Documentação
Caracóis Alados Famintos

Caio Vinicius Barbosa Santos - Caio.B.Santos@hotmail.com - 726503
Guilherme Franco Pires - nc.nd@hotmail.com – 726526
Leonardo Felipe de Souza Morais - lefeso@outlook.com - 726557
Otávio César Toma da Silva - otavio.cts@hotmail.com - 726576

São Carlos - 2017

Índice

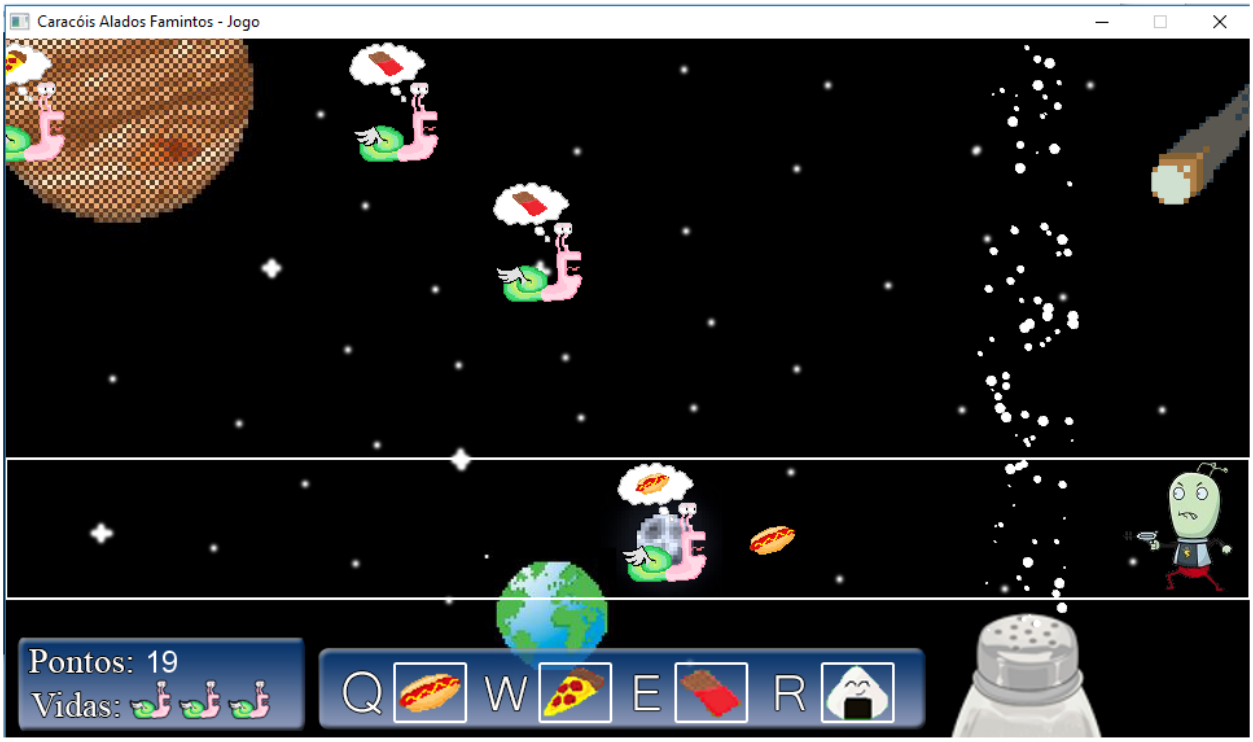
1. Introdução:

O jogo desenvolvido chama-se Caracóis Alados Famintos e em sua aplicação utiliza estruturas de dados do tipo Fila e a biblioteca gráfica Allegro 5. Ele consiste em 4 caminhos onde surgem caracóis que vão em direção a uma linha de sal, e o jogador deve selecionar esses caminhos e atirar em cada caracol a comida correta que ele está pensando, antes que ele chegue no sal.

