Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Informática
Bacharelado em Ciência da Computação
6902-01 - Paradigma de Programação Lógica e Funcional

Paradigma de Programação Lógica e Funcional
Trabalho 2 - Racket

Alunos: Diogo Alves de Almeida RA: 95108

Professor: Wagner Igarashi

Maringá
2017

1 Introdução

Foi proposto para o segundo trabalho da disciplina de Paradigma de Programação Lógica e Funcional lecionada pelo professor Dr. Wagner Igarashi a implementação de um jogo em Racket no qual deveria haver duas soluções para a resolução do jogo. Esse relatório tem como propósito apresentar uma análise do algoritmo feito e comparar as duas soluções, apresentando as informações necessárias. O jogo escolhido para a implementação foi o Snake e foi implementado como veremos mais adiante uma função que faz a cobra andar sozinha procurando a comida. O código base foi baseado num código disponível no link `https://gist.github.com/antedeguemon/85fe254b5bf7464497fc`

2 Especificações

A seguir a imagem da especificação da máquina e do programa utilizado para a implementação do código.

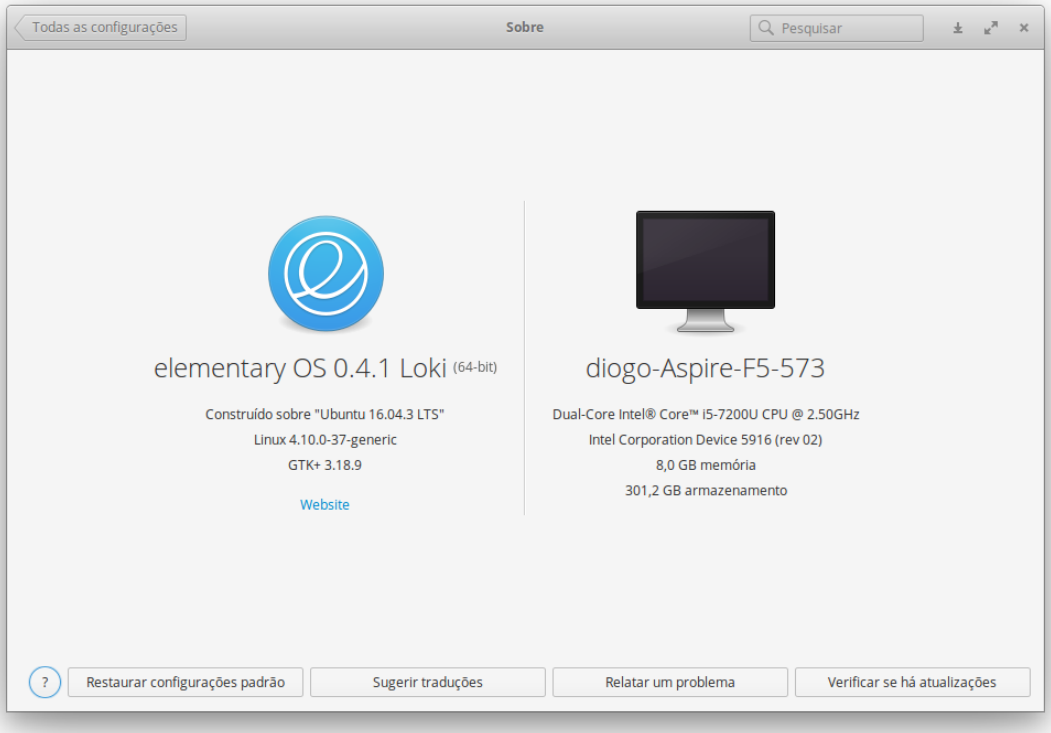

Figura 1: Especificação da máquina utilizada.

Figura 2: Especificação do programa utilizado.

3 Descrição

De modo geral, o algoritmo começa com o programa lendo o nome do jogador no terminal e setando as variáveis *pontos* e *recorde* sendo igual a zero, depois disso aparece a mensagem ao usuário indicando para qual das opções ele deseja selecionar: opção 1 onde ele irá para o modo jogador no qual ele mesmo controla a cobra e a opção 2 onde o jogo irá para o moto IA (Inteligência Artificial) onde a cobra anda sozinha procurando a comida através da função que veremos mais adiante.

O mapa do jogo consiste em um plano cartesiano com as coordenadas x e y, e a cobra anda pelo mapa a partir dessas coordenadas. A cobra consiste em uma lista onde cada posição dessa lista contém uma outra lista com 2 elementos que equivale a um ponto do plano com as coordenadas x e y. Cada vez que a cobra encontra a comida, é adicionado um ponto em sua lista fazendo com que ela cresça e também é aumentado um elemento na variável *ponto* indicando quantas comidas a cobra já comeu, também é verificado se a quantidade de pontos atual é maior que o recorde do jogo. Caso positivo, o ponto atual é o recorde do jogo e o nome do jogador é setado na variável *nome-recorde* para mostrar no fim do jogo o nome do jogador que contém o recorde do jogo. Se a cabeça da cobra encostrar nas extremidades desse plano ou encostar em algum ponto que esteja na cobra a função *fim-de-jogo* é chamada encerrando o jogo e mostrando para o usuário os pontos feitos por ele e o recorde do jogo.

