UNICAMP
Universidade Estadual de Campinas

FEEC
Faculdade de Engenharia Elétrica e de Computação

Aluno: Mateus Neves Barreto
R.A.: 142358
Disciplina: IA006
Professor: Ricardo R. Gudwin

Relatório – Aula 3

1 Atividade 1

A seguir são apresentados os "*prints*" da criação da criatuta com a regra "move-to-food".

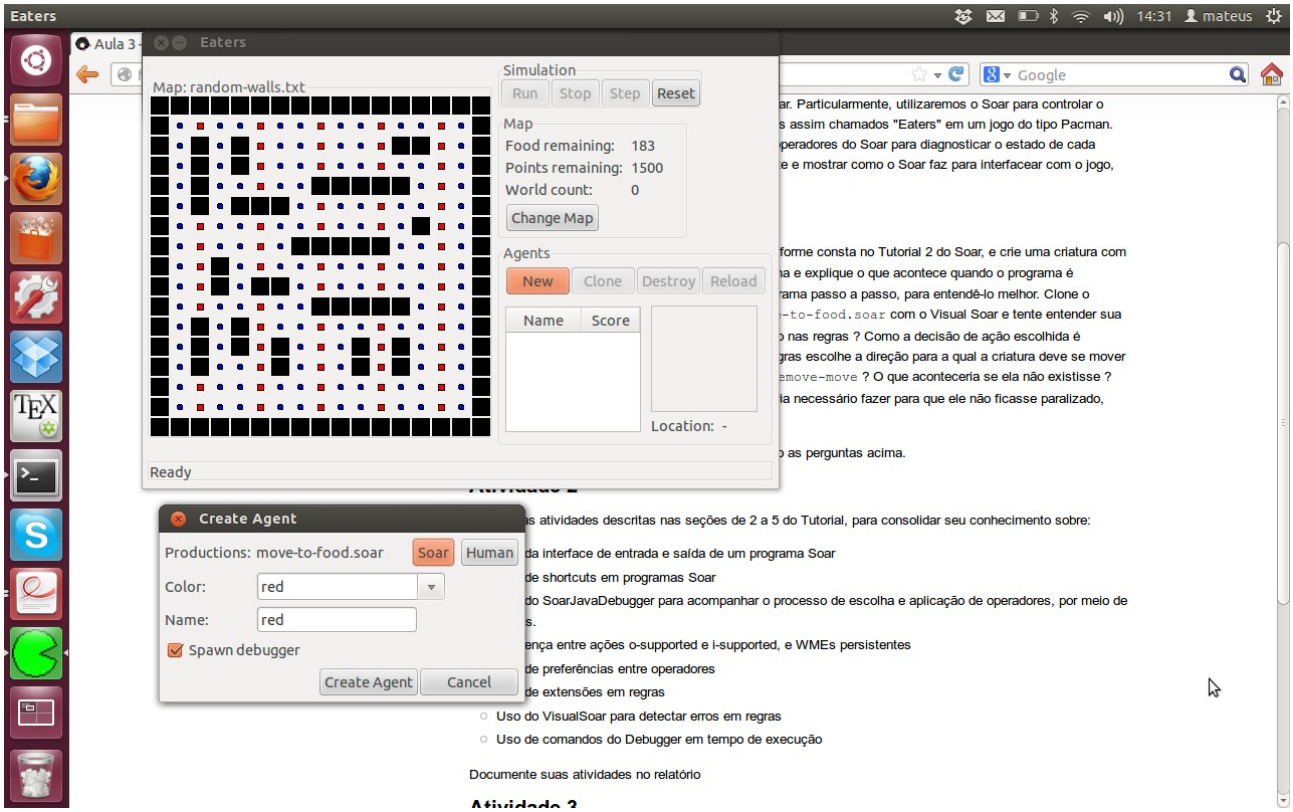

Figura 1 – Criando agente "move-to-food".

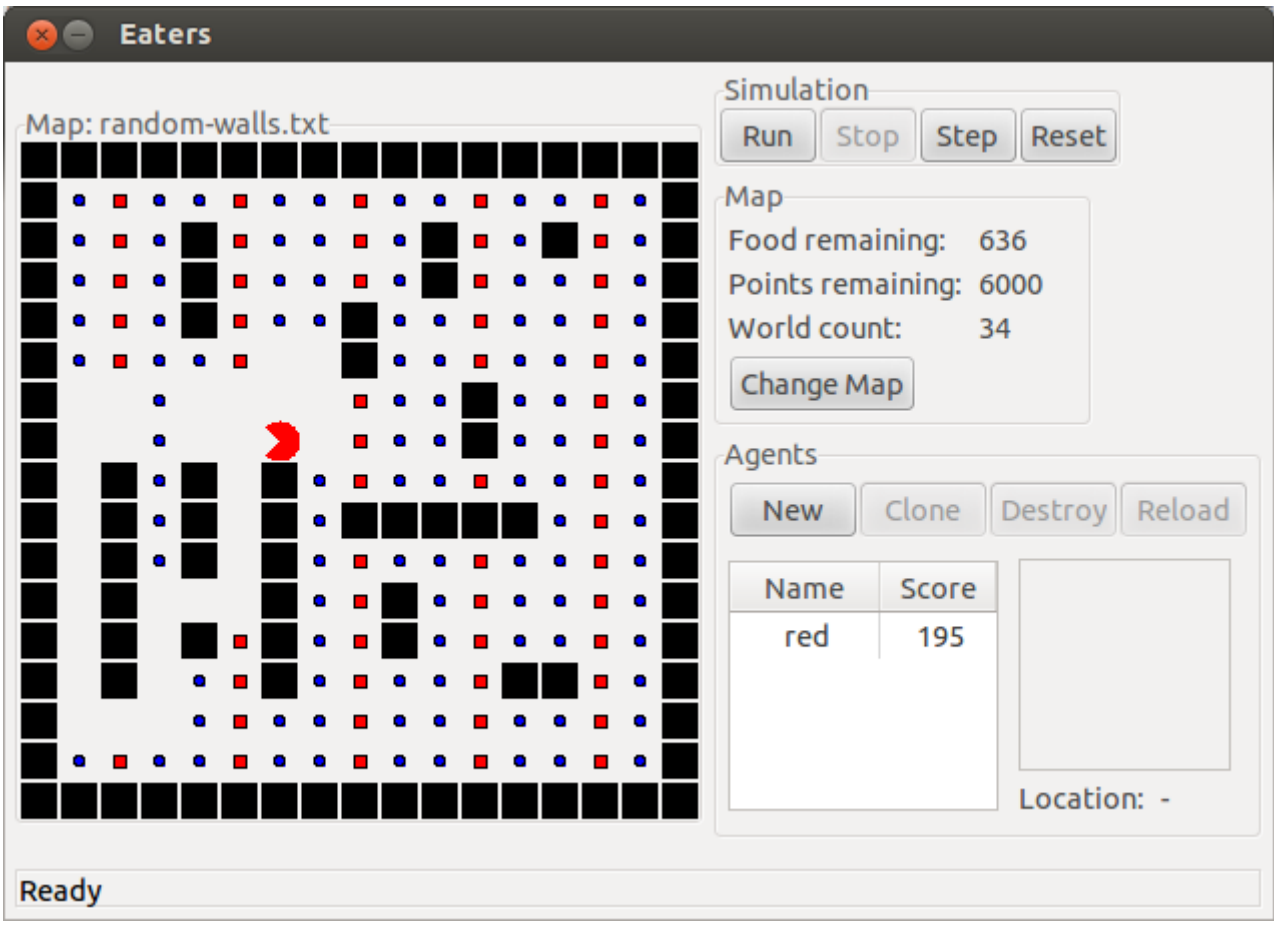

Figura 2 – Executando agente "move-to-food".

Pode-se observar na Figura 2, que o agente está cercado de espaços sem "comida". Este estado gerado, impossibilita a escolha de uma regra para as regras propostas no arquivo "move-to-food.soar", com isso o agente fica estagnado em sua posição não conseguindo mais buscar "comida".

As regras presentes no arquivo "move-to-food.soar" apenas consideram a criação de propostas se o agente possuir uma "comida comum" ou uma "comida de bônus" em sua vizinhança. A direção que o agente deve tomar caso exista a comida é escolhida randomicamente, devido ao operador ser seguido pelo sinal de igualdade "=". Por exemplo, na existência de apenas uma comida na vizinhança, o agente tomará a direção desta comida, porém na existência de duas ou mais comidas na vizinhança, a direção que o agente tomará será randômica presente nas direções das comidas vizinhas.

