Instituto Superior de Engenharia

Politécnico de Coimbra

Programação Orientada a Objetos

Licenciatura em Engenharia Informática - 2022/2023

Trabalho prático – Meta 2

Beatriz Isabel Inácio Maia - a2020128841

Ruben Soares Agostinho - a2020157100

Índice

1. Introdução

Este trabalho tem como objetivo construir um simulador em C++ de uma reserva natural habitada por animais com comportamento autônomo e variado. O simulador permite que o utilizador interaja com a reserva, consegue visualizar o seu conteúdo e com ordens altera o que acontece na simulação. O simulador é apresentado na forma de alguns caracteres no ecrã, alguns dos quais (os animais) vão movimentar-se quando o utilizador avançar para o próximo instante da simulação.

2. Classes

Para este trabalho foram criados 6 ficheiros (sendo um o main) com os respetivos.h.

2.1 Reserva.h

```
class Reserva {
public:
    static int getNextId();
    static int getID();
    static int aumentaInstante();
    static int getInstante();
    static int idAnterior();
    Reserva(int novasLinhas, int novasColunas);

    void adicionaAnimal(Animal *animal, int linha, int coluna);

    bool colocaAlimento(Alimento *alimento, int linha, int coluna);

    void desenhaReserva(int posicaoLinha, int posicaoColuna, int limiteMaximo);

    int getNLinhas() const;

    int getNColunas() const;

    ~Reserva();

    //Funções que retornam os vetores
    vector<vector<Zona *>> getZonas();
```

```
private:
    vector<vector<Zona *>> reservaGeral;
    static int id;
    int nLinhas;
    int nColunas;

    static int instante;
```

static int getNextId() – Função que tem como objetivo retomar o próximo ID do animal ou alimento.

static int getID() - obtém ID atual do animal ou alimento.

static int aumentaInstante() – aumenta o instante atual .

static int getInstante() – retorna o instante atual.

static int idAnterior() – retorna o id anterior.

Rerserva(int novasLinhas, int novasColunas) - construtor da classe Reserva que recebe como parâmetros o número de linhas e número de colunas da reserva.

void adicionarAnimal(Animal *animal, int linha, int coluna) - adicionar animal na reserva numa determinada linha e coluna.

bool colocaAlimento(Alimento *alimento, int linha, int coluna) - adicionar alimento na reserva numa determinada linha e coluna

void desenhaReserva(int posicaoLinha, int posicaoColuna, int limiteMaximo) - desenha a reserva no terminal, com uma posição inicial de linha e coluna e um máximo de linhas e colunas a serem mostradas.

int getNLinhas() const - retorna o número de linhas da Reserva

int getNColunas() const – retorna o número de colunas da Reserva

~Reserva() - destrutor da classe Reserva

getZonas() – retorna o vetor de vetores de zonas da reserva

2.2. Zona.h

```
class Zona {

public:
    Zona();
    void adicionaAnimal(Animal *animal);
    bool colocaAlimento(Alimento *alimento);
    bool removeAnimal(int id,bool deixaCorpo);
    bool removeAlimento();


    Alimento* getAlimento();
    vector<Animal*> getAnimais();

    string toString();
private:
    Alimento* alimento;

    vector<Animal*> animais;
};
```

Zona() -- construtor de zona.

void adicionaAnimal(Animal *animal) - adiciona o animal ao vetor dos animais.

bool colocaAlimento(Alimento *alimento) - adiciona o alimento ao vetor dos alimentos.

bool removeAnimal(int id, bool deixaCorpo) – remove um animal do vetor de animais

bool removeAlimento() - remove o alimento da zona.

Alimento* getAlimento() - obter o alimento que está numa zona.