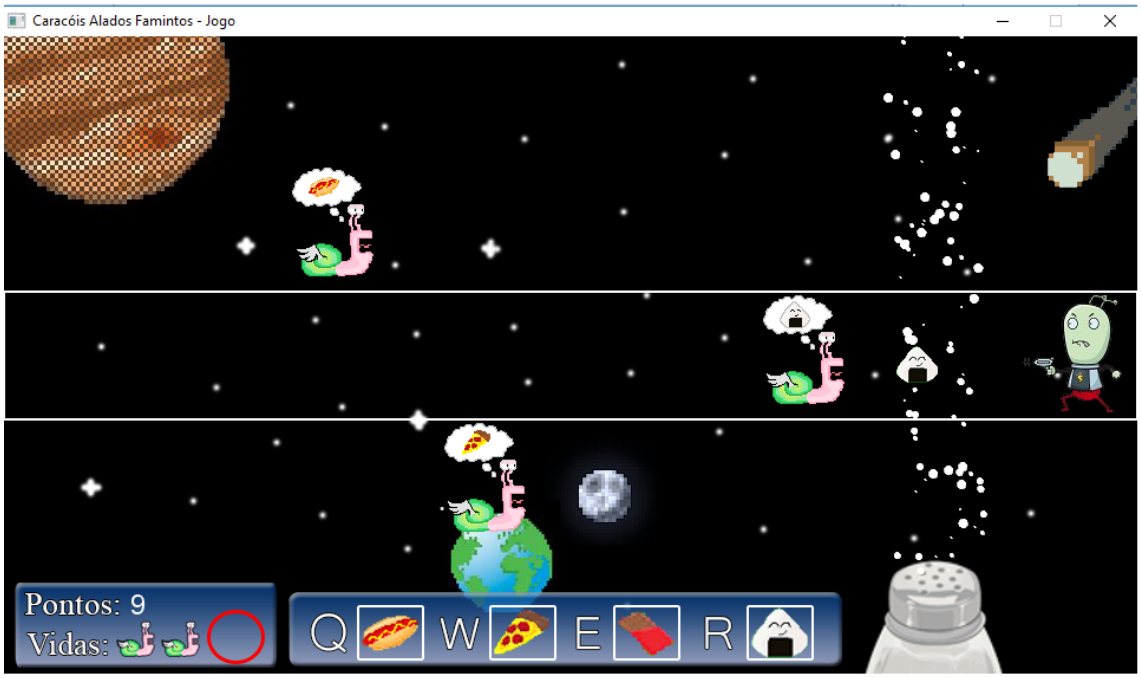
2. Descrição do jogo:

O programa se inicia com uma tela de menu onde o jogador, utilizando o mouse, pode escolher entre os botões “Jogar” ou “Sair”. Se ele clicar no botão “Sair”, então o jogo é fechado. Mas quando ele clica no botão “Jogar”, o jogo realmente começa indo para a tela principal do game e os caracóis começam a aparecer.

O jogador deve selecionar os caminhos por meio das setas para cima e para baixo do teclado, e atirar as comidas nos caracóis apertando as letras Q, W, E e R.
Cada tecla atira uma comida diferente e essa comida deve ser a mesma em que o respectivo caracol está pensando.