A função da regra "apply*move-to-food*remove-move" é remover o operador da memória de trabalho que já foi "executado". Aparentemente a remoção desta regra não causa efeito direto na execução do agente, porém é certo que sua memória de trabalho tenderá a crescer de acordo com o tempo de execução, diferente da execução do agente com esta regra presente.

A estratégia para que o agente não fique paralisado após um tempo e a inserção do "estado empty" na regra de proposição, considerando-o com uma preferência menor de escolha. Isso evitaria que o agente estagnasse após um tempo.

2 Atividade 2

2.1 move-to-north

A Figura 3 apresenta o código soar para a criação das regras que o agente se move apenas na direção norte. Todas as proposições e aplicações das ações para os agentes estão sendo realizadas via escrita e leitura na memória de trabalho, pelos comandos de estados de io.imput e io.output, entrada e saída respectivamente.

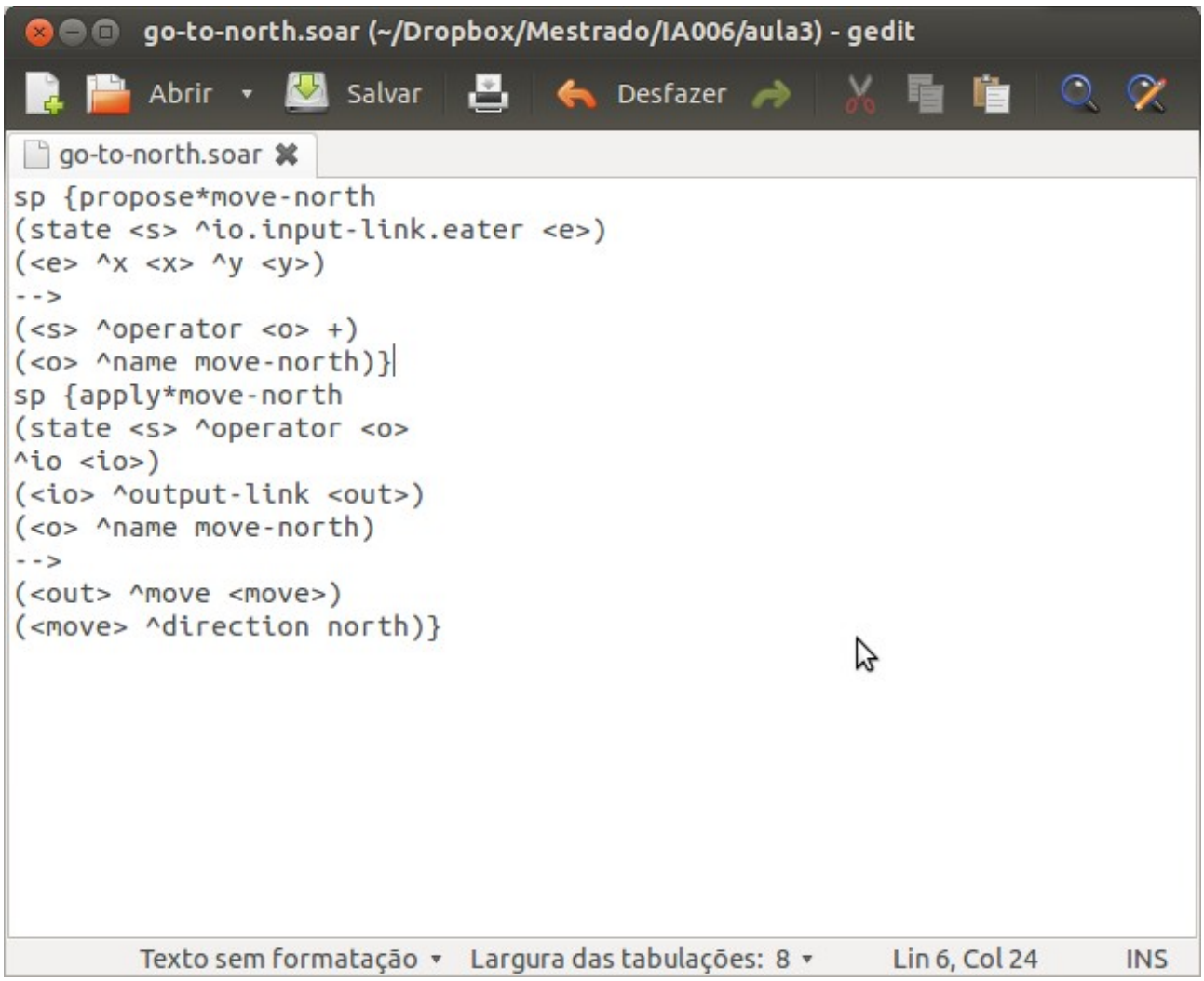

Figura 3 – Agente para o norte.

A Figura 4 apresenta a analise da execução do código apresentado na Figura 3 até a execução da regra remove-move.

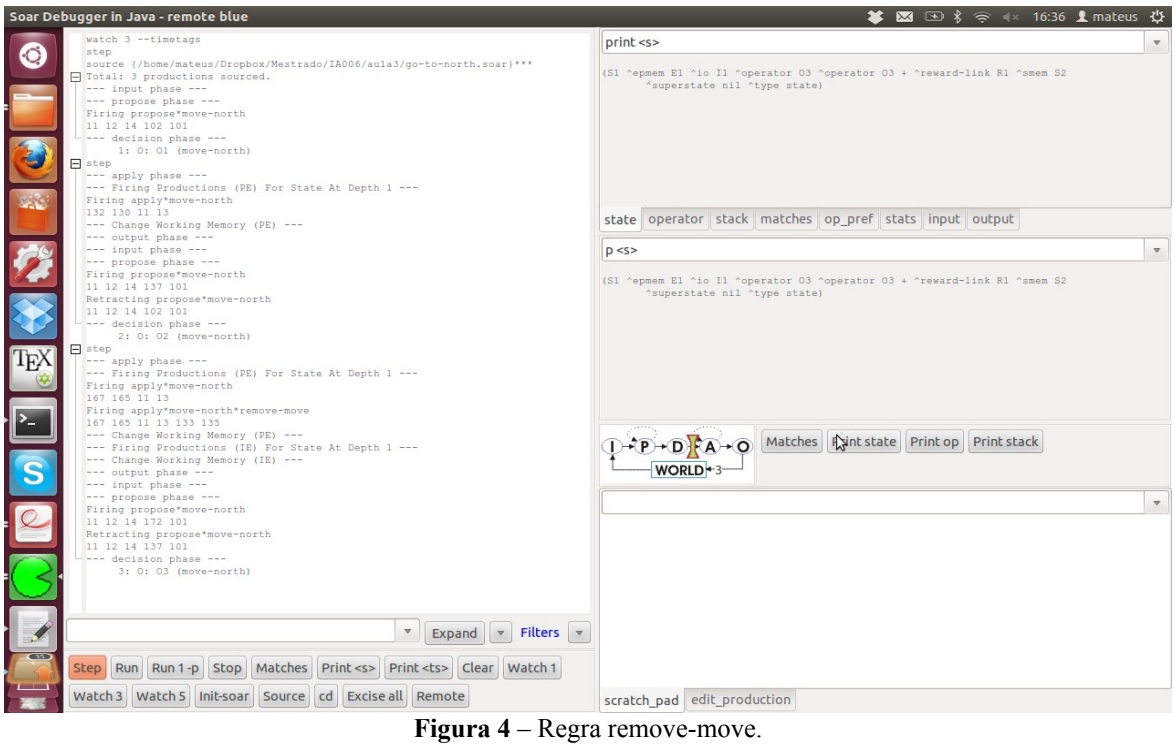

Figura 4 – Regra remove-move.

As Figuras 5 e 6 apresentam a utilização do break pointer no canto inferior direito.