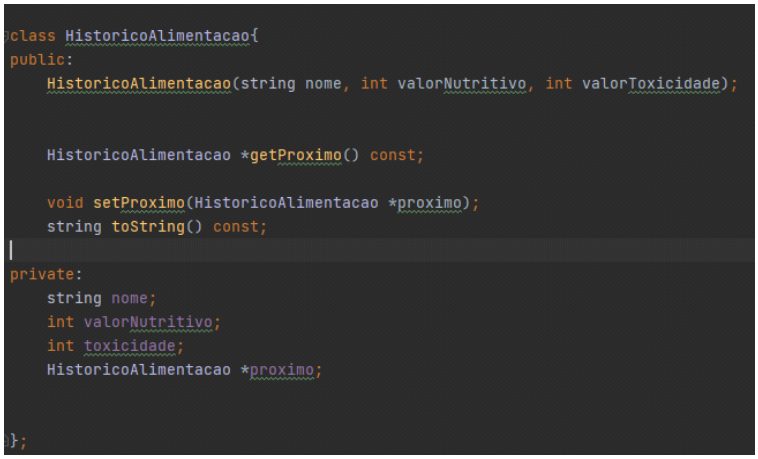
vector <Animal*> getAnimais() - obter os animais numa zona.

string toString() - imprime o tipo de alimento e o tipo dos animais.

Alimento* alimento- um ponteiro para o alimento presente na zona.

vector<Animal*> animais - um vetor de ponteiros para os animais presentes na zona

2.3. Animal.h



```
class HistoricoAlimentacao{
public:
    HistoricoAlimentacao(string nome, int valorNutritivo, int valorToxicidade);


    HistoricoAlimentacao *getProximo() const;

    void setProximo(HistoricoAlimentacao *proximo);
    string toString() const;

private:
    string nome;
    int valorNutritivo;
    int toxicidade;
    HistoricoAlimentacao *proximo;


};
```

Class HistoricoAlimentacao: mantem guardada informação sobre o que foi comido, o seu valor nutritivo e o seu valor de toxicidade.

HistoricoAlimentacao(string nome, int valorNutritivo, int valorToxicidade) – construtor da classe.

HistoricoAlimentacao *getProximo() const – devolve um ponteiro para o próximo na lista.

void setProximo(HistoricoAlimentacao *próximo) – aponta o ponteiro para o próximo na lista.

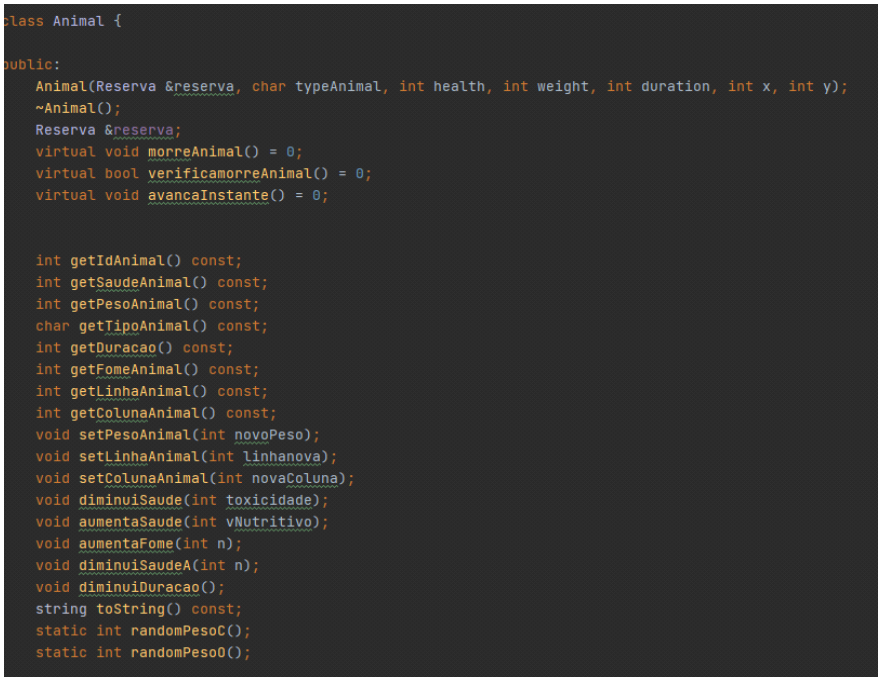
String toString() const – devolve uma representação em formato string do objeto.

string nome – nome do alimento.

int valorNutritivo – valor nutritivo do alimento.

int toxicidade – valor de toxicidade do alimento.

