universidade de aveiro
departamento de eletrónica, telecomunicações e informática

Inteligência Artificial

Desenvolvimento de um agente autónomo para o jogo Snake

Danilo Silva 113384

Tomás Fernandes 112981

João Gaspar 114514

IA 2024-2025

Arquitetura do Agente – 3 Camadas

Camada Reativa

Camada responsável por receber o estado atual do ambiente e atualizar o estado interno do agente.

- O ficheiro ***student.py*** recebe o estado atual do ambiente via WebSocket.
- A classe ***Snake*** processa esses dados, extraindo a informação e atualizando o estado interno no método ***update.***
- Estes dados são depois utilizados, em cada iteração, nas outras camadas, de modo ao agente poder definir objetivos e **tomar decisões imediatas** como:
  - Evitar colisões com paredes;
  - Evitar colisões com outro agente em multiplayer;
  - Evitar colisões com o próprio agente.

Camada de Reações Intermédias

Camada de ligação entre reações imediatas e reações com planeamento a longo prazo.

- A classe ***SnakeDomain*** utiliza os métodos ***snake_in_sight()*** e ***check_food_in_sight()*** para redefinir novos objetivos com base no estado atual do ambiente.
- Se uma ou mais frutas estiverem no seu campo de visão, o agente **descarta o plano atual**, e **define um novo plano** em que o primeiro objetivo é a fruta mais próxima.
- Se outro agente estiver no seu campo de visão, o agente **reavalia o plano** e define outro, caso necessário, mantendo-se afastado para evitar colisões e movimentos bruscos.

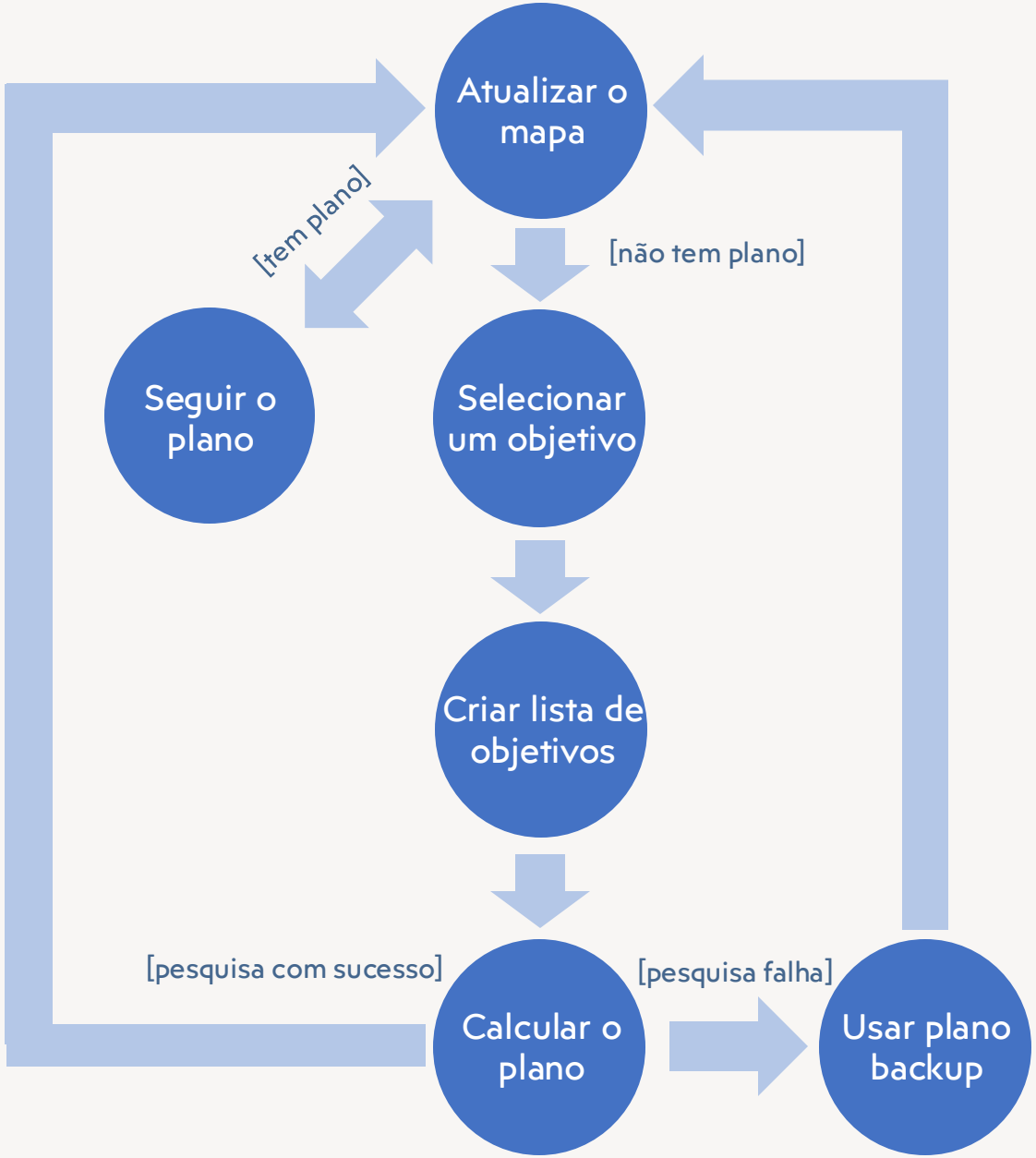
Camada Deliberativa

Camada responsável pelo planeamento a longo prazo, definindo objetivos com base num algoritmo de pesquisa.

