## INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA



**ÁREA DEPARTAMENTAMENTAL DE ENGENHARIA DE ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES**

## Licenciatura em Engenharia Informática e Multimédia

**Inteligência Artificial para Sistemas Autónomos**

**Semestre de Verão 19/20**

Síntese do 3º Trabalho Prático

**Turma:** 41D

**Data:** 20 junho 2020 **Docente:** Luís Morgado **Nome:** Luís Fonseca **Número:** 45125

Introdução

**O terceiro trabalho de IASA consiste em implementar e desenvolver a concretização de modelos e arquiteturas de agentes inteligentes (tendo por base uma Plataforma de Simulação de Agentes). Foi necessário definir o que era um sistema autónomo inteligente, já que neste trabalho o agente tem de agir de forma autónoma inteligente, já que neste trabalho o agente tem de agir de forma autónoma para recolher todos os alvos da melhor forma possível. Sendo assim um sistema autónomo inteligente é um sistema que usa inteligência para aprender o ambiente que o rodeia.**

Aprendizagem Automática

**Pode-se definir aprendizagem automática como a aplicação da inteligência artificial que atribui a agentes inteligentes a capacidade de aprender de modo automático, aprendendo e melhorando a partida da experiência.**

**O processo de aprendizagem é inicializado através da observação de toda a informação disponível como é disso exemplo a experiência direta. A partir desta informação procuram-se padrões, de modo a que possam ser tomadas melhores decisões no futuro tendo como base os exemplos passados que são dados.**

Agente

Neste trabalho o agente tem de agir de forma autónoma e tem como principal objetivo recolher todos os alvos da forma mais eficiente possível. O agente possui sensores que lhe permitem ter uma perceção do ambiente exterior de forma a poder processar informação. Através dos seus atuadores o agente altera ou não o ambiente que o rodeia. Isto leva a um ciclo de realimentação, onde existe acoplamento com o ambiente.

Arquitetura

O ambiente relaciona-se com o agente de forma a ser possível estar perante um sistema autónomo inteligente. No contexto deste trabalho o ambiente vai ser **estático,** pois permanece inalterado apenas muda sob ação do agente. Um ambiente pode ser representado de várias formas e tem várias propriedades que se encontram abaixo explicadas:

* Discreto vs Contínuo: um ambiente pode ser discreto se existe um número finito de perceções e ações possíveis para o agente e contínuo caso contrário;
* Determinístico vs Estocástico: um ambiente determinístico, cada ação tem um efeito único garantido, onde não existem incerteza quanto ao resultado da ação.
* Estático vs Dinânimo: um ambiente estático permanece inalterado enquanto o agente decide a próxima ação a executar. Pelo contrário uns ambientes dinâmicos com vários agentes encontram-se a agir num ambiente ao mesmo tempo;
* Totalmente vs Parcialmente observável: consiste na capacidade de observar o ambiente, observado totalmente parcialmente;
* Episódico vs Sequencial: episódico se a próxima ação depende do estado atual e Sequencial se depende dos estados passados;
* Agente único vs Múltiplos: se apenas existe um agente ou vários;

Arquitetura

**Um agente contém 3 tipos de arquiteturas, que são:**

* **Arquitetura reativa: este tipo de arquitetura é baseado na capacidade de um agente reagir rapidamente às mudanças no seu ambiente. Para tal, o agente deve ser capaz de se aperceber do seu ambiente e atuar sobre o mesmo. O agente tem a capacidade de decidir as suas ações sem consultar um modelo interno do mundo;**
* **Arquitetura de subsunção: este tipo de arquitetura permite que os comportamentos sejam organizados em camadas e que sejam responsáveis pela concretização independente de m objetivo. O resultado do comportamento pode ser a entrada de outro comportamento. Existe a possibilidade de comportamentos de camadas superiores assumirem o controlo sobre comportamentos das camadas inferiores, onde a camada inferior não tem conhecimento das camadas superiores – Hierarquia de comportamentos. Nesta arquitetura as camadas superiores controlam as camadas inferiores, onde as saídas das camadas inferiores podem ser controladas pelas camadas superiores.**
* **Arquitetura Deliberativa: este tipo de arquitetura segue a abordagem clássica da Inteligência Artificial, onde os agentes atuam com pouca autonomia e possuem modelos simbólicos explícitos dos seus ambientes. Esta arquitetura também tem por base o tempo passado, presenta ao futuro, ao contrário da reativa que apenas tem em conta o presenta e possivelmente o passado,**

