Projeto Prático: Simulador de Sistemas Multi-Agente com Aprendizagem

Disciplina: Agentes Autónomos

Professor: Luís Nunes / Rita Peixoto / Sancho Oliveira / Daniel Morgado

Prazos de Entrega:

- **Planeamento: Relatório sumário (max 4 pag.) com diagrama de classes e discussão das opções de desenho (métricas de desempenho em cada ambiente, sincronização, utilização de threads, recompensa intrínseca / extrínseca, opções de sensores, … etc), ~~10-nov~~ 30-nov, 8:00, entrega no *Moodle*;**
- **Apresentação 15 a 19-dez (a agendar no *Moodle*);**
- **Relatório a 12-jan. (23:59, entrega no *Moodle*)**

I. Introdução e Objetivos

Este projeto prático visa a construção de um **simulador flexível em Python** capaz de modelar e executar cenários típicos para sistemas multi-agente (**SMA**). O simulador deverá permitir a **integração e avaliação de diferentes arquiteturas e estratégias de agentes autónomos** em ambientes dinâmicos e cooperativos/competitivos.

Os objetivos principais são:

1. **Desenvolver uma arquitetura de simulação modular** para problemas de SMA.
2. **Implementar, pelo menos, dois problemas clássicos** de SMA (**Farol** e **Recoleção/Foraging** ou **Labirinto**).
3. Alguns agentes têm políticas fixas (pré-programadas) mas outros devem poder funcionar em modo de **Aprendizagem** (e.g., *reinforcement learning*, algoritmos evolucionários, estratégias de adaptação *ad-hoc*) e em modo de **Teste/Avaliação**.
4. **Deve ser possível avaliar e registar** o desempenho das estratégias dos agentes face a métricas de sistema predefinidas (número de acções até terminar o episódio, recompensa média por episódio, recompensa descontada).

II. Requisitos Técnicos e Funcionais

O projeto deverá ser inteiramente desenvolvido em **Python** e deve cumprir os seguintes requisitos:

A. Arquitetura do Simulador

1. **Motor de Simulação:** Deve gerir o ciclo de tempo (passos de simulação) e a ordem de execução das ações dos agentes. O interface de `Simulador` deve suportar:
    - `cria(nome_do_ficheiro_parametros: string): MotorDeSimulacao`
    - `listaAgentes(): Agente[]`
    - `executa()`
2. **Definição do Ambiente:** Deve ser capaz de **representar** o ambiente (e.g., grelha 2D, grafo) e os seus elementos dinâmicos (recursos, obstáculos, outros agentes). O interface base do `Ambiente` deve suportar métodos como:
    - `observacaoPara(agente: Agente): Observacao`
    - `atualizacao()`
    - `agir(accao: Acção, agente: Agente)`
3. **Módulo de Agente:** Definir uma interface `Agente` que tenha métodos como:
    - `Agente cria(nome_do_ficheiro_parametros: string)`
    - `observação(obs: Observação)`
    - `Accao age()`
    - `avaliacaoEstadoAtual(recompensa: double)`
    - `instala(sensor: Sensor)`
    - `comunica(mensagem: String, de_agente: Agente)`

Os agentes são, preferivelmente, Threads independentes, os métodos comuns para a comunicação podem ser definidos numa classe abstrata. A interação entre Agentes e Ambiente deve respeitar o interface e fazer-se exclusivamente através deste.

Nem todas as operações precisam de implementação para estes ambientes funcionarem corretamente, mas devem ser incluídas para ter um interface completo e para permitir facilmente extensões nos trabalhos mais avançados..

B. Implementação dos Problemas

O simulador deve ser capaz de instanciar, pelo menos, duas das seguintes configurações:

1. **Farol:**
    - **Ambiente:** Espaço 2D com um farol. Inicialmente sem obstáculos, a versão final deve poder ter obstáculos
    - **Agentes**: os agentes observam apenas a direção em que está o farol e o seu objetivo é ir para o farol.
2. **Problema da Recoleção (*Foraging*):**
    - **Ambiente:** Uma grelha 2D com células que podem conter **recursos** (com diferentes "valores"), **ninhos/pontos de entrega** e **obstáculos**.
    - **Agentes:** Devem ter a capacidade de se **movimentar**, recolher recursos e depositá-los no ninho.
    - **Objetivo do Sistema:** Maximizar a quantidade total de recursos recolhidos no tempo limite (cooperação).
3. **Problema do Labirinto (*Maze*):**

- **Ambiente:** Uma grelha 2D com **paredes/obstáculos**, um **ponto de partida** e um **ponto de chegada**.
- **Agentes:** Devem **movimentar-se** e evitar obstáculos. Pode ser um sistema multi-agente (e.g., 2 agentes a procurar a saída, um a ajudar o outro).
- **Objetivo do Sistema:** Encontrar o caminho mais curto ou a saída no menor número de passos (cooperação ou concorrência se houver recursos limitados no caminho).

Deve ser possível ter um modo de visualização simples (janela com espaço 2D em que se movimentam agentes, e onde estão indicados os objetivos / recursos / ninho).