- A classe ***SnakeDomain*** é responsável por definir a lista de objetivos utilizando a classe ***MultiObjectiveSearch***.
- Ao escolher um determinado objetivo, o agente define como **destino final** a própria cauda, garantindo assim que consegue ir ao destino e também sair de lá, para que não realize movimentos que mais tarde possam levar a colisões.
- Desta maneira, o agente procura criar **planos a longo prazo**, com o objetivo de evitar situações em que não haja uma solução.
- Além disto, o agente também toma decisões com base nos seus atributos:
  - Caso ***traverse=true*** e ***sight>5***, o agente evita Super Foods, e guarda as posiões para mais tarde.

Algoritmo

1. Atualizar o mapa (Mapeamento do Ambiente):
   - O agente atualiza o mapa com as posições exploradas e inexploradas através da ***sight***, ou seja, para cada posição na sight, a posição é removida dos sítios que ainda não visitou e adicionada aos sítios que já visitou. As posições exploradas são geridas por uma queue, cujo tamanho máximo corresponde a metade das posições do mapa.
2. Selecionar um objetivo/Seguir o plano:
   - Se existir um plano já definido, o agente continua a **seguir o plano**.
   - Caso apareça uma **fruta** ou uma **cobra** na sua visão, o agente recalcula um novo plano para seguir.
   - Se não existir nenhum plano atual, o agente **seleciona um novo objetivo**. Este objetivo é escolhido tendo em conta a posição com **maior densidade de regiões inexploradas** pelo agente.
3. Criar lista de objetivos (Pesquisa MultiObjetivo):
   - Após definir o objetivo a ser alcançado (uma fruta ou uma posição no mapa), o agente cria uma lista de objetivos para efetuar a pesquisa em árvore.
   - A lista dos objetivos é composta pelo objetivo principal (selecionado pelo agente) e pela cauda.
4. Calcular o plano (Pesquisa gulosa):
   - Com a lista dos objetivos bem definida, o agente efetua uma pesquisa usando o algoritmo de **pesquisa gulosa** para chegar até à posição da cauda, passando pelo objetivo principal.
   - Para melhorar a eficiência da pesquisa foi implementada uma **heurística** baseada na distância entre a posição da cabeça da cobra e o objetivo. Além disso, é atribuída uma penalidade a posições próximas de outras cobras.
   - Ao obter o plano, o agente utiliza esse plano até chegar ao objetivo principal e guarda o caminho restante (até à cauda) para plano backup.
   - O plano de backup é utilizado em último caso, isto é, quando a pesquisa em árvore não encontra solução.



Benchmark

Para testar o agente foi realizado o seguinte teste:

- O programa foi executado 10 vezes em single player;
- Um timeout de 3000 steps foi definido;
- Para comparação de pontuação, todos os testes foram executados no mesmo ambiente, ou seja, usando o mesmo tamanho do mapa, o mesmo número de frutas no mapa, etc. A única diferença entre os testes foi o mapa;
- Foi guardada tanto a pontuação como o step no momento do término de cada execução.

Resultados:

Testes	Teste1	Teste2	Teste3	Teste4	Teste5	Teste6	Teste7	Teste8	Teste9	Teste10
Pontuação	121	80	96	149	62	112	101	111	82	107
Step	3000	3000	3000	3000	3000	3000	3000	3000	3000	3000

Pontuação média: 102.1

Nota: O step em cada teste mostra que o agente chegou até ao final da execução (step 3000) em todos os testes, algo que na entrega passada não foi adquirido.

Conclusão

1. Escolha do algoritmo:
   - Foi utilizado Greedy Search com heurística personalizada.
   - A estratégia multiobjetivo mostrou-se eficiente para lidar com prioridades dinâmicas.
   - A inclusão da cauda como destino final garante maior segurança, permitindo que o agente alcance objetivos sem se prender.
2. Aspetos Positivos:
   - Eficiência: O uso de Greedy Search permite decisões rápidas e focadas no objetivo.
   - Segurança: A penalização de proximidade com agentes reduz o risco de colisões e a cauda como objetivo final evita que o agente se prenda em caminhos sem saída.
   - Adaptação: A heurística adapta-se às condições do meio, como por exemplo, a aproximação de um agente, a mudança no modo 'traverse', etc.
   - Robustez: A estratégia de evitar Super Foods e guardar para o final do jogo prolonga a sobrevivência em cenários complexos.
3. Possíveis melhorias:
   - Evitar passar por regiões já exploradas ao calcular um plano.
   - Refinar o gerenciamento do mapa para otimizar a densidade de regiões inexploradas e melhorar o algoritmo de procura.
   - Implementar múltiplas heurísticas adaptativas para diferentes cenários no jogo.

Apesar de algumas melhorias potenciais, a estratégia implementada é bem sucedida, eficiente e adequada para o contexto do problema, balanceando segurança e adaptabilidade