Departamento de Engenharia Informática e de Sistemas

Licenciatura em Engenharia Informática

Relatório Trabalho Prático

Programação Orientada a Objetos 2022/2023

Simulador de Reserva Natural

Trabalho Realizado por:

Quévin Moderno 2019135563

Vitor Simões 2019127806

1. Introdução

Este relatório descreve o desenvolvimento de um simulador de uma reserva natural povoada por diversos animais, utilizando a linguagem C++ e seguindo os princípios da Programação Orientada a Objetos. O projeto foi desenvolvido no contexto da disciplina de Programação Orientada a Objetos (2022/2023) do Instituto Superior de Engenharia de Coimbra.

2. Objetivos

O objetivo principal deste trabalho é construir um simulador que permita a interação com uma reserva natural, onde animais com diferentes comportamentos se deslocam e interagem autonomamente. O utilizador pode visualizar a reserva e dar comandos que afetam o que acontece no ambiente. O programa deve ser implementado em modo texto, garantindo a utilização das boas práticas de programação orientada a objetos.

3. Funcionalidades Implementadas

3.1. Simulação da Reserva

A reserva é representada por uma área retangular onde os animais se movem e interagem. O utilizador pode deslocar a vista da reserva e ver detalhes sobre os animais e os alimentos presentes. A reserva é dinâmica, com elementos que mudam de estado à medida que o tempo avança.

3.2. Comportamento dos Animais

Os animais simulados no ambiente possuem características como nascimento, morte, movimento e alimentação. Cada espécie de animal tem comportamentos específicos:
Coelho: Desloca-se aleatoriamente e consome alimentos que cheiram a verdura.
Ovelha: Desloca-se aleatoriamente, consome erva e reage a outros animais maiores.
Lobo: Predador que persegue e mata outros animais menores.
Canguru: Pacífico, com comportamento defensivo nas primeiras fases da vida.
Animal-mistério: Comportamento customizado.

3.3. Tipos de Alimentos

Existem diferentes tipos de alimentos disponíveis na reserva, que podem ser consumidos pelos animais. Cada alimento possui características como valor nutritivo e toxicidade, e pode evoluir ao longo do tempo.

3.4. Interação com o Utilizador

O utilizador interage com o simulador por meio de comandos de texto, como criar animais, alimentar diretamente, e avançar o tempo da simulação. Todos os comandos são validados antes de serem executados.

3.5. Interface de Texto

A interface do programa é feita em modo texto, com a reserva sendo representada por caracteres que simbolizam animais e alimentos. O utilizador pode dar comandos para deslizar a área visível e visualizar informações detalhadas de uma posição específica.

4. Estruturas de classes

a) Interface: Tem como função mostrar os dados do simulador. Representa a interface do programa e é a única que comunica com o utilizador.

b) Simulador: Gere a interação com o utilizador, interpretando os comandos e atualizando o estado da reserva.

```
class Simulator {

    Reserve reserve;

    int comand(string txt);
    bool load(string file);

    void NextInstant();
    void NextInstant(int n);
    void NextInstant(int n, int p);

    bool endReserve = false;
    string showGUI;
public:
    Simulator(int rows, int columns);

    ~Simulator();
    string getshow(){string a = showGUI; showGUI = ""; return a;}
```

A classe *Simulator* trata de toda a logica do programa.
Encapsula todos os dados relativos ao programa, mais especificamente, os dados da reserva. É a única que trata dos pedidos da interface, sendo a responsável por interpretar os seus pedidos e utilizar as outras classes para os realizar. Ao terminar o seu construtor e ativado, ativando também o destrutor da reserva.

c) Reserva: Responsável pela gestão do espaço da reserva, controla os elementos (animais e alimentos) e o avanço do tempo na simulação.

```
class Reserve {
    map< int, vector<int> > positions;
    vector<Animal*> animals;
    vector<Food*> food;


    int visibleRows[2] = { [0]: 1,  [1]: VISIBLEROWS};
    int visibleColumns[2] = { [0]: 1,  [1]: VISIBLELINES};
    int ROWS;
    int COLUMNS;

    void AddinMap(int rows, int columns);
    bool PositionAvailable(string specie, const vector<int> &pos);
    string findinPosition(const vector<int> &pos);
    vector<int> getIDbyPosition(const vector<int> &pos);
```

Esta classe inclui e gere os objetos das classes Food e Animal, armazenando as suas informações de coordenadas dentro da reserva. Para guardar as suas posições, decidimos criar um map, tem com key o seu id e como atributo a posição onde este se encontra na reserva. Esta abordagem permite gerir eficientemente a localização dos elementos no ambiente simulado.
A classe Reserve estabelece uma relação de **agregação** com as classes Animal e Food, uma vez que estes elementos só podem existir dentro do contexto da reserva. Os objetos Animal e Food são criados dinamicamente no decorrer da simulação, dentro da classe Reserve.
Por conseguinte, o destrutor da classe Reserve é responsável por garantir a correta remoção dos objetos Animal e Food, eliminando as referências a esses objetos criados em runtime, assegurando uma adequada gestão de memória e evitando vazamentos de memória (memory leaks).

d) Food: Representa os alimentos na reserva, com propriedades como valor nutritivo e toxicidade, e diferentes tipos de alimentos (relva, cenoura, corpo, bife).

```
class Food {
    int ID;
    string letter;

    int nutricity;
    int toxicity;
    vector<string> smell;

public:
    Food(string specie,int id);
    ~Food();

    int getID() const {return this->ID;}
    string getLetter(){return this->letter;}

    friend ostream& operator<< (ostream& output, Food* a);
```

Armazena e gere todas as informações relacionadas aos alimentos presentes na reserva. Ela é responsável por manter uma **coleção polimórfica** de diferentes tipos de alimentos, permitindo que cada um desempenhe as suas funções específicas. Através da utilização de **funções virtuais**, a classe possibilita que os alimentos implementem comportamentos distintos de forma polimórfica, de acordo com as suas características e tipos.

e) Animal: Classe base que contém as propriedades e comportamentos comuns a todos os animais, como saúde, fome, movimento, e interação com o ambiente.

```
class Animal {

    int ID;
    string specie;
    string letter;

    string history;
    int hunger;
    double weight;
    int health;
    bool alive;


    string setHistory(string specie, int nutricity, int toxicity);
public:
    Animal(string specie, int id);
    ~Animal();
```

Guarda informações relativas aos animais. É responsável pela sua coleção polimórfica de Animais.

Coelho, Ovelha, Lobo, Canguru: Classes derivadas da classe Animal, cada uma implementando as particularidades das suas respetivas espécies.

f) Save: Tem como função guardar momentos do jogo.

```
class SaveState {
    string name;
    Simulator simulator;
    int day;

public:
    SaveState(Simulator& sim): simulator(sim){
        name = simulator.saveName;
        day = simulator.getday();
    }

    string getName(){return this->name;}
    int getDay(){return this->day;}
    Simulator getSIM(){return this->simulator;}
};
```

5. Conclusão

Este trabalho permitiu a aplicação de conceitos de Programação Orientada a Objetos na construção de um simulador complexo, gerindo múltiplas entidades e interações dinâmicas. Foram desenvolvidas as funcionalidades principais para simular uma reserva natural com animais e alimentos, respeitando as restrições e requisitos especificados no enunciado.