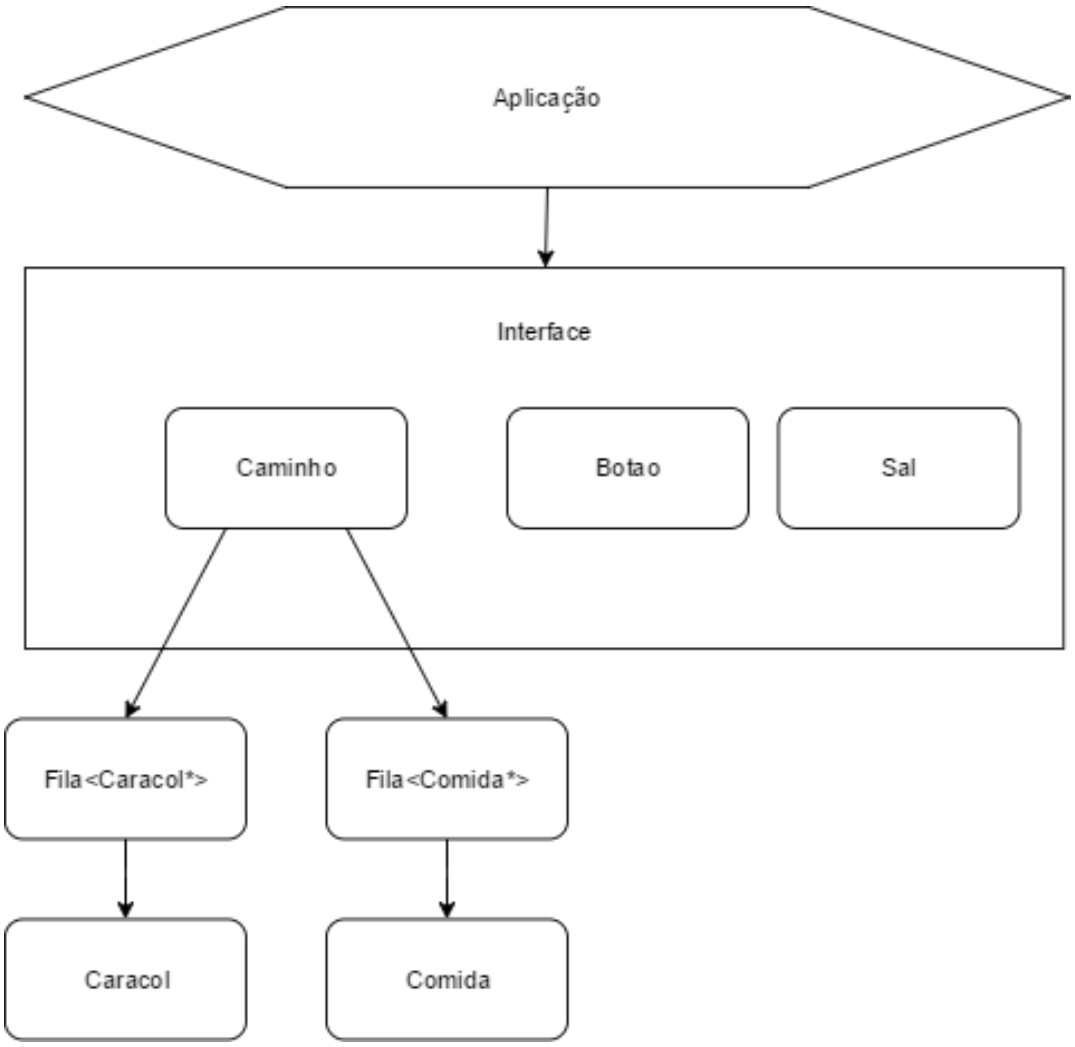
Quando houver a colisão entre uma comida e um caracol, se a comida for a mesma que ele está pensando, então eles somem e o jogador marca um ponto. Caso contrário, o caracol não some e o jogador perde uma vida.
O jogador também perde uma vida de algum caracol chegar no limite do caminho, que é uma linha de sal.



O objetivo é fazer o maior número de pontos até perder todas as 3 vidas!

Quando isso acontece, aparece a tela de game over, com as opções dos botões “Menu” para voltar para o menu e “Desistir” para sair do jogo.

3. Arquitetura do jogo:



As principais classes do jogo são as classes Caracol, Comida, Fila e Caminho.

Classe Caracol:

```
6   class Caracol{
7
8       int x, y, velocidade, altura, largura, contadorImg;
9       ALLEGRO_BITMAP *imagem[2];
10      int desejo;
11      static ALLEGRO_BITMAP *img_c_hotdog[2], *img_c_pizza[2], *img_c_chocolate[2], *img_c_sushi[2];
12
13      public:
14      Caracol();
15      Caracol(int x, int y, int velocidade, int altura, int largura);
16      ~Caracol();
17      void andar();
18      void desenhar();
19
20      int getX();
21      void setX(int);
22      int getY();
23      void setY(int);
24      int getVelocidade();
25      void setVelocidade(int);
26      int getAltura();
27      void setAltura(int);
28      int getLargura();
29      void setLargura(int);
30      int getDesejo();
31      void setDesejo(int);
32
33      static void inicializarImagens();
34
35  };
```