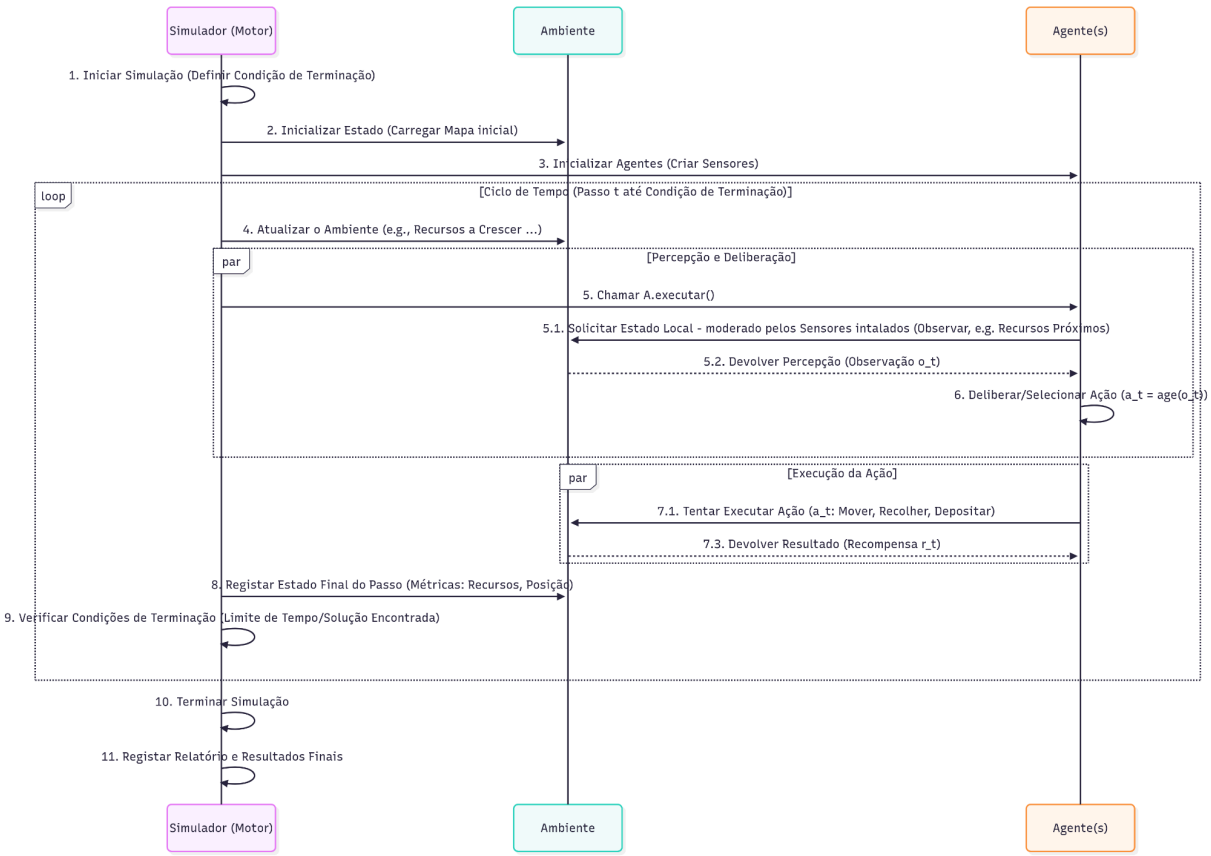

Figura 1. Exemplo de diagrama de sequência para o desenrolar do programa. Os agentes podem ser linhas de execução independentes, desde que sincronizadas nos momentos certos pelo simulador. No limite os agentes seriam programas independentes com ligações por socket ao ambiente, mas isso não é exigível neste trabalho. A paralelização das acções dos agentes no ambiente, também não é necessária a este nível.

C. Modos de Operação

Os agentes com capacidade de aprendizagem devem suportar e diferenciar dois modos de funcionamento:

1. **Modo de Aprendizagem (*Learning Mode*):**
    - A **política do agente pode ser modificada** (e.g., através de algoritmos de *Q-learning* ou algoritmos genéticos) durante a simulação.
    - Deve **registar** dados de desempenho por *episódio* para posterior análise da curva de aprendizagem.
2. **Modo de Teste (*Test Mode*):**
    - Toda a simulação / episódio deve ser executada com uma **política de agente fixa/pré-treinada**.
    - O foco é na **avaliação do desempenho** (e.g., número de acções médio até à solução, taxa de sucesso, recompensa média, recompensa descontada, …)

III. Entrega e Avaliação

O trabalho será avaliado com base nos seguintes pontos:

A. Apresentação e Implementação - partilha de link no github (50%)

- **Clareza da apresentação**
- **Implementação dos Problemas:** Correta modelação dos problemas e modos de visualização.
- **Modularidade e Design:** Uso de classes e abstrações (e.g., separação clara entre `Ambiente`, `Agente` e `Simulador`) - idealmente deveria ser possível usar ambientes e/ou agentes de outros grupos no seu simulador, sem alterações.
- **Correta Implementação dos Modos:** Funcionalidade distinta para os modos de Aprendizagem e Teste e registo de resultados.
- **Qualidade do Código:** Clareza, comentários.

B. Relatório (50%)

O relatório deve incluir:

1. **Descrição da Arquitetura:** Diagrama de classes e explicação da estrutura de módulos.
2. **Implementação dos Agentes:** Detalhe das estratégias implementadas
3. **Análise de Resultados e comparação para os vários tipos de agente nos vários problemas:**
    - **Curva de Aprendizagem:** Apresentar a evolução do desempenho no **Modo de Aprendizagem**.
    - **Avaliação em Modo de Teste** (usando métricas como taxa de sucesso, passos médios, etc.).

4. Obrigatório incluir **link github para projeto com README** adequado a testar o seu funcionamento

---

IV. Questões relevantes a considerar:

- Como irá modelar a percepção do agente em cada ambiente (e.g., campo de visão limitado, percepção global)?
- Quais são as métricas de desempenho mais relevantes para cada problema?
- Será que os algoritmos usados reagem do mesmo modo a diferentes métricas?