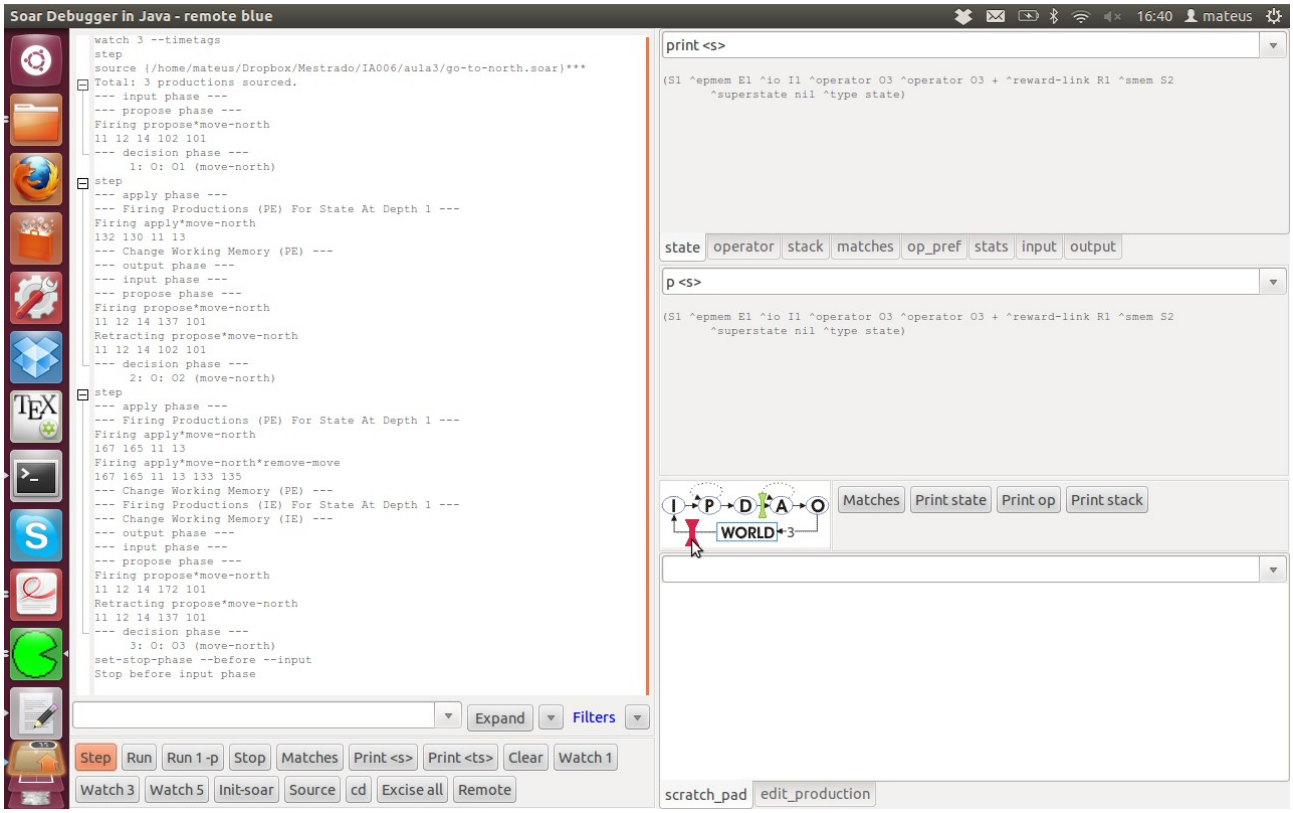

Figura 5 – Utilizando o break pointer - 1.

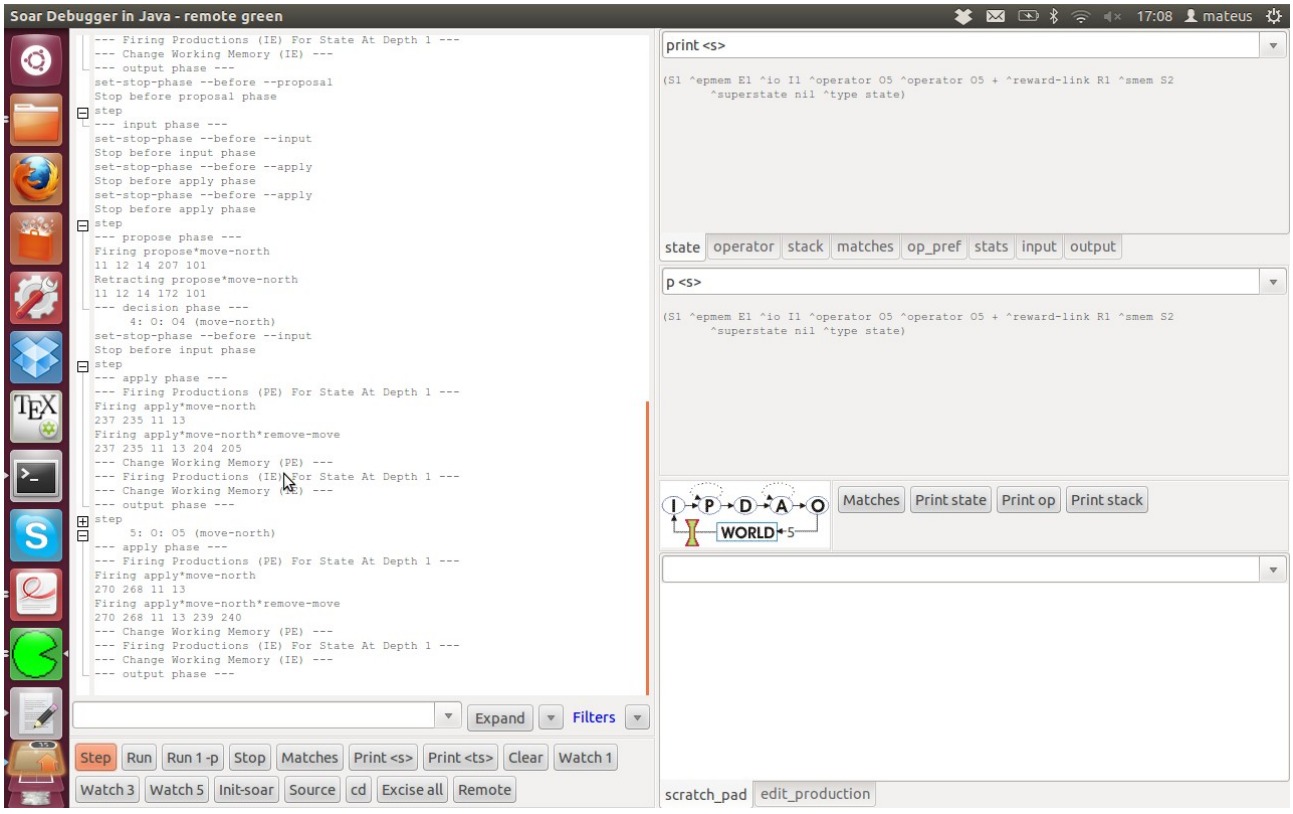

Figura 6 – Utilizando o break pointer – 2.

2.2 move-to-food

Neste sub-tópico serão apresentadas as tarefas realizadas para as ações nos agentes para os mesmos se moverem em direção as comidas, comuns e de bônus. A Figura 7 apresenta o código completo

contendos as resgras para realizar tal acão.