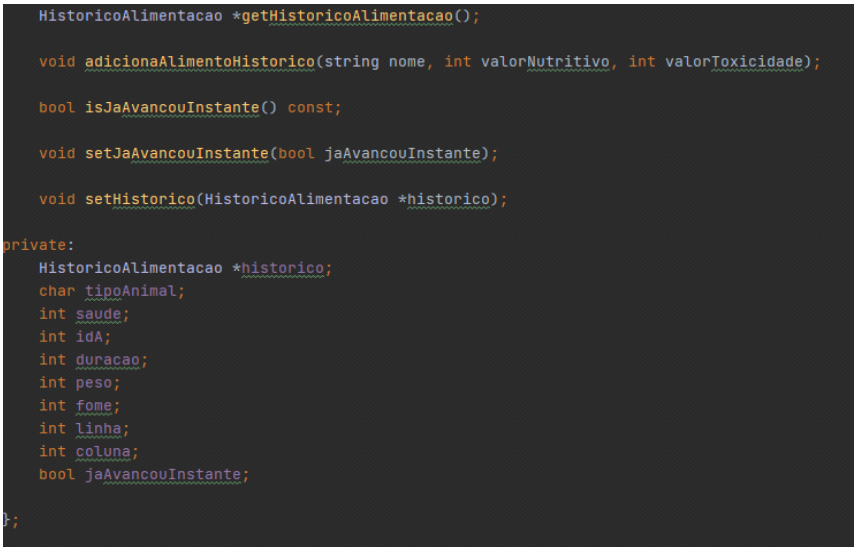
HistoricoAlimentacao *próximo – ponteiro para o próximo objeto HistoricoAlimentacao.



```
class Animal {

public:
    Animal(Reserva &reserva, char typeAnimal, int health, int weight, int duration, int x, int y);
    ~Animal();
    Reserva &reserva;
    virtual void morreAnimal() = 0;
    virtual bool verificamorreAnimal() = 0;
    virtual void avancaInstante() = 0;


    int getIdAnimal() const;
    int getSaudeAnimal() const;
    int getPesoAnimal() const;
    char getTipoAnimal() const;
    int getDuracao() const;
    int getFomeAnimal() const;
    int getLinhaAnimal() const;
    int getColunaAnimal() const;
    void setPesoAnimal(int novoPeso);
    void setLinhaAnimal(int linhanova);
    void setColunaAnimal(int novaColuna);
    void diminuiSaude(int toxicidade);
    void aumentaSaude(int vNutritivo);
    void aumentaFome(int n);
    void diminuiSaudeA(int n);
    void diminuiDuracao();
    string toString() const;
    static int randomPesoC();
    static int randomPesoO();
```



```
    HistoricoAlimentacao *getHistoricoAlimentacao();

    void adicionaAlimentoHistorico(string nome, int valorNutritivo, int valorToxicidade);

    bool isJaAvancouInstante() const;

    void setJaAvancouInstante(bool jaAvancouInstante);

    void setHistorico(HistoricoAlimentacao *historico);

private:
    HistoricoAlimentacao *historico;
    char tipoAnimal;
    int saude;
    int idA;
    int duracao;
    int peso;
    int fome;
    int linha;
    int coluna;
    bool jaAvancouInstante;

};
```

Animal(Reserva &reserva, char typeAnimal, int health, int weight, int duration, int x, int y) – constructor da classe.

~Animal() – destrutor da classe.

Reserva &reserva – referencia da Reserva que está a ser usada.

virtual void morreAnimal() = 0 – função virtual pura.

virtual bool verificamorreAnimal() = 0 – função virtual pura.

virtual void avancaInstante() = 0 – função virtual pura.

