

```
  (:action usePassage
    ;; Usa passagem do espaço de origem ?from para o destino ?to se possuir uma
    ;; chave ?key. Não é obrigatório os tiles serem vizinhos (teletransporte).
    :parameters    (?player - player ?from - tile ?to – tile ?key - key)
    :precondition  (and  (haveObject ?key) (atTile ?player ?from)
                         (passage ?from ?to)
                   )
    :effect        (and  (not (atTile ?player ?from)) (not (haveObject ?key))
                         (atTile ?player ?to)
                         (when (not (visited ?to)) (visited ?to))
                   )
  )

   (:action throwBomb
    ;; Joga uma bomba no tile ?tile, explodindo em cruz (+), que mata o
    ;; minotauro. É obrigatório o agente ter a bomba e o minotauro estar vivo.
    :parameters    (?tile – tile ?bomb - bomb ?minotaur – minotaur ?player -
                   player ?pTile - tile)
    :precondition  (and  (haveObject ?bomb) (isAlive ?minotaur)
                         (atTile ?player ?location)
                         (exists  (?pNeighbour – tile)
                                  (and  (neighbour ?pTile ?pNeighbour)
                                        (neighbour ?pNeighbour ?tile)
                                  )
                         )
                         (or      (atTile ?minotaur ?tile)
                                  (exists   (?mNeighbour - tile)
                                            (and  (neighbour ?tile ?mNeighbour)
                                                  (atTile ?minotaur ?mNeighbour)
                                            )
                                  )
                         )
                   )
    :effect        (and  (not (haveObject ?bomb)
                         (not (isAlive ?minotaur)
                   )
  )
)
```