Após o fim da partida o usuário poderá sair fechando a janela ou recomeçar o jogo digitando a letra **r**, onde irá para a página inicial novamente e deverá indicar o nome do jogador e modo que deseja.

A primeira solução proposta para a resolução do jogo é o modo jogador, onde o usuário é livre para controlar a cobra usando as teclas **w** (cima), **d** (direita), **s** (baixo) e **a** (esquerda), lembrando que se o usuário andar na direção oposta após ter comido alguma comida a cabeça irá bater no resto do corpo fazendo com que ele perca. Veremos a seguir a função que verifica as teclas apertadas pelo teclado e realiza as ações na interface setando a direção que a cobra deve ir.

```
1 (define (canvas-key frame) (class canvas%
2   (define/override (on-char key-event)
3     (cond ; todo: in_list? -> set direcao
4       [(eq? (send key-event get-key-code) '#\a) (set! direcao 'esquerda)]
```

```
5         [(eq? (send key-event get-key-code) '#\d) (set! direcao 'direita)]
6         [(eq? (send key-event get-key-code) '#\w) (set! direcao 'cima)]
7         [(eq? (send key-event get-key-code) '#\s) (set! direcao 'baixo)]
8         [(eq? (send key-event get-key-code) '#\1) (set! IA #f) (send comeco stop)
              (send jogo start 100)]
9         [(eq? (send key-event get-key-code) '#\2) (set! IA #t) (send comeco stop)
              (send jogo start 100)]
10        [(eq? (send key-event get-key-code) '#\r) (set! IA #f) (send jogo stop) (
              restart) (pega-nome) (send comeco start 100)]))
11    (super-new [parent frame])))
```

A segunda solução proposta para a resolução do jogo é um pouco mais sofisticada, ela é um pouco parecida com uma IA pois a cobra parece tomar decisões a partir do cálculo da distância de sua cabeça até a comida. Veremos o algoritmo a seguir:

```
1  (define (anda-sozinho)
2    (define x  (- (pega-cabeca 0 cobra) (list-ref comida 0)))
3    (define y  (- (pega-cabeca 1 cobra) (list-ref comida 1)))
4    (define dif-x (abs x))
5    (define dif-y (abs y))
6    (cond
7     [(> dif-x dif-y)
8      (if (> x 0) (cond [(eq? direcao 'direita) 'baixo]
9                        [else 'esquerda])
10                 (cond [(eq? direcao 'esquerda) 'cima]
11                       [else 'direita]))]
12    [else
13     (if (> y 0) (cond [(eq? direcao 'baixo) 'direita]
14                       [else 'cima])
15                 (cond [(eq? direcao 'cima) 'esquerda]
16                       [else'baixo]))])))
```

Nas linhas 2 e 3 definimos um x e y onde é calculado a distância entre a cabeça da cobra e a comida em relação as coordenadas x e y. Em seguida nas linhas 4 e 5 são calculados os valores absolutos para trabalharmos com valores positivos e verificarmos qual sentido a cobra deverá ir, na linha 7 vemos se a distância maior está no eixo x ou y, caso estiver no eixo x, a cobra deverá andar no sentido horizontal (para a direita ou esquerda), caso esteja no eixo y, deverá andar no sentido vertical (para cima ou para baixo). Feito isso, na linha 8 e 13 é verificado se o x ou y calculado anteriormente é positivo ou negativo para sabermos a direção da cobra, ou seja, se $x > 0$ significa que ela deverá ir para a esquerda, em seguida é apenas verificado se ela está na direção da direita, pois se ela for para a esquerda imediatamente ela baterá nela mesmo, então é definido que nesse caso ela irá para baixo e depois para a esquerda, evitando esse contato. De maneira análoga é feito isso para as outras direções.

O algoritmo não trata o caso em que a cobra pode bater nela mesmo no decorrer do caminho, sendo falho nas vezes que o corpo da cobra está entre o caminho da cabeça e a comida, portanto, para obter um sucesso duradouro durante o jogo é necessário contar um pouco com a sorte durante o surgimento da comida para que o corpo não fique no caminho da cabeça até a comida.

4 Conclusão

Não foi possível definir qual solução foi melhor tendo em vista que há inúmeras variáveis para colocar em questão como por exemplo a habilidade do usuário no modo jogador e da posição onde a comida é setada no modo IA. Na visão computacional o uso da memória para os dois algoritmos são mínimos pois não há nenhuma complexidade grande e o tempo de execução é constante uma vez que eles não iteram sobre nada durante a execução.