**Neste trabalho foi estudado também os Processos de decisão sequencial, onde surge o problema da decisão ao longo do tempo. Nestes processos surge o conceito de utilidade de uma ação.**

**Um dos processos estudados foram os processos de decisão de Markov (PDM). De acordo com o PDM a previsão dos estados seguintes só depende do estado presente e o mundo está representado da seguinte forma:**

**S – Conjunto de estados;**

**A(s) – Conjunto de ações possíveis no estado s pertencente a S;**

**T (s, a,** **s’) – Probabilidade de transição de s para s’ através de a;**

**R (s, a, s’) – Retorno ou recompensa esperado na transição de s para s’ através de a;**

γ - Taxa de desconto para recompensas diferidas no tempo, ou seja, é a perda de oportunidade. Só toma valores 0 ou 1 [0,1] representando o fato de desconto temporal;

**t – Tempo discreto: 1,2,3…**

Aprendizagem

**Com base na aprendizagem automática, a aprendizagem é uma melhoria de desempenho(D) para uma determinada tarefa(T), com a experiência(E). Onde por exemplo, em jogar xadrez a tarefa seja jogar xadrez, o desempenho a percentagem de jogos ganhos, e a experiência os jogos realizados. O conceito de aprendizagem é totalmente diferente de memorização. A aprendizagem baseia-se na generalização, ou seja, na formação de abstrações que consistem em modelos.**

Aprendizagem por reforço

A Aprendizagem por reforço baseia-se numa aprendizagem que tem por base a interação com o ambiente e da realização de comportamentos de forma a ganhar experiência. A partir de um determinado estado o agente escolhe a ação para mudar para o estado seguinte. Essa ação gera um reforço (Aquilo que na prática concretiza a motivação através de um incentivo) positivo ou negativo.

Explorar vs Aproveitar

O agente depois de aprender tem que aplicar o que aprendeu, mas, no entanto, é difícil de saber quando é que o agente já aprendeu o suficiente para aplicar.

Como tal, o agente pode explorar ou aproveitar:

* O agente ao explorar, escolhe uma ação que permite explorar o mundo de forma a melhorar a aprendizagem;
* O agente ao aproveitar escolhe a ação que leva à melhor recompensa de acordo com a aprendizagem. Segue uma estratégia Greedy que corresponde a uma ação Sôfrega. Se o agente aproveita muito fica mais “medronho” por não arriscar;

Épsilon-Greedy

É escolhida uma ação aleatória com probabilidade épsilon.

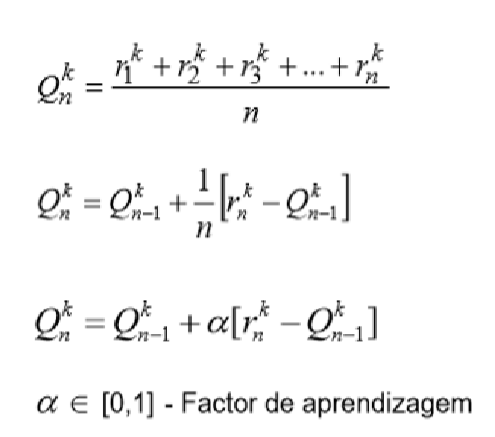


Figura 1 - valor da Ação

Na figura de baixo, é possível de ver como funciona a aprendizagem por reforço, com este algoritmo:

