Departamento de Engenharia Informática e de Sistemas

Licenciatura em Engenharia Informática

Relatório Trabalho Prático

Programação Orientada a Objetos 2022/2023

META FINAL

Trabalho Realizado por:

Quévin Moderno 2019135563

Vitor Simões 2019127806

Classes utilizadas

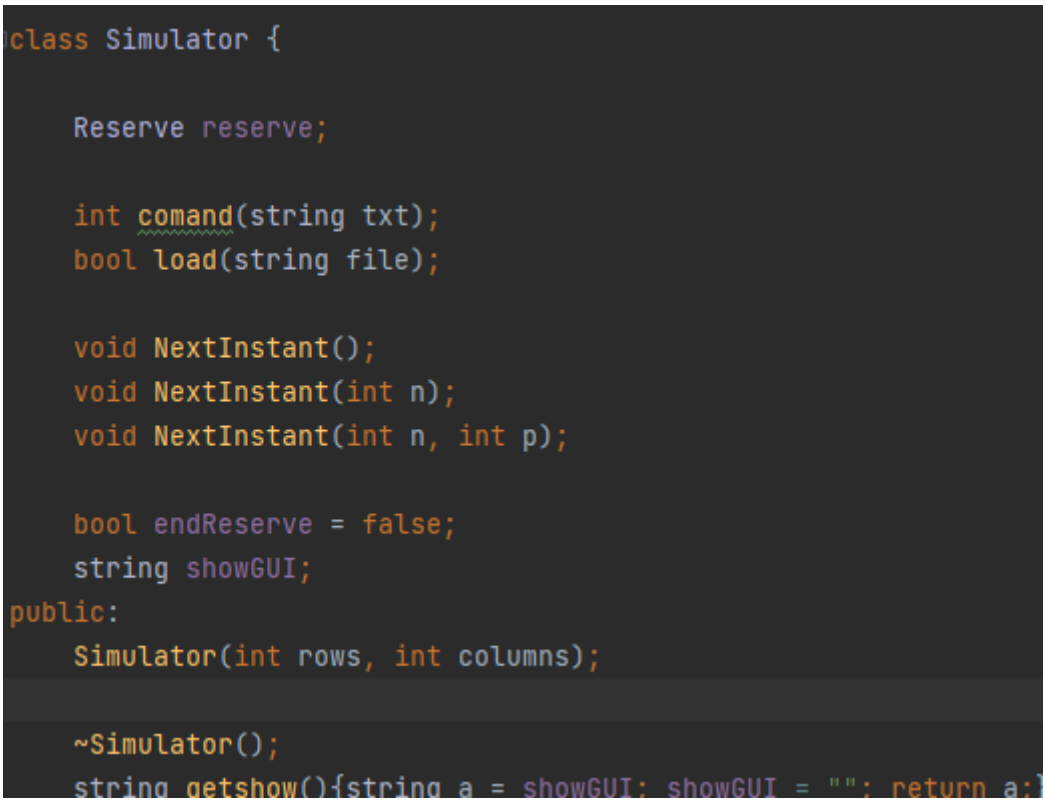
1. Classe Interface

```
class Simulator {
    Reserve* reserve;
    bool endReserve = false;
    bool comand(string txt);
    bool load(string file);
    void NextInstant();
    void NextInstant(int n);

public:
    Simulator(int rows, int columns);
    ~Simulator();
```

Esta classe tem como função mostrar os dados do simulador. Representa a interface do programa e é a única que comunica com o utilizador.

2. Classe Simulador



```
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.

3. Classe Reserva

```
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 a classe Food e a classe Animal. É responsável pela sua gestão e guarda a informação relativa às suas coordenadas nesta reserva.
Para guardar as suas posições onde estão inseridos decidimos criar um map, tem com key o seu id e como atributo a posição onde este se encontra na reserva.
A classe Reserve tem uma relação de agregação com as classes Animal e Food porque os animais e comida não podem existir sem que a reserva exista e por isso que estes objetos são criados dentro desta classe.
Por isto, ao ser ativado o seu destrutor, é responsável por eliminar o caminho para os objetos Food e Animal criados em runtime.

4. Class Food

```
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);
```

Guarda informações relativas às comidas. É responsável pela sua coleção polimórfica de Comidas. Possui funções virtuais para as suas comidas desempenharem as suas diferentes tarefas, através de um comportamento polimórfico.

5. Classe Animal

```
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

6. Classe Save

```
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;}
};
```

Classe Save. Tem como função guardar momentos do jogo.

Problemas no trabalho:

Save: A classe em si está correta, mas não se encontra funcional. Criamos construtores por copia na classe Reserva para duplicar os seus objetos e criamos uma função clone nos animais para duplicarem os seus objetos. Por alguma razão, a implementação não esta correta.

Kangaroo: O filho não está a ser devidamente criado, não fica guardado na bolsa da mãe.