Funções get – retornam o ID, saúde, peso, tipo, duração, fome, linha e coluna do animal.

Funções set- mudam o peso, linha e coluna do animal para um valor especificado.

void diminuiSaude(int toxicidade) – função que reduz a saúde do animal tendo em conta a toxicidade especificada.

void aumentaSaude(int vNutritivo) – função que aumenta a saúde do animal tendo em conta o valor nutritivo especificado.

void aumentaFome(int n) – função que aumenta a fome.

void diminuiSaudeA(int n) – função que reduz a saúde.

Void diminuiDuracao() – função que diminui a duração do animal.

String toString() const – função que coloca a informação em formato string.

Static int randomPesoC() – função que decide o valor inicial de peso do Coelho tendo em conta um intervalo especifico.

Static int randomPesoO() – função que decide o valor inicial de peso da Ovelha tendo em conta um intervalo especifico.

HistoricoAlimentacao *getHistoricoAlimentacao() - ponteiro para HistoricoAlimentacao.

void adicionaAlimentoHistorico(string nome, int valorNutritivo, int valorToxicidade) - função que adiciona alimento ao histórico de alimentação.

bool isJaAvancouInstante() const - função que confirma se o animal já se movimentou no instante em questão.

void setJaAvancouInstante(bool jaAvancouInstante) - função que muda o estado do animal caso já tenha movimentado no instante em questão.

void setHistorico(HistoricoAlimentacao *historico) - função que muda o histórico de alimentação do animal.

No private temos:

HistoricoAlimentacao *historico que é o ponteiro para o histórico de alimentação e as variáveis úteis para os animais. Além disso contém bool jaAvancouInstante - variavel para registar se o animal já se movimentou ou não.

```
class Coelho: public Animal{
public:
    Coelho(Reserva &reserva, int x,int y);
    ~Coelho();

    void avancaInstante() override;
    void morreAnimal() override;
    bool verificamorreAnimal() override;
};
```

Class Coelho - classe que serve para a criação do objeto Coelho.

Coelho(Reserva &reserva, int x, int y) - construtor da classe coelho.

~Coelho() - destrutor da classe coelho.

void avancaInstante() override - função que representa o comportamento do coelho quando se avança um instante.

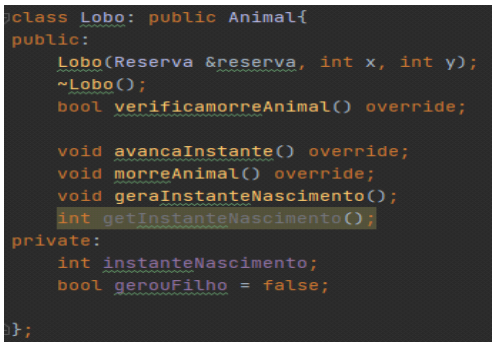
void morreAnimal() override - função que representa o comportamento do coelho quando morre.

bool verificamorreAnimal() override - função que verifica se realmente o coelho vai morrer.

```
class Ovelha: public Animal{
public:
    Ovelha(Reserva &reserva, int x, int y);
    ~Ovelha();

    void avancaInstante() override;
    void morreAnimal() override;
    bool verificamorreAnimal() override;
};
```

A classe ovelha tem exatamente as mesmas funções que a classe do Coelho mas para a ovelha.