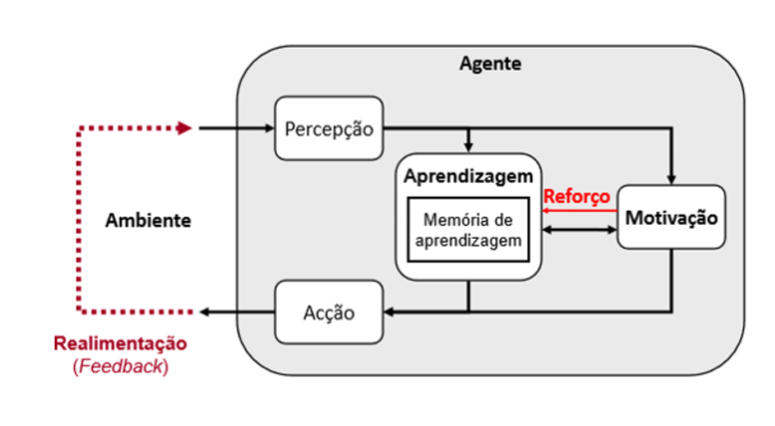


Figura 2 - aprendizagem por reforço

Algoritmo SARSA

Este algoritmo previne grandes perdas e os caminhos podem não ser os melhores por ser uma exploração aleatória.

Algoritmo Q-Learning

Este caminho escolhe os melhores caminhos, pois os melhores são aqueles que maximizam a função Q, ou seja, a decisão é ótima

Processos De Aprendizagem

Existem 2 tipos de processos de aprendizagem:

* Política de seleção de ação única (On-Policy) – usa a mesma política de seleção de ação para comportamento e para programação de valor, explorando todas as ações (política épsilon-greedy).
* Política de seleção de ação diferenciadas – usam políticas de seleção de ação distintas para comportamento e propagação de valor, otimizando a função de valor Q(s,a).

Na implementação…

Com a teoria estudada, foi feita a realização deste tema, em código, desta na linguagem python.

Para isso iremos ter a classe AgenteProspector, que irá herdar os métodos do Agente (agente este usando as bibliotecas psa e pee, fornecidas pelo docente da disciplina) e com os seguintes métodos:

* Executar (): que serve para executar o nosso agente, este método permite que o agente percecione, processe uma perceção e possa atuar segundo uma ação.
* \_\_percecionar () que lê um sensor;
* \_\_processar () que permite ao agente processar uma determinar perceção;
* \_\_atuar () que permite ao agente atuar segundo uma ação, caso o agente contenha uma ação, ele atua sobre essa ação.

class AgenteProspector(Agente):  
  
 def \_\_init\_\_(self, controlo):  
 self.controlo = controlo  
  
 def executar(self):  
 percepcao = self.\_\_percepcionar()  
 accao = self.\_\_processar(percepcao)  
 self.\_\_actuar(accao)  
  
 def \_\_percepcionar(self):  
 return self.sensor\_multiplo.detectar()  
  
 def \_\_processar(self, percepcao):  
 return self.controlo.processar(percepcao)  
  
 def \_\_actuar(self, accao):  
 if accao is not None:  
 return self.actuador.actuar(accao)

Outros dos packages que iremos conter neste projeto é o agente conter diferentes reações, reações essas que são:

* Contornar: permite ao agente contornar, tanto para a esquerda como para a direita

class Contornar(Reaccao):  
  
 def \_detectar\_estimulo(self, percepcao):  
 return (percepcao[DIR].contacto and percepcao[DIR].obstaculo) \  
 or (percepcao[ESQ].contacto and percepcao[ESQ].obstaculo)  
  
 def \_gerar\_resposta(self, estimulo):  
 accao = Mover(FRT)  
 resposta = Resposta(accao)  
 return resposta

* Evitar: o agente, caso encontre um obstáculo, evita-o;

class EvitarObst(Reaccao):  
  
 def \_detectar\_estimulo(self, percepcao):  
 return percepcao[FRT].contacto and percepcao[FRT].obstaculo  
  
 def \_gerar\_resposta(self, estimulo):  
 accao = Rodar(DIR)  
 resposta = Resposta(accao)  
 return resposta

