Trabalho - Sistema Multiagente

INE5430 - Inteligência Artificial

Matheus Dhanyel Roque e Pedro Henrique Aquino Silva

Julho de 2022

Este trabalho consistia na alteração do problema de Robôs Mineradores em Jason para que eles conseguissem trocar informações e colaborar no problema da coleta. Infelizmente, os alunos somente conseguiram implementar os primeiros três itens: fazer com que os *collectors* colaborassem compartilhando as posições dos recursos requisitados pelo *builder*, avisando o esgotamento de recursos em posições já conhecidas, e permitir aos *collectors* armazenarem informações sobre onde se encontram os próximos recursos que o *builder* requisitar.

Para implementar o compartilhamento de informações sobre posições de recursos, foi criado uma nova crença chamada `resource` que quando recebida, determinava o seguinte plano:

```
+resource(X,Y,R)
  : resource_needed(R) & not my_pos(X,Y) & checking_cells
  <- .print("I shall go fetch resources in ", X, " ,", Y);
  !go(X,Y).

+!go(X,Y) : resource(X,Y,R) & resource_needed(R)
   <- move_towards(X,Y);
      .wait(100);
      !go(X,Y).
```

Com esta nova crença, ao chegar em um recurso desejado, os coletores avisam aos demais a posição do recurso utilizando o `broadcast`:

```
@cfr[atomic]
+!check_for_resources
   :  resource_needed(R) & found(R) & my_pos(X,Y)
   <-  !stop_checking;
    .print("Announcing current resource found in ", resource(X,Y,R), " to
   others");
    .broadcast(tell, resource(X,Y,R));
    !take(R,boss);
  !continue_mine.
```

Dado que o plano para `resource` requer que o estado do coletor contenha `checking_cells`, um coletor que já está em alguma atividade de coleta não altera sua atividade ao receber uma nova crença de `resource`. Esta implementação satisfaz ao item (a) do enunciado.

Para o item (b), poucas alterações foram feitas em relação à questão anterior. Quando um robô encontra o recurso desejado, ele o leva até a posição onde se encontra o *builder*. Após entregar o recurso, o coletor ativa novamente o objetivo `check_for_resource`, e retorna à sua posição anterior (na qual se encontra ou encontrava o recurso). Caso nesta posição ele não encontre mais o recurso necessário, ou seja, perceba o esgotamento do recurso, ele avisa aos demais robôs que a crença `resource` relacionada àquela posição deve ser removida. No código em Jason, isto é implementado como:

```
@cfr2[atomic]
+!check_for_resources
   :  resource_needed(R) & not found(R) & my_pos(X,Y)
```

```
    <- .broadcast(untell, resource(X,Y,R));
       .print("Removed resource location at ", resource(X,Y,R));
       .wait(100)
       move_to(next_cell).
```

No item (c), é pedido que os robôs possam guardar informações de todos os locais onde há recursos. Isto é implementado de forma similar ao item (a), enviando uma nova crença `resource` para os demais coletores, e continuando a procura pelo recurso esperado no momento:

```
+!check_for_resources
  : resource_needed(R) & found(S) & my_pos(X,Y)
  <- .broadcast(tell, resource(X,Y,S));
    .print("Announcing future resource found in ", resource(X,Y,S), " to
   others");
  .wait(100);
  move_to(next_cell).
```

Os itens (d) e (e) não foram implementados, uma vez que diversas demandas dos estudantes os impediram de dedicar mais tempo à este trabalho. Contudo, algumas estratégias foram pensadas ou testadas, sem sucesso na implementação, mas que serão comentadas brevemente a seguir.

No item (d), imaginou-se que seria possível adicionar ao plano da crença `resource` uma outra crença `previous` que armazene a posição original do coletor ao ativar o plano. Desta forma, ao esgotar o recurso indicado pelo plano da crença `resource` recebida, o coletor poderia simplesmente recuperar a posição (x, y) da posição `previous` e retornar à ela. Isto seria implementado da seguinte forma:

```
+resource(X,Y,R)
  : resource_needed(R) & not my_pos(X,Y) & my_pos(Z,W) & checking_cells
  <- +pos(previous, Z, W);
    .print("I shall go fetch resources in ", X, " ,", Y);
  !go(X,Y).
```

Além disso ainda seria necessário implementar algum mecanismo para ao perceber o esgotamento do recurso, verificar se existe uma posição `previous` e ativar `!go(previous)`.

No item (e), seria necessário encontrar algum meio de avaliar todas as distâncias entre as posições apontadas por crenças `resource` e o *builder*. Isso poderia ser feito de forma similar ao que é descrito no exemplo *Gold Miners* do livro de Jason disponibilizado no Moodle. Infelizmente, não conseguimos implementar um protótipo satisfatório para este item.