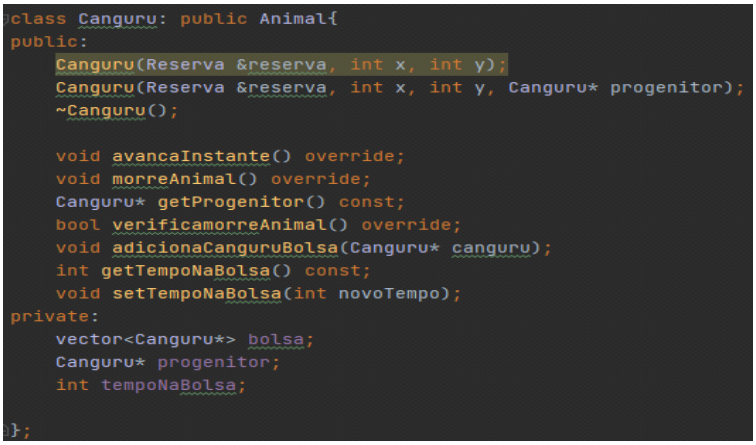
A classe Lobo tem exatamente as mesmas funções que a classe do Coelho mas para o Lobo. Além destas tem também:

void geraInstanteNascimento() - função que decide o instante em que o lobo pode ter um filho.

Int getInstanteNascimento() - função que devolve o instante em que o lobo pode ter um filho.

Int instanteNascimento - variavel para o instante em que o lobo vai ter um filho.

bool gerouFilho = false - variavel que informa se o lobo já teve um filho ou não.



Na classe Canguru para além das funções utilizadas também na classe do Coelho esta tem:

Canguru(Reserva &reserva, int x, int y, Canguru* progenitor) - construtor da classe canguru.

Canguru* getProgenitor() const – função para obter o canguru progenitor.

void adicionaCanguruBolsa(Canguru* canguru) - função que adiciona um canguru filho à "bolsa" do progenitor.

int getTempoNaBolsa() const - função que retorna o tempo que o filho esteve na "bolsa" do progenitor.

void setTempoNaBolsa(int novoTempo) - função que muda o tempo do filho na bolsa para um tempo especificado.

Vector<Canguru*> bolsa - vetor que guarda os filhos de um canguru quando estes se encontram na bolsa.

Canguru* progenitor - ponteiro para o progenitor do canguru.

int tempoNaBolsa - varivel para o tempo que o canguru se encontra na "bolsa" do progenitor.

```
class Misterio: public Animal{
public:
    Misterio(Reserva &reserva, int x, int y);
    ~Misterio();

    void avancaInstante() override;
    void morreAnimal() override;
    bool verificamorreAnimal() override;
};
```

A class Misterio tem exatamente as mesmas funções que a classe do Coelho mas para o animal Misterio.

2.4 Alimento.h

```
class Reserva;
class Alimento {

public:
    Alimento(char typeAlimento, int vNutritivo, int toxi, int duration, initializer_list<string> cheiro, int x, int y);
    virtual ~Alimento();

    int getIdAlimento() const;
    virtual char getTipoAlimento()const =0;
    virtual string getNomeAlimento()const =0;
   // virtual bool desapareceAlimento() const= 0;


    int getToxicidade() const;
    int getvalorNutritivo() const;
    int getDuracao() const;
    vector<string> getCheiro();
    int getLinhaAlimento();
    int getColunaAlimento();
```

Alimento(char typeAlimento, int vNutritivo, int toxi, int duration, intializer_list<string> cheiro, int x, int y) - construtor da classe.

virtual ~Alimento() - destrutor da classe.

Virtual char getTipoAlimento()const = 0 - função virtual pura.

Virtual string getNomeAlimento()const = 0 - função virtual pura..

As funções get retornam os parâmetros que definem o alimento como o id, toxicidade, valor nutritivo, etc.

```
    virtual void avancaInstante() = 0;

    void aumentaToxicidade();
    void diminuiValorNutritivo();
    void diminuiValorDuracao();
    void aumentaValorNutritivo();


    virtual string toString() const;

private:
    char tipoAlimento;
    int valorNutritivo;
    int idAlimento;
    int toxicidade;
    int duracao;
    int linha;
    int coluna;
    vector<string> cheiro;
};
```

No private temos as variáveis que definem o alimento como o id, toxicidade, duração, etc. Além disso temos as funções:

Virtual void avancaInstante() = 0 - função virtual pura.