* Explorar: o agente explora o ambiente em seu redor;

class Explorar(Comportamento):  
  
 def activar(self, percepcao):  
 # percepcao tem peso  
 angulos = [ESQ, DIR, FRT]  
 angulo = choice(angulos)  
 accao = Mover(angulo)  
 return Resposta(accao)

* Recolher: quando o agente encontra um alvo, recolhe esse mesmo alvo;

class Recolher(Hierarquia):  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_([AproximarAlvo(), EvitarObst(), Contornar(), Explorar()])

De seguida, iremos conte outro package, que contém uma outra biblioteca, biblioteca esta que contém alguns dos algoritmos mencionadas em cima.

Para o algoritmo de Épsilon Greedy, foi construída a classe *SelAccaoEGreedy*, no seu construtor, irá receber uma memória, as ações que realiza e o valor do epsilon.

Foram também realizados os métodos:

* Selecionar\_acao(s) que permite uma ação aleatória, neste caso o agente explora o ambiente ou então retorna a última ação;
* max\_accao(s) que num conjunto de ações, retorna aquele que é máxima;
* explorar(s) onde o agente explora o ambiente, segundo um conjunto de ações;

class SelAccaoEGreedy(SelAccao):  
  
 def \_\_init\_\_(self, mem\_aprend, accoes, epsilon = 0.01):  
 self.\_\_mem\_aprend = mem\_aprend  
 self.accoes = accoes  
 self.\_\_epsilon = epsilon  
  
 def selecionar\_accao(self, s):  
 valorAleatorio = random()  
 if valorAleatorio < self.\_\_epsilon:  
 return self.explorar(s)#explorar  
 else:  
 return self.max\_accao(s)#aproveitar  
  
 def max\_accao(self, s):  
 return max(self.accoes, key=lambda a: self.\_\_mem\_aprend.obter(s, a))  
  
   
 def explorar(self, s):  
 return choice(self.accoes

No final, foi implementado também os **processos de decisão de Markov (também conhecido pela sigla PDM) que recebe uma gama e um delta\_max.**

**Foram implementados os seguintes métodos:**

* **utilidade(modelo) que consiste em retornar o valor da utilidade;**
* **útil\_accao(s,a,U,modelo) que consiste em retornar o valor da utilidade a essa respetiva ação;**
* **politica(U,modelo) que retorna o valor da política;**
* **resolver(modelo) que permite resolver este processo de decisão, lendo o valor da utilidade e da política;**

class PDM():  
 def \_\_init\_\_(self, gama, delta\_max):  
 self.\_\_gama = gama  
 self.\_\_delta\_max = delta\_max  
  
 def utilidade(self, modelo):  
 U = {s: 0 for s in modelo.S()}  
 while True:  
 Uant = U.copy()  
 delta = 0  
 for s in modelo.S():  
 U[s] = max(self.util\_accao(s, a, Uant, modelo)  
 for a in modelo.A(s))  
 delta = max(delta, abs(U[s] - Uant[s]))  
 if delta < self.\_\_delta\_max:  
 break  
 return U  
  
 def util\_accao(self, s, a, U, modelo):  
 T = modelo.T  
 R = modelo.R  
 gama = self.\_\_gama  
 return sum(p \* (R(s, a, sn) + self.\_\_gama \* U[sn])  
 for (p, sn) in T(s, a))  
  
 def politica(self, U, modelo):

politicas = {}  
 for s in modelo.S():  
 politicas[s] = max(modelo.A(s), key=lambda a: self.util\_accao(s, a, U, modelo))  
 return politicas

def resolver(self, modelo):  
 U = self.utilidade(modelo)  
 P = self.politica(U, modelo)  
 return U, P

Conclusões

Neste trabalho prático foi implementada toda a teórica que foi explicada neste documento. A implementação deste trabalho consistiu em primeiro lugar estudar a teoria, compreender os seus conceitos e após isso implementar o código.

Bibliografia

Slides fornecidos pelo docente Luís Morgado