O MUNDO DE WUMPUS

May 2023

Table 1 – Ambiente de tarefa

Tipo de Agente	**Medida de desempenho**	**Ambiente**	**Atuadores**	**Sensnores**
O mundo do Wumpus	Pegar Ouro; Matar Wumpus; Volar para a posição (0, 0) pelo menor caminho.	Matriz x por x; Obstáculos.	Ação: Andar, atirar e pegar ouro.	Perceber o ambiente;

Table 2 – Ambiente de tarefa e suas características

Tipo de Agente	**Observável**	**Agentes**	**Determinístico**	**Espódico**	**Estático**	**Discreto**
O mundo do Wumpus	parcialmente observável	multiagente	Determinístico	Sequencial	Dinânico	Discreto

- **Parcialmente observável:** Precisa de um estado interno para acompanhar as mudanças no ambiente.
- **Multiagente:** Para maximiza a medida de desempenho depende de outros agentes.
- **Determinístico:** Se o próximo estado do ambiente é completamente determinado pelo estado atual e pela ação executada pelo agente, dizemos que o ambiente é determinístico;
- **Sequenciais:** Em ambientes sequenciais, a decisão atual poderia afetar todas as decisões futuras.
- **Estático** O ambiente não se altera quando o caçador está processando informações.
- **Discreto:** O ambiente do jogo tem um número finito de estados distintos. Também tem um conjunto discreto de percepções e ações.

1 Parte 1 - Projeto

A **etapa 1** consiste na construção de um ambiente, matriz x por x, que será usada para estanciar os demais agentes: caçador, wumpus, ouro e buraco. Nessa etapa o agente é totalmente reativo as percepções do ambiente, são elas: fedor, brisa e brilho.A percepção brilho por não ser percebida nas casas adjacentes, diferente das demais, não tem tanta influência nas reações do agente. Veja que nessa fase não há memória envolvida, portanto, as ações do agente, andar nas quatros direções, atira e pegar ouro, é completamente aleatória.

Foi desenvolvido em python o código desse ambiente, que pode ser acessado com o link abaixo:
Wumpus — Reativo

2 Parte 2 - Projeto

A parte 2 desse projeto versa na criação de um agente baseado na utilidade, para tanto, este deve conter além das suas características, as de 3 outros modelos, sendo estes:

1. **Agentes reativos simples:** Esse é o programa parte 1, já feita.
2. **Agentes reativos baseados em modelo:** Esse é um programa que descreve o funcionamento do mundo. Nele há a presença de memórias que permite a retenção de informações e possibilita a inferência, gerando assim conhecimento que auxilia na tomada de disseção.
3. **Agentes baseados em objetivos:** Nesse programa além das outras características anteriores citadas, há a adição de um objetivo.

O agente baseado na **utilidade** tem como característica uma função *utilidade do agente* que é essencialmente uma internalização da medida de desempenho.

2.1 Raciocínio aplicado na programação

Aqui serão citadas as **funções** desenvolvidas no projeto parte 2 e comentadas brevemente como funcionam:

- **perceber_ambiente():** Essa função retorna uma lista de percepções que o caçador está sentindo no ambiente.
- **perigo(lista_percepcao):** Essa função retorna True or False. Se há ou não perigo na região.
- **upload_historico(pos_anterior, tamanho, lista_percepcao, d_historico, ir_para_posicao):** Essa função armazena as informações passadas por parâmetro:

  pos_anterior → posição anterior do caçador;

  lista_percepcao → As percepções daquela posição;

  ir_para_posicao → posição escolhida, nova posição.
- **atirar(posicao_atirar):** Recebe a posição que deve ser atirada a flecha, retorna True or False, matou ou não o wumpus.
- **posicao_(l_casas_perigosas, l_casas_seguras, l_casas_percorridas, d_historico):** Retorna a posição que o caçador deve ir. Essa função faz, baseado nas listas,l_casas_perigosas, l_casas_seguras, l_casas_percorridas e d_historico, qual posição adjacente ao caçador é a ideal ou a melhor para se ir.

  Nessa função é verificado se a posição atual do caçador tem casas adjacentes que geram algum risco, caso não, então significa que as casas adjacentes são seguras, assim verifica-se se já foi percorrida pelo caçador. Caso todas as casas adjacentes ao caçador tenha sido precorridas sendo seguras há a seleção aleatórias da nova posição.

  Já se houver risco, percorre-se as listas: l_casas_perigosas e l_casas_percorridas verificando qual posição adjacente atual **não estão na lista** e as salvam em outra lista. percorre também a lista, l_casas_seguras buscando qual das posições salvas na nova lista **estão na lista** casa seguras.

  Depois obter-se-á uma lista com posições possíveis, assim, verifica-se o tamanho da lista, se houver mais de uma posição possível a posição será escolhida randomicamente.

- **voltar_pos_anterior(c_posicao, d_historico):** volta o caçador para a posição anterior, para isso recebe a posição atual do caçador e recebe o dicionário com as informações e consulta qual era a posição anterior do caçador.
- **procesar_historico(l_casas_percorridas, d_historico)** Essa função cruza as posições adjacentes atuais do caçador, com as posições adjacentes das casas que ele já esteve, caso haja uma **intercessão** entre as posições adjacentes classifica-se está como possível **casa perigo**. Depois disso percorrem-se as percepções da posição relacionada a essa casa adjacente classificada com possível casa perigo e comparam-se as percepções atuais do caçador com as percepções da posição que teve sua casa adjacente classificada como perigosa, caso haja igualdade entre elas, significa que corresponderá ao habitante da casa perigo. No fim é retornado um lista com a posição perigo e o habitante dela, podendo ser: None, wumpus ou buraco.
- **inferir(l_casas_percorridas, d_historico):** Usa a função **procesar_historico(l_casas_percorridas, d_historico)** que retorna uma possível posição perigo e seu habitante, a parti dessas informações de retorno é verificado se é o **Wumpus** ou **Buraco** ou **None**. Se wumpus, atirar nessa direção, se Buraco armazenar posição na lista casas_perigosas, se None não faça nada.

2.2 Coisas que faltam implementar

O programa está na fase agente baseado em objeto. Faltando apenas a implementação do retorno para a casa (0,0) pelo menor caminho possível.

OBSERVAÇÕES:

1. Foi detectado um comportamento, interessante que será debugado, se os poços e o wumpus ficarem em uma posição que não der de criar intercessões. O Caçador andará apenas em locais que ele inferir seguro, assim ele não é ousado para ir em posições que tenha probabilidade 50%/50%, assim terá rodadas que ele entra em loop;
2. A parte baseado na utilidade ainda será implementada.
3. ainda estamos processando todas as informações, provavelmente faltarão coisas que não conseguimos perceber no momento.

   o projeto com a Parte 2 pode ser acessado com o link abaixo:

   Wumpus — Utilidade