Void aumentaToxicidade() - função que aumenta a toxicidade do alimento.

Void diminuiValorNutritivo() - função que diminui o valor nutritivo do alimento.

Void diminuiValorDuracao() - função que diminui a duração do alimento.

Void aumentaValorNutritivo() - função que aumenta o valor nutritivo do alimento.

Virtual string toString() const - imprime o tipo de alimento.

```
class Relva : public Alimento{
public:
    Reserva &reserva;

    ~Relva() override;

      char getTipoAlimento() const override;
      string getNomeAlimento() const override;
      void apareceRelva();//5
      void avancaInstante() override;
      bool desapareceAlimento();

    Relva(Reserva &reserva,int x, int y);
private:
bool gerou=false;

};
```

Na classe Relva temos:

Reserva &reserva - referencia da reserva.

~Relva() override - destrutor da class relva.

char getTipoAlimento() const override - função que devolve o tipo do alimento.

String getNomeAlimento() const override - função que devolve o nome do alimento.

void apareceRelva() - função que adiciona relva à reserva.

void avancaInstante() override - função que representa o comportamento da relva quando passa um instante.

bool desapareceAlimento() - função que remove o alimento da reserva quando a duração acaba.

Relva(Reserva &reserva, int x, int y) - construtor da class relva.

bool gerou = false - varivel que informa se a relva já gerou outra ou não.

```
class Cenoura: public Alimento{
public:
    Cenoura(int x,int y);
    ~Cenoura() override;

    string getNomeAlimento() const override;
    char getTipoAlimento() const override;
    void avancaInstante() override;

};
```

A classe Cenoura contêm as funções da classe Relva mas para a Cenoura só tendo a diferenciação de no seu construtor não necessitar da Reserva.

```
class Corpo: public Alimento{
public:
    Corpo(int x, int y, int vNutritivo, int toxi);
    Corpo(int x, int y);
    ~Corpo() override;

    string getNomeAlimento() const override;
    char getTipoAlimento() const override;
    void avancaInstante() override;
    //bool desapareceAlimento() override;

private:
    int instanteInicial;
    bool geraValorNutritivo=true;
    int valorNutritivoInicial;
};
```

Class Corpo - class que serve para criação do objeto Corpo.

~Corpo() override - destrutor da class corpo.

Char getTipoAlimento() const override - função que devolve o tipo do alimento.

String getNomeAlimento() const override - função que devolve o nome do alimento.

Void avancaInstante() override - função que representa o comportamento do corpo quando passa um instante.

Int instanteInicial - varivel para o instante em que o corpo é criado.

bool geraValorNutritivo = true - variavel que determina se o corpo tem valor nutritivo.

int valorNutritivoInicial - variavel para o valor nutritivo inicial do alimento corpo.

```
class Bife: public Alimento{
public:
    Reserva &reserva;
    Bife(Reserva &reserva,int x, int y);
    ~Bife() override;

    string getNomeAlimento() const override;
    char getTipoAlimento() const override;
    void avancaInstante() override;
    bool desapareceAlimento();
};
```

A classe Bife contêm as funções da classe Relva mas para o Bife.

```
class AlimentoMisterio: public Alimento{
public:
   Reserva &reserva;
   AlimentoMisterio(Reserva &reserva,int x,int y);
    ~AlimentoMisterio() override;

    string getNomeAlimento() const override;
    char getTipoAlimento() const override;
    void avancaInstante()override;
    bool desapareceAlimento();

};
```

A classe AlimentoMisterio contêm as funções da classe Relva mas para o AlimentoMisterio.

2.5 Comandos.h

```
class InicioJogo {
        public:
        static int getCoelhoSaude();
        static int getOvelhaSaude();
        static int getLoboSaude();
        static int getCanguruSaude();
        static int getMisterioSaude();
        static int getCoelhoVida();
        static int getOvelhaVida();
        static int getLoboVida();
        static int getCanguruVida();
        static int getMisterioVida();
        static int getCoelhoPeso();
        static int getOvelhaPeso();
        static int getLoboPeso();
        static int getCanguruPeso();
        static int getMisterioPeso();
```

A classe InicioJogo serve para inicialização das variáveis dos objetos do jogo.

