Robôs Mineiradores - Sistema Multiagentes

Artur Barichello (16200636)
Bernardo Schmidt Farias (19100519)

27 de junho de 2022

Resumo

Esse trabalho se propõe a desenvolvedor um sistema orientado a agentes para resolver o problema dos Robôs Mineiradores. Como base, foi utilizado o exemplo disponibilizado pelo Moodle, entretanto, com algumas alterações.

1 Algumas alterações no ambiente

1.1 Número de robôs

Originalmente, eram utlizados 3 robôs no código base, aqui, optou-se por usar 4. Essa informação foi alterada no documento `planetEnv.java`

Listing 1: Declaração do novo agente

```
//...
col2 = new int[2];
col2[X] = middle;
col2[Y] = middle;
col3 = new int[2];
col3[X] = middle;
col3[Y] = middle;
// new agent
col4 = new int[2];
col4[X] = middle;
col4[Y] = middle;
```

2 Quantidade de recursos

Também foi alterada a partir do código base, a quantidade de recursos e a disponibilidade desses recursos no campo. Foram alterados os seguintes parâmetros: quantidade de recurso num mesmo nodo, o número de nodos no mapa e a quantidade de recurso necessária, essas alterações feitas nos arquivos `planetEnv.java` e `site.java`:

Listing 2: Alterações nos recursos `site.java`

```
public Site() {

    r1needed = 40;
    r2needed = 50;
    r3needed = 60;

    r1store = 0;
    r2store = 0;
    r3store = 0;

    completed = false;
}
```

Listing 3: Alterações nos recursos `planetEnv.java`

```
for (int i = 0; i < 15; i++) {
    //...
    planet[x][y] = new Resource(1, 3);
    resourcemap[x][y] = true;
}

for (int i = 0; i < 10; i++) {
    //...
    planet[x][y] = new Resource(2, 6);
    resourcemap[x][y] = true;
}

for (int i = 0; i < 20; i++) {
    //...
    planet[x][y] = new Resource(3, 3);
```

```
            resourcemap[x][y] = true;
        }
```

3 Solução do problema

3.1 Comunicação entre robôs

Foi necessário implementar uma maneira de estabelecer comunicação entre os robôs. Essa comunicação será útil para que os robôs se notifiquem entre si sobre a descoberta ou esgotamento de recursos.

Primeiro, foi implementado a o `check_for_resources`, que é responsável por comunicar aos outros agentes que o recurso foi encontrado e onde ele se encontra.

Listing 4: Informando a descoberta de um recurso

```
+!check_for_resources
    :  resource_needed(R) & found(R) & position(X,Y)
    <- !stop_search;
            .broadcast(tell,resource(X,Y,R));
       !take(R,boss);
       !continue.
```

Foi necessário também implementar uma maneira de os agentes comunicarem o **esgotamento** de um recurso, avisando aos outros robôs que aquele recurso não está mais disponível.

Listing 5: Informando o esgotamento de um recurso

```
+!check_for_resources
    :  resource_needed(R) & not found(R) & position(X,Y)
    <- move_to(next_cell);
    .broadcast(untell,resource(X,Y,R)).
```

3.2 Armazenamento de informações

Foi implementado também uma forma de os robôs armazenarem informações sobre recursos localizados, mesmo que não estejam sendo procurados, de modo que outros agentes possam encontrá-lo quando necessário.

Listing 6: Informando a descoberta de um recurso diferente

```
+!check_for_resources
    :   resource_needed(R) & found(S) & position(X,Y)
    <-  .broadcast(tell,resource(X,Y,S));
            .wait(100);
        move_to(next_cell)
```

3.3 Retornando a posição inicial após ajudar na coleta

Também foi implementada uma maneira de o agente, após colaborar na coleta de um recurso, volte a navegar "aleatoriamente"a partir do ponto que estava antes de ser chamado para auxiliar na coleta.

Listing 7: Retornando a posição inicial após ajudar na coleta

```
+!check_for_resources
    :   resource_needed(R) & found(R) & position(X,Y)
    <-  !stop_search;
          .broadcast(tell,resource(X,Y,R));
           .print("Agent calling for mining help at: ", X, " ", Y);
        !take(R,boss);
        .print("Agent is going back to the position where it was
                called for help: ", X, " ", Y);
        !go(X,Y);
        !continue.
```