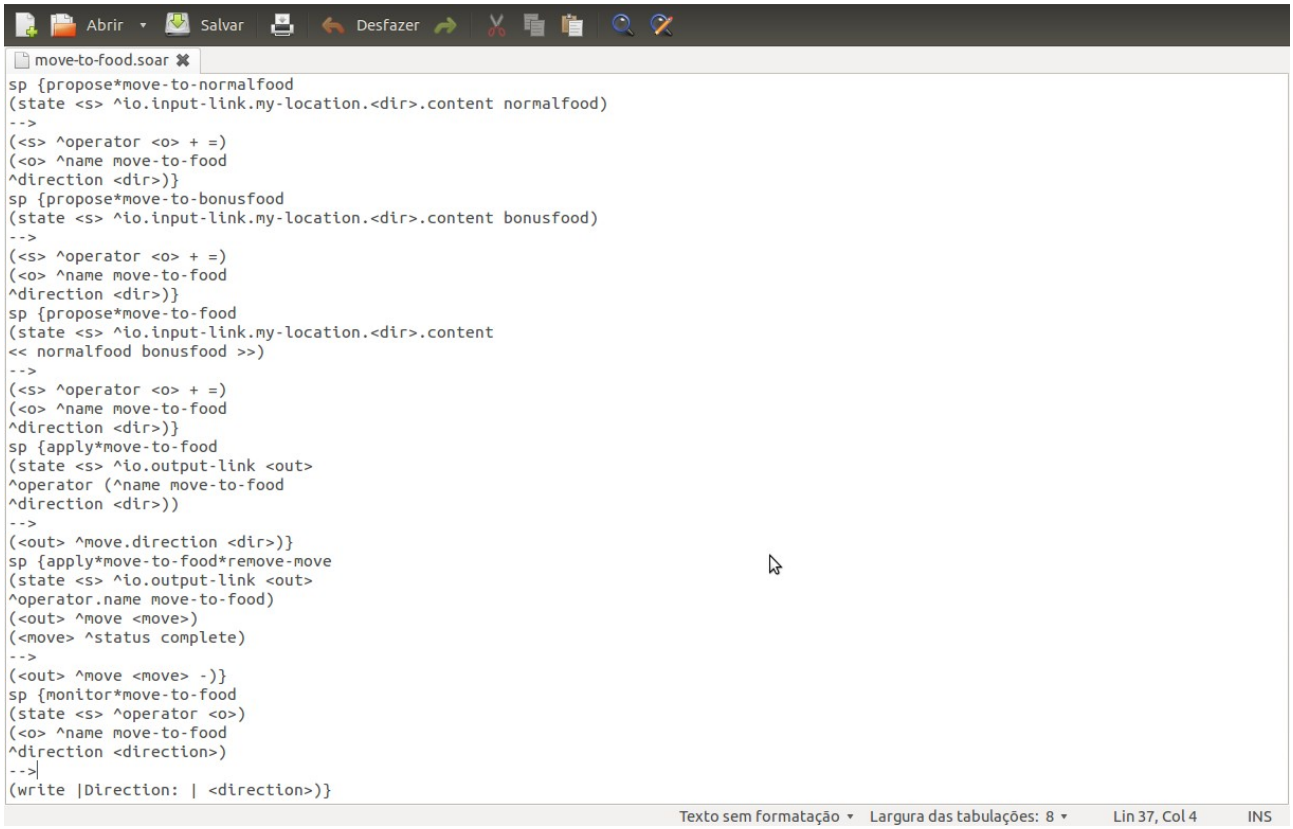

Figura 7 – Código move-to-food.

A Figura 8 apresenta a execução do agente com o código move-to-food.

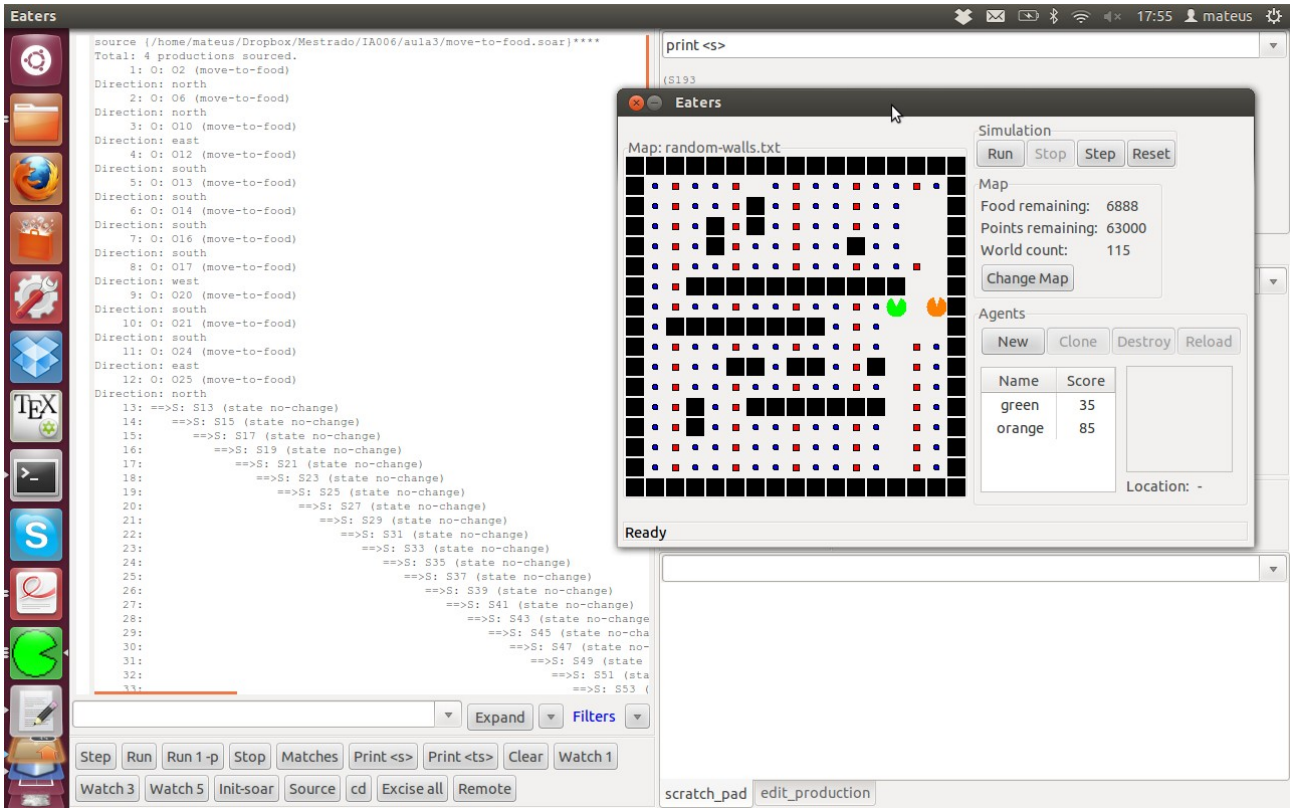

Figura 8 – Execução move-to-food.

Pode-se observar na Figura 9 a impressão do operador escolhido e sua respectiva direção, pelo comando “(write |Direction: | <direction>)”.

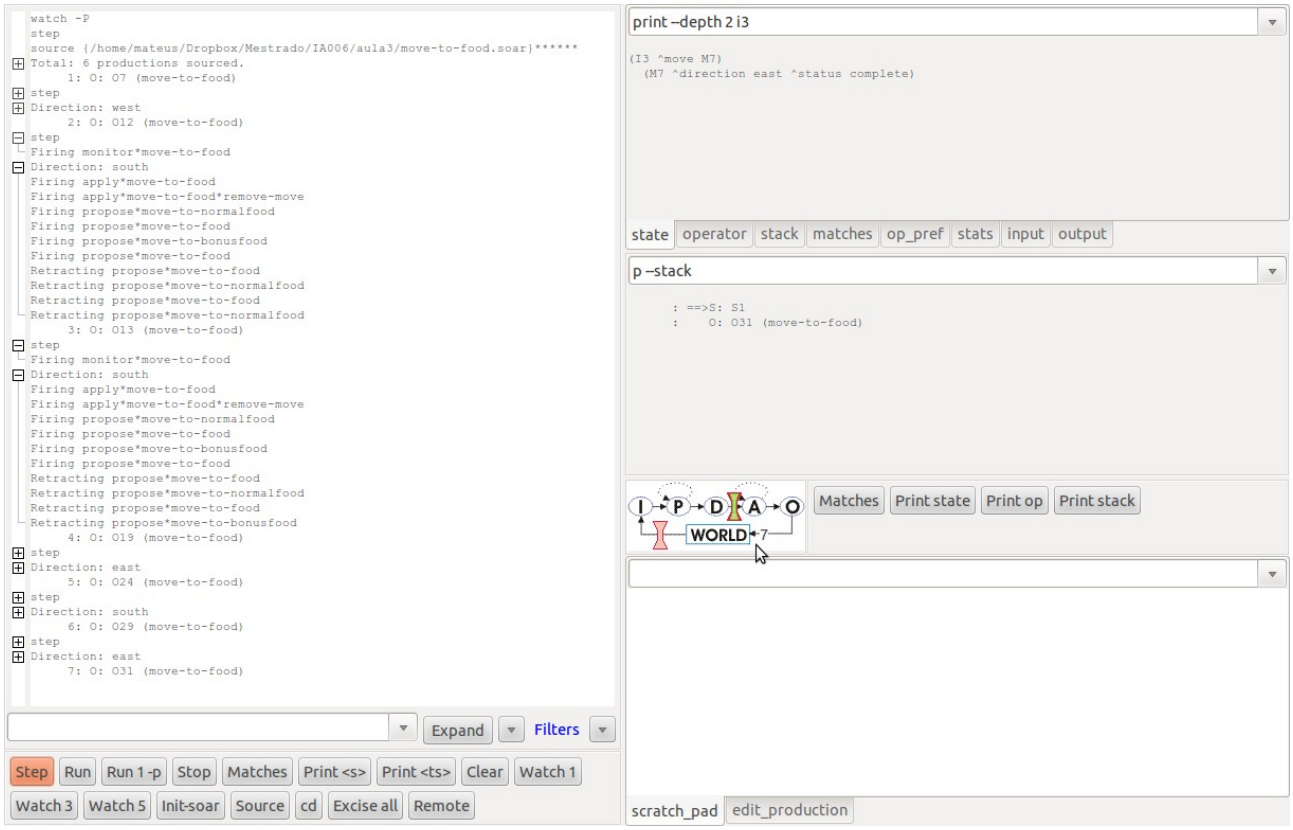

Figura 9 – Impressão da direção do operador selecionado.