Static int getAnimalSaude() - funções que retornam a saúde do animal em questão.

Static int getAnimalVida() - funções que retornam a duração do animal em questão.

Static int getAnimalPeso() - funções que retornam o peso do animal em questão.

```
static void setCoelhoSaude(int sCoelho);
static void setOvelhaSaude(int sOvelha);
static void setLoboSaude(int sLobo);
static void setCanguruSaude(int sCanguru);
static void setMisterioSaude(int sMisterio);
static void setCoelhoVida(int vCoelho);
static void setOvelhaVida(int vOvelha);
static void setLoboVida(int vLobo);
static void setCanguroVida(int vCanguro);
static void setMisterioVida(int vMisterio);
static void setCoelhoPeso(int pCoelho);
static void setOvelhaPeso(int pOvelha);
static void setLoboPeso(int pLobo);
static void setCanguruPeso(int pCanguru);
static void setMisterioPeso(int pMisterio);
```

Static void setAnimalSaude - funções que mudam a saúde do animal em questão para um valor em específico.

Static void setAnimalVida - funções que mudam a duração do animal em questão para um valor em específico.

Static void setAnimalPeso - funções que mudam o peso do animal em questão para um valor em especifico.

```
        private:
        static int sCoelho;
        static int sOvelha;
        static int sLobo;
        static int sCanguru;
        static int sMisterio;

        static int vCoelho;
        static int vOvelha;
        static int vLobo;
        static int vCanguru;
        static int vMisterio;

        static int pCoelho;
        static int pOvelha;
        static int pLobo;
        static int pCanguru;
        static int pMisterio;

};
```

static int sAnimal - variável para a saúde do animal em questão.

static int vAnimal - variável para a duração do animal em questão.

static int pAnimal - variável para o peso do animal em questão.

```
class CriarJogo{
        private:

        public:
            bool executarConfig(int &linhas, int &colunas);
};
```

bool executarConfig(int &linhas, int &colunas) - função que cria a tabela de jogo e lê valores do ficheiro constantes.txt

```
class Comandos {

public:
    Comandos(Reserva &reserva, map<string, Reserva *> &saveGames); //construtor dos comandos
    void inserirComandos(int linhasReservaInput, int colunasReservaInput);
    bool verificaComando(const string& comando); //refencia para a variável que só consegues ler

    map<string, Reserva *> &getSaveGames() const;

    Reserva & getReserva();
```

Comandos(Reserva &reserva, map<string>, Reserva *> &saveGames) - construtor dos comandos.

void inserirComandos(int linhasReservaInput, int colunasReservaInput) - função para inserir comandos com linha e coluna especifica.

bool verificaComando(const string& comando) - função que verifica se o comando inserido é possível de executar.

Reserva & getReserva() – função que retorna a referencia da reserva.

```
private:

    bool criaAnimal(const string& input);
    bool mataAnimal(const string& input);
    bool mataAnimalId(const string& input);
    bool colocaAlimento(const string& input);
    bool alimentaAnimais(const string& input);
    bool alimentaAnimaisId(const string& input);
    bool removeAlimento(const string& input);
    bool removeEspaco(const string& input);
    bool verEspaco(const string& input);
    bool veInformacao(const string& input);
    bool proximoInstante(const string& input);
    bool listarIDReserva(const string& input);
    bool ListarIDReservaVisivel(const string& input);
    bool ArmazenarEstado(const string& input);
    bool RecuperarEstado(const string& input);
    bool CarregarTxt(const string& input);
    bool MoverVisualizacao(const string& input);
```

bool comando(const string& input) - funções que representam os comandos possíveis de ser executados no jogo.

```
    int linhasReserva, colunasReserva;

    map<string, Reserva *> & saveGames;

    bool adicionaSaveGame(const string& nomeSave, Reserva * reservaSave);

    Reserva &reserva;//referncia para a reserva

    int posicaoZonaVisivelLinhas;
    int posicaoZonaVisivelColunas;
    const int limiteMaximoLinhasColunas;

};
```

int linhasReserva, colunasReserva - variáveis para as linhas e colunas da reserva especificadas durante a criação.

