Documentação do Software

Trabalho Prático Final – Agentes Inteligentes
Presa e predador

Autores: Gabriel Henrique Silva Pinto
Rodrigo Emilio Pinto Silva

Belo Horizonte
06/2020.

Documentação do Software

Introdução

Escopo do software

Nome do sistema e de seus componentes principais

Presa e predador

Missão ou objetivo do software

O trabalho prático final da disciplina consistirá na implementação de um sistema de simulação de agentes inteligentes utilizando os conceitos de Programação Orientada a Objetos. A ideia é criar um mundo virtual e agentes capazes de interagir nesse mundo.

Descrição do domínio do Cliente (Regras de Negócio)

Número	*Regra de Negócio*	*Descrição*
1	*Inicializar*	*recebe uma referência do mundo e uma posição inicial x,y*
2	*Mover*	*nova posição x,y*
3	*Decidir*	*decide o que irá fazer (escolha de estado baseado nas suas percepções*
4	*Executar*	*executa um comportamento com base no estado definido anteriormente*
5		
6		
7		

Usuários e sistemas externos

Descrição

Número	*Usuários*	*Definição*
1	*Gabriel*	*Visual studio code*
2	*Rodrigo*	*Visual studio code*
3		

Documentação do código

Documentação da Estrutura de dados geral do software

Matriz
Matriz é a uma estrutura de dados do tipo vetor com duas ou mais dimensões.
Os itens de uma matriz tem que ser todos do mesmo tipo de dado.
Na prática, as matrizes formam tabelas na memória.
Exemplo de declaração de matriz com 2 dimensões usando linguagem C
float Media[5][2];
Onde:
O valor 5 representa a quantidade de linhas.
O valor 2 representa a quantidade de colunas.
Dizemos que esta matriz é do tipo 5 X 2.
Como temos 5 linhas com 2 posições de armazenamento em cada linha, temos capacidade para armazenar até 10 elementos (itens) do tipo float.
Será necessário utilizar um índice para cada dimensão da matriz, logo uma matriz bidimensional terá 2 índices, um para posicionar a linha, outro para a coluna.
Assim, como no vetor, o índice da primeira posição é zero.

Vetores
O vetor é uma estrutura de dados indexada, que pode armazenar uma determinada quantidade de valores do mesmo tipo.
Os dados armazenados em um vetor são chamados de itens do vetor.
Para localizar a posição de um item em um vetor usamos um número inteiro denominado índice do vetor.
Lembrando que o índice da primeira posição é zero
Sintaxe:
Tipo NomeDoVetor[quantidade_de_itens];

Classe
Uma classe é um tipo definido pelo usuário que contém o molde, a especificação para os objetos, assim como o tipo inteiro contém o molde para as variáveis declaradas como inteiros. A classe envolve, associa, funções e dados, controlando o acesso a estes, definí-la implica em especificar os seus atributos (dados) e suas funções membro (código).

Procedimento showState

```
// Procedimento para escrever na tela o estado
void showState(int s) {
    if(s == 1) {
        cout << "Passeando";
    }else if(s == 2) {
        cout << "Fugindo   ";
    }else if(s == 3) {
        cout << "Perseguindo";
    }else{
        cout << "Indefinido";
```

```
    }
}
```

Procedimento showDirection

```
// Procedimento para escrever na tela a direção
void showDirection(int d) {
    if(d == UP) {
        cout << "Cima  ";
    }else if(d == DOWN) {
        cout << "Baixo";
    }else if(d == RIGHT) {
        cout << "Direita";
    }else if(d == LEFT) {
        cout << "Esquerda";
    }else{
        cout << "Indefinido";
    }
}
```

Procedimento showDirection

```
    // Limpa a matriz
    void init() {
        // Loop para definir como "NULL" todas posições da matriz
        for(int i = 0; i < L; i++)
            for(int j = 0; j < C; j++)
                matrix[i][j] = NULL;
    }
```