A Figura 10 apresenta a impressão da direção do operador selecionado concatenado com a variável “type”/tipo da comida, esse impressão é feita com o comando “(write (crlf) | Propose move | <dir> |, for | <type>)”.

Figura 10 – Impressão da direção e o tipo do operador selecionado.

2.3 Debugging commands

Neste sub-tópico é apresentado o debbuging da execução do agente com o código move-to-food.soar. Entre as análises, pode-se observar prints, com profundidades da árvore de execução; analises da memória de trabalho pelo comando wmes; prints de posições específicas da memória; matches: listas de regras prontas para serem executadas separadas por três resultados, o-assertions (que serão operadores), i-assertions (que criarão os argumentos de suporte para os operadores) e retractions (que excluirão os argumentos de suporte). Nas Figuras 11 e 12 pode-se observar estas analises.

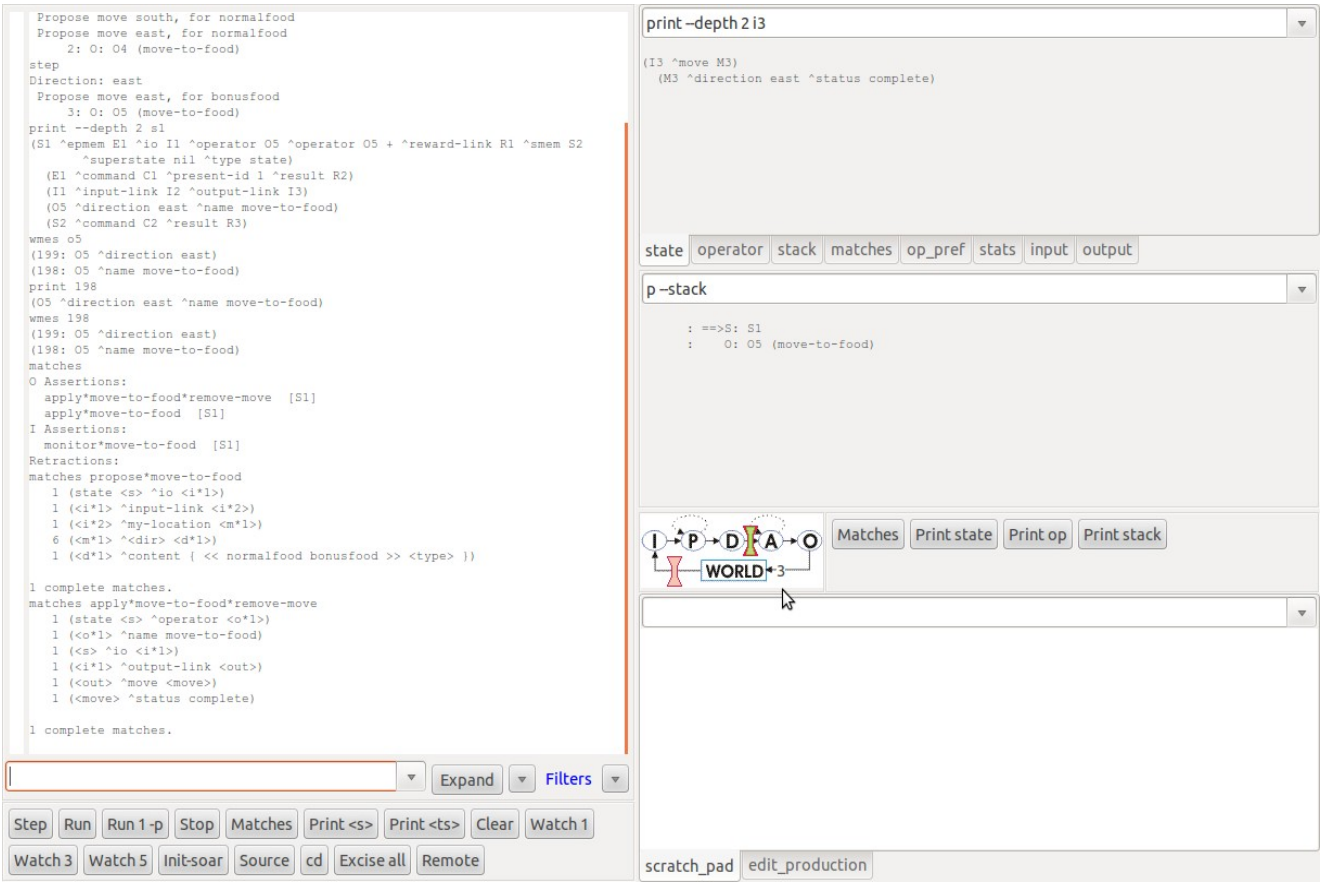

Figura 11 – Debugging commands- 1.

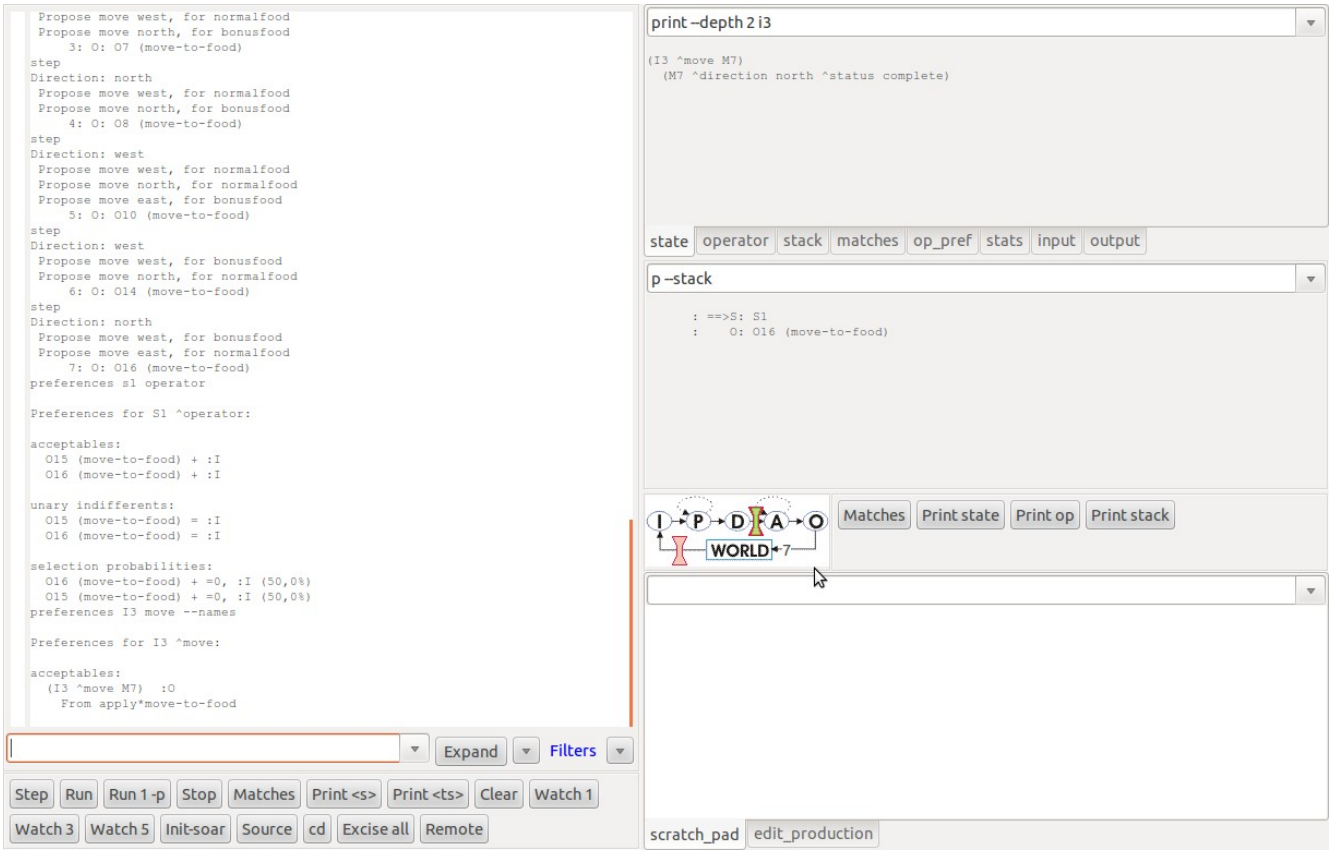

Figura 12 – Debugging commands- 2.