Reserva &reserva - referencia para a reserva.

int posicaoZonaVisivelLinhas - variável para a posição até onde as linhas visíveis são apresentadas.

int posicaoZonaVisivelColunas - variável para a posição até onde as colunas visíveis são apresentadas.

const int limiteMaximoLinhasColunas - variável para o limite máximo de linhas e colunas que o jogo pode ter.

3. Opções Tomadas

3.1 Animal Mistério

O animal mistério é representado por um “M”. Esta animal tem uma saúde inicial de “SMisterio =40”, 20kg de peso inicial e 20 de duração.

Este animal consegue aperceber-se do que o rodeia a 6 posições de distância na Reserva. Desloca-se 3 posições a cada instante aleatoriamente.

Quando se apercebe de um alimento com cheiro a “rosas” desloca-se para a sua posição e consome-o.

Morre quando a saúde ou duração chegam a 0.

A cada instante a fome aumenta 3. Se tiver mais do que 20 de fome passa a perder 4 de saúde e a deslocar-se 4 posições.

Gera um filho a cada 8 instantes a uma distância máxima de 4 posições.

3.2. Alimento Mistério

Representado por um “a”. Este alimento vai ter um valor nutritivo 5 e toxicidade 1.

Tem uma duração de 20 instantes e desaparece quando a duração chega a 0.

Tem cheiro de “rosas” e o seu valor nutritivo aumenta 1 a cada instante.

3.3. Obtenção dos valores através do input do utilizador

No comando executarConfig no “Comandos.cpp” optamos por perguntar ao utilizador por quantas linhas e por quantas colunas seria formada a reserva. Se o valor for invalido é pedido ao utilizador para inserir novamente as linhas e colunas.

3.4. Istringstream na manipulação de comandos

Para os comandos inseridos pelo utilizador ou lidos de um ficheiro, utilizamos um objeto do tipo istringstream. Este objeto permite-nos repartir o comando lido em comando principal e os seus argumentos. Através disto conseguimos realizar a validação do número de argumentos lidos e do seu tipo de dados

3.5. Classes derivadas

Estas classes foram usadas nos alimentos e animais. Para representar vários tipos destes objetos definimos várias classes que derivam funcionalidades especificas. Usámos funções virtuais, declarando as funções como virtuais nas classes base. Já que estas funções foram declaradas como virtuais nas classes, elas são virtuais em todas as classes derivadas desta. Nas classes derivadas para que uma função seja a "redefinição" de uma função virtual da classe base, utilizamos o identificador override para prevenir erros por parte do programador quando redefine a função.

4. Funcionalidades

- Funcionalidades Implementadas

- Coelho implementado na totalidade
- Ovelha implementada na totalidade
- Lobo implementado na totalidade
- Canguru implementado na totalidade
- Todos os alimentos estão implementados na totalidade
- Todos os comandos (exceto os que estão nas funcionalidades não implementadas)
- Histórico de Alimentação

- Funcionalidades Parcialmente Implementadas

- Animal Mistério

- Funcionalidades não implementadas

- Armazenar o estado da reserva em memória -> comando store
- Reativar um estado da reserva previamente armazenado em memória -> comando restore

5. Conclusão

A realização deste relatório esteve em concordância com o decorrer da unidade curricular de Programação Orientada a Objetos e chegamos à conclusão de que através da segmentação por vários ficheiros facilita a leitura e interpretação simulador de uma reserva natural povoada por diversos animais, além de que através da programação em C++ podemos usar classes e bibliotecas com propriedades e comportamentos diferentes.