Procedimento save

```
   void save() {
        // Exibe a matriz antes de salvar
        show();

        char save = 'n';
        // Pergunta se o usuário deseja salvar
        cout << "Deseja salvar o estado final (s para SIM / n para NÃO):
 ";
        cin >> save;

        // Condição que verifica se a resposta do usuário foi positiva
        if(save == 's'){
            FILE *fp;
            char fileName[30];
```

```
            // Solicita que o usuário digite o nome do arquivo e sua ext
ensão
            cout << "Digite o nome do arquivo e sua extensão (Ex: backup
.txt): ";
            cin >> fileName;

            // Abre o arquivo
            fp = fopen(fileName, "w");
            // Verifica se houve êxito ao abrir e, se não, exibe uma men
sagem e encerra
            if(!fp) { printf("Erro na abertura do arquivo"); exit(0); }

            // Insere na primeira linha do arquivo as dimensões da matri
z
            fprintf(fp, "%i %i\n", L, C);

            // Loop para inserir no arquivo a matriz
            for(int i = 0; i < L; i++){
                for(int j = 0; j < C; j++)
                    fprintf(fp, "%i ", AgentType(matrix[i][j]));

                fprintf(fp, "\n");
            }

            // Fecha o arquivo
            fclose(fp);
        }
    }
```

Procedimento move

```
    // Move o agente para uma nova posição
    void move(int newX, int newY) {
        // Condição que verifica se a nova posição não ultrapassa os lim
ites do tabuleiro
        if(newX >= 0 && newX <= C-1 && newY >= 0 && newY <= L-1) {
            // Condição que verifica se já não existe um agente na nova
posição
            if (!existAgent(newX, newY)) {
                aWorld->matrix[posY][posX] = NULL;
                posX = newX;
                posY = newY;
                aWorld->matrix[posY][posX] = this;
            }
        }
    }
```

Função validateDirection

```
    // Valida e, se necessário, altera a direção do movimento para não p
erder a jogada
    int validateDirection(int direction, int posX, int posY) {
        // Loop para mudar a direção quando o agente chegar no limite VE
RTICAL do tabuleiro
        while((direction == UP && posY == 0) || (direction == DOWN && po
sY == L-1)) {
            if(posX == 0){
                direction = RIGHT;
            }else if(posX == C-1){
                direction = LEFT;
            }else{
                direction = RANDOM_DIRECTION_X;
            }
        }

        // Loop para mudar a direção quando o agente chegar no limite HO
RIZONTAL do tabuleiro
        while((direction == LEFT && posX == 0) || (direction == RIGHT &&
 posX == C-1)) {
            if(posY == 0){
                direction = DOWN;
            }else if(posX == L-1){
                direction = UP;
            }else{
                direction = RANDOM_DIRECTION_Y;
            }
        }

        return direction;
    }
```

Procedimento exec

```
 void exec() {
        // Valida e, se necessário, altera a direção do movimento para n
ão perder a jogada
        direction = validateDirection(direction, posX, posY);

        // Executa o movimento
        if(direction == UP) {
            move(posX, posY - 1);
        }else if(direction == RIGHT) {
            move(posX+1, posY);
        }else if(direction == DOWN) {
            move(posX, posY+1);
```

```
        }else if(direction == LEFT) {
            move(posX-1, posY);
        }

        // Exibe as atuais informações
        cout << "\t\tAtual estado: "; showState(state);
        cout << " - Atual direção: "; showDirection(direction);
        cout << endl;
    }
```

Procedimento decide

```
    void decide() {
        // Exibe as últimas informações
        cout << "C: 2";
        cout << "\t\tÚltimo estado: "; showState(state);
        cout << " - Última direção: "; showDirection(direction);

        // Reseta o estado e direção
        state = RANDOM_WALK;
        direction = RANDOM_DIRECTION;

        // Verifica se existe presas em um raio de PERCEIVE_PREY quadros
e, se necessário, define um novo estado e direção
        verifyRadius();
    }
```