2.4 Semantic errors

Neste sub-tópico serão apresentadas as analises direcionadas aos erros semânticos. A Figura 13 apresenta o código que será analisado.

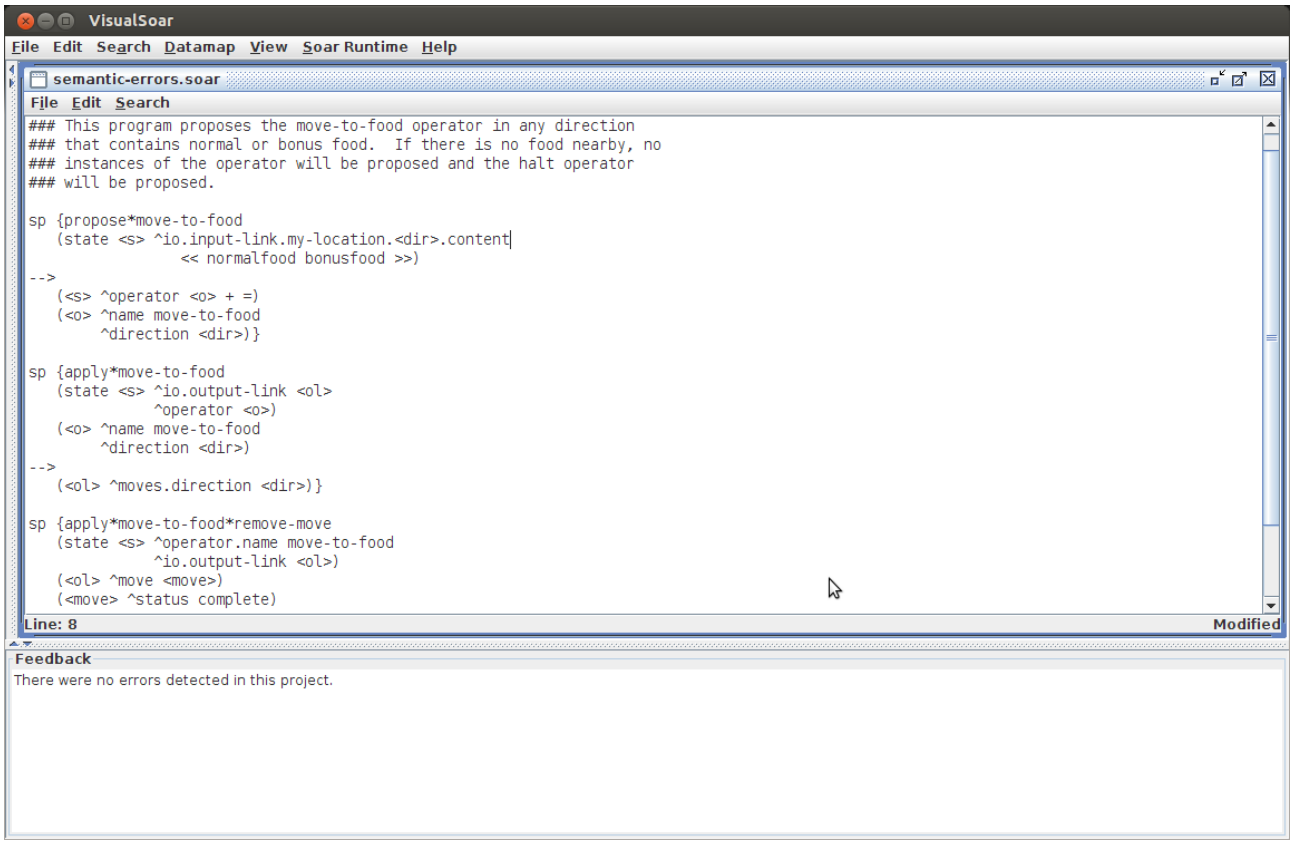

Figura 13 – Código erro semântico.

No caso específico foi necessário apenas a correção de dois comandos que foram trocados: o contant por content e o moves por move. Observamos o erro gerrado na Figura 14 que não seleciona nenhum operador para realizar alguma ação e o erro corrigido na Figura 15.

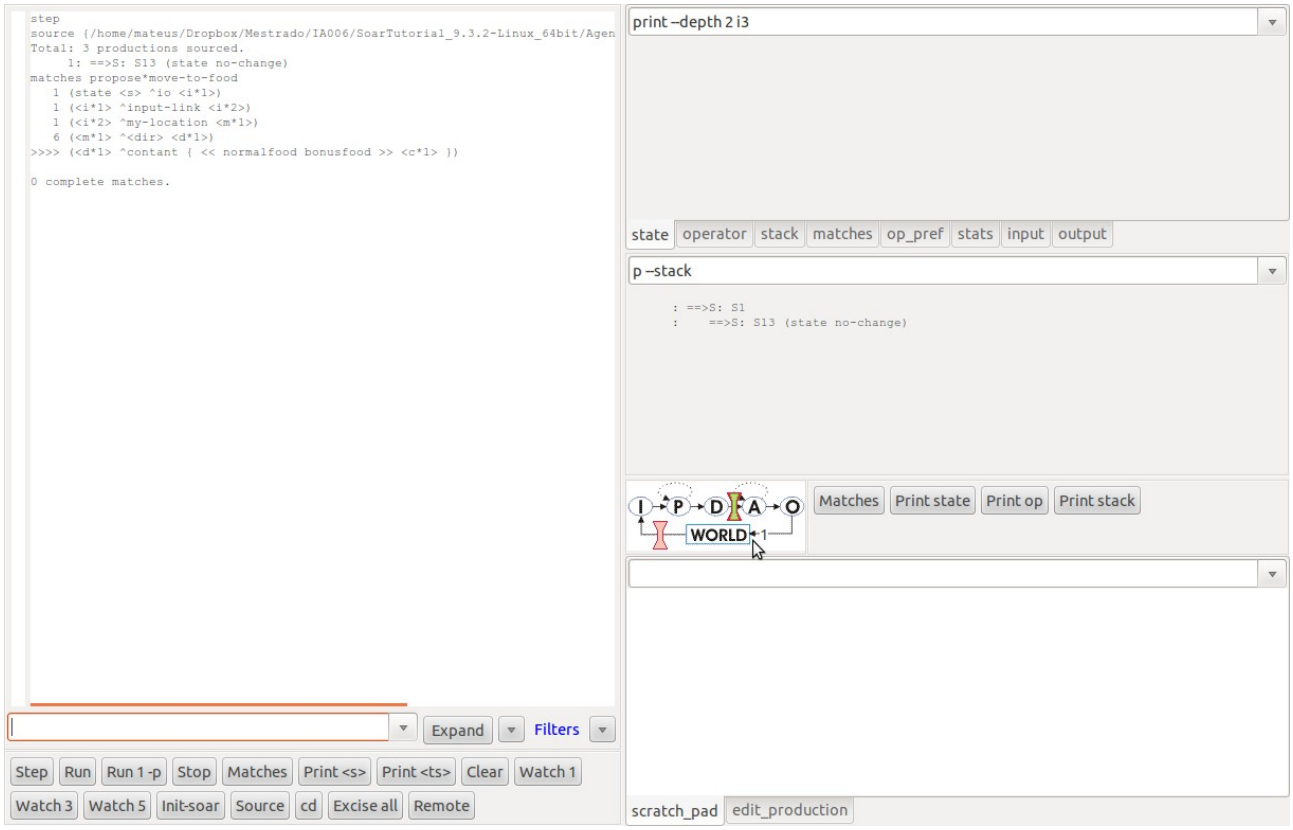

Figura 14 – Erro semântico - 1.

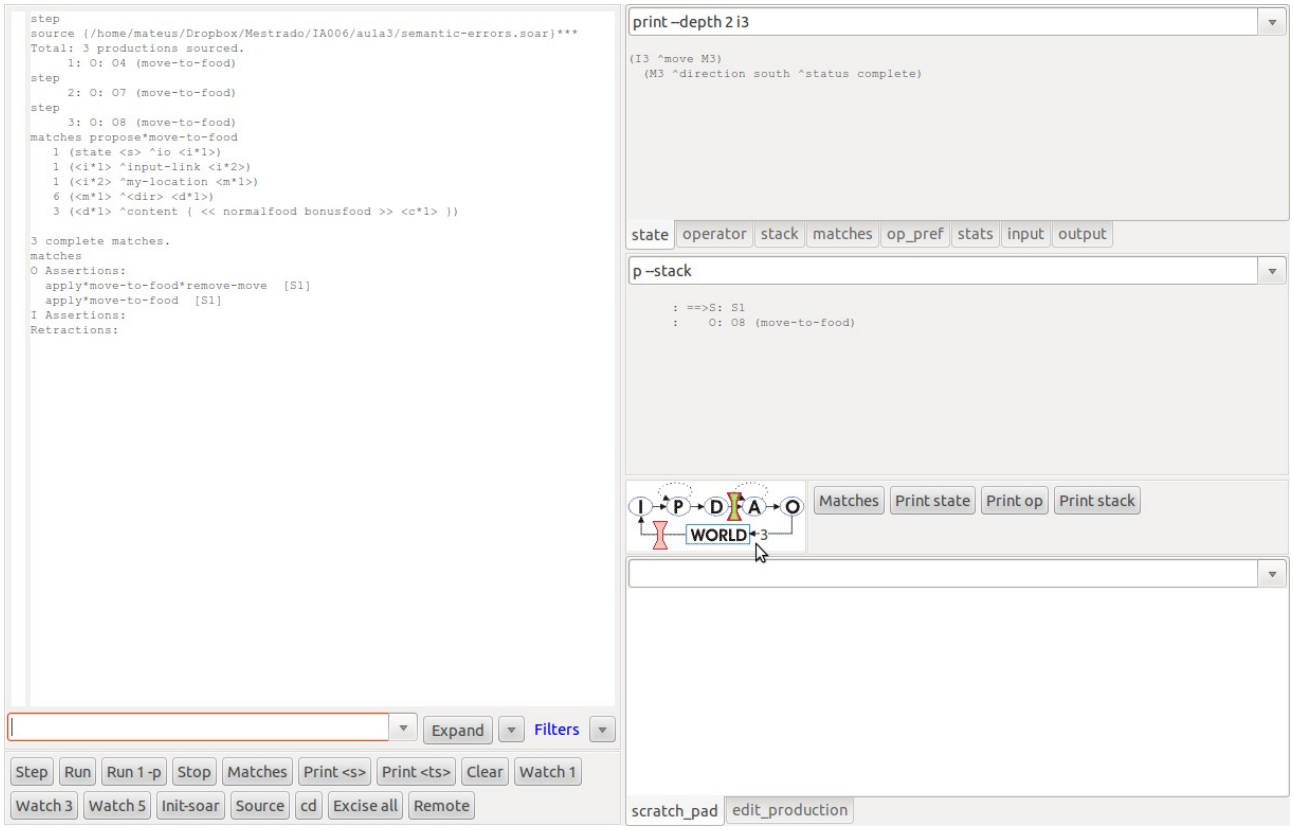

Figura 15 – Erro semântico - 2.

3 Atividade 3

Neste tópico são utlizadas técnicas para evitar que o agente busque comida com mais eficiência.
Os argumentos de preferências auxiliam nesta etapa. Estes operadores são: +, -, >, < e =; respectivamente aceitável, rejeitar, melhor/maior, pior/menor e indiferente (utilizado para função randômica).
As Figuras 16 e 17 apresentam a execução do agentes com a utilização dos argumentos de preferência, para a otimização de busca por comidas com maior recompensa, ou seja, comida bônus. Se não possuir comida bônus na vizinhaça o agente busca por comida normal e assim por diante.

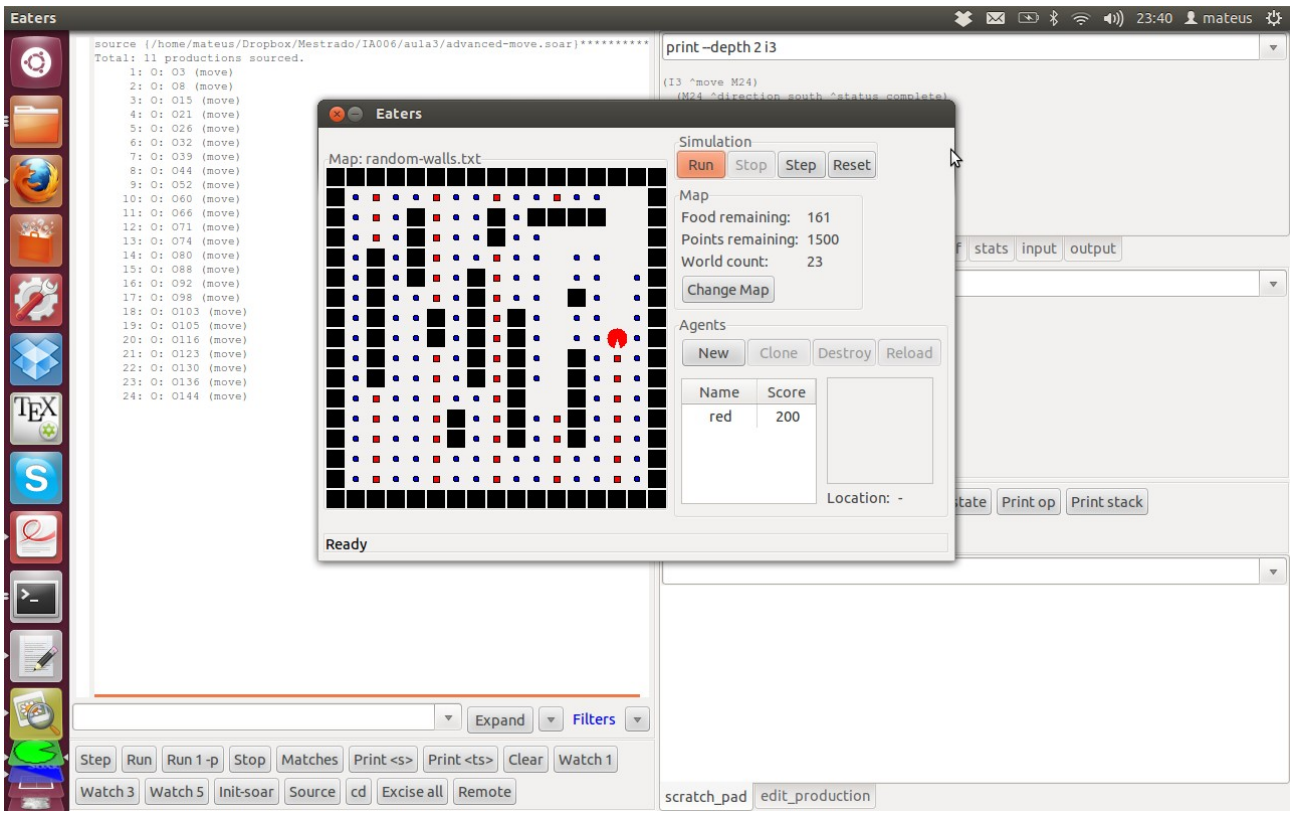

Figura 16 – Operador com movimento avançado - 1.

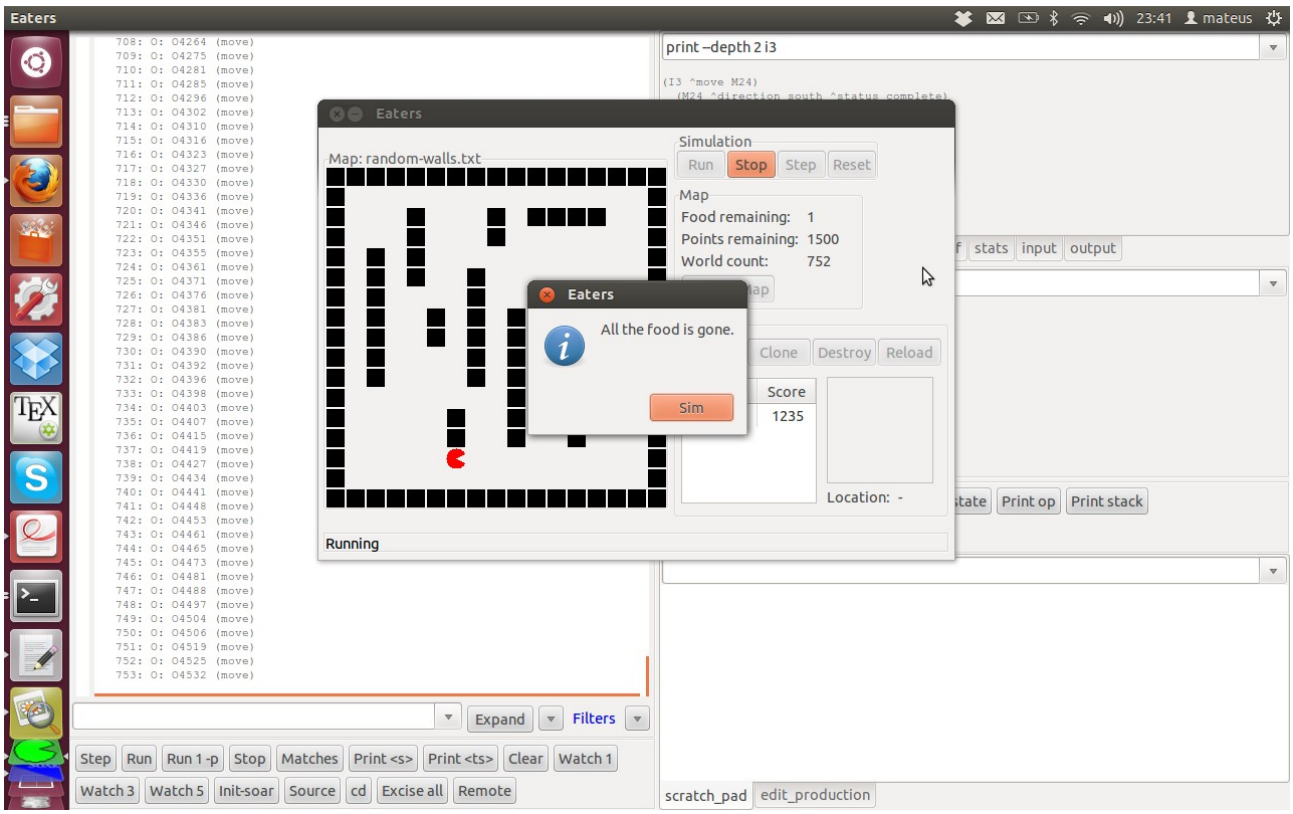

Figura 17 – Operador com movimento avançado - 2.

A Figura 18 apresenta o agente com operador de salto. Cada salto tem o mesmo custo que uma comida normal.

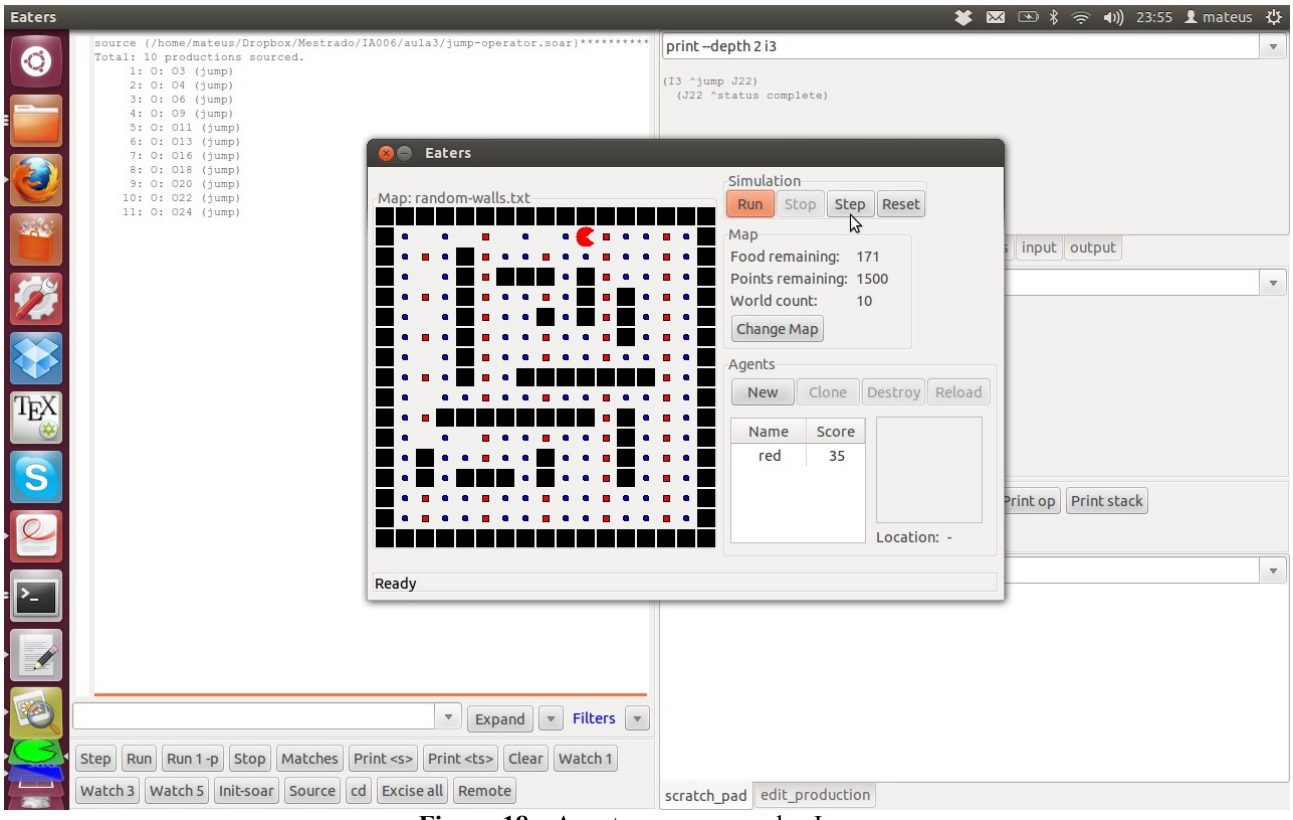

Figura 18 – Agente com o operador Jump.

4 Atividade 4

Para realizar esta tarefa, foi testado duas alternativas sem sucesso. A primeira foi, tentar forçar o agente a preferência por casas horizontais, caso as melhores opções sejam casas com o conteúdo vazio (empty):

```
sp {select*move*prefer-horizontal
(state <s> ^operator <o> +)
(<o> ^name move
^content empty
^direction <<west east>>)
-->
(<s> ^operator <o> >)}
```

Esta alternativa falhou, pois assim que o agente atinge paredes (wall) ele é obrigado a subir e se na subida ele também encontra um limitador, isso acaba gerando a sua volta, demorando mais tempo para o mesmo sair das regiões vazias. Consequentemente ficando com menos pontos competindo com os agentes avançados.

A segunda tentativa, frustada também, foi realizar o *merge* entre os agentes que possuem salto e movimento com os agente que guardam a última direção. Porém, assim que é gravado a última direção, as proposições de ações deixam de ser escolhidas, gerando estate-no-change. Para tentar contornar esse problema, foi tentado gravar o objeto inteiro no output-link, porém foi encontrada uma dificuldade de se apagar o último e manter apenas um operador gravado na memória. Essa tarefa, não estava presente em nenhum tutorial até o momento. A tarefa de apagar um estado criado em <s> foi possível, porém não funcionava para o *merge*.