Classe Comida:

include\comida.h

```
6   class Comida{
7
8       int x, y, velocidade, altura, largura;
9       ALLEGRO_BITMAP *imagem;
10      int tipo;
11      static ALLEGRO_BITMAP *img_hotdog, *img_pizza, *img_chocolate, *img_sushi;
12
13      public:
14      Comida();
15      Comida(int x, int y, int velocidade, int altura, int largura, int tipo);
16      ~Comida();
17      void andar();
18      void desenhar();
19
20      int getX();
21      void setX(int);
22      int getY();
23      void setY(int);
24      int getVelocidade();
25      void setVelocidade(int);
26      int getAltura();
27      void setAltura(int);
28      int getLargura();
29      void setLargura(int);
30      int getTipo();
31      void setTipo(int);
32      static void inicializarImagens();
33
34  };
35
```

Cada um dos 4 caminhos é um objeto da classe Caminho, que contém uma fila de caracóis e uma fila de comidas.

```
class Caminho{

    int x0, y0, x1, y1;
    Fila<Caracol*> filaCaracois;
    Fila<Comida*> filaComidas;

    static ALLEGRO_BITMAP *img_alien;

    public:
    Caminho(int x0, int y0, int x1, int y1);
    ~Caminho();
    int getX0();
    void setX0(int);
    int getY0();
    void setY0(int);
    int getX1();
    void setX1(int);
    int getY1();
    void setY1(int);

    void adicionarCaracol();
    void removerCaracol();
    void desenharCaracois();
    void atualizarCaracois();

    void adicionarComida(int);
    void removerComida();
    void desenharComidas();
    void atualizarComidas();

    void desenhar();
    void verificaColisoes(int&, int&);

    void esvaziarFilaCaracois();
    void esvaziarFilaComidas();

    static void inicializarImagens();

};
```

Na classe Caminho foram criadas as funções adicionarCaracol() para criar um novo caracol na fila do caminho, removerCaracol() para remover o primeiro caracol da fila, desenharCaracois() para desenhar todos os caracóis da fila no display, e atualizarCaracois() para movimentar os caracois da fila a cada frame.

Também existem as funções adicionarComida(), removerComida(), desenharComidas() e atualizarComidas() para gerenciar a fila das comidas.

A função verificaColisoes(int& pontos, int& vidas) é muito importante, pois verifica se o primeiro elemento da fila de caracóis colidiu com o primeiro elemento da fila de comidas, além de detectar se o primeiro caracol da fila de caracóis chegou na linha de sal ou se a primeira comida da fila de comidas não acertou nenhum caracol e saiu da tela:

```
void Caminho::verificaColisoes(int& pontos, int& vidas){
    if (!filaCaracois.vazia() && !filaComidas.vazia()){
        if(filaCaracois.primeiroElem()->getX() > filaComidas.primeiroElem()->getX()){
            if(filaCaracois.primeiroElem()->getDesejo() == filaComidas.primeiroElem()->getTipo()){
                this->removerCaracol();
                this->removerComida();
                pontos++;
            }
            else{
                this->removerComida();
                vidas--;
            }
        }
    }
    if(!filaCaracois.vazia()){
        if(filaCaracois.primeiroElem()->getX()+filaCaracois.primeiroElem()->getLargura()/2 > 770){
            this->removerCaracol();
            vidas--;
        }
    }
    if(!filaComidas.vazia()){
        if(filaComidas.primeiroElem()->getX() < 0)
            this->removerComida();
    }
}
```

A classe Botão foi criada para representar os botões do menu e da tela de game over:

```
class Botao{
    private:
        int x, y, largura, altura;
        bool selecionado;
        ALLEGRO_BITMAP *imagem;
    public:
        Botao();
        Botao(int, int, string);
        ~Botao();
        void atualiza(int, int);
        bool estaSelecionado();
        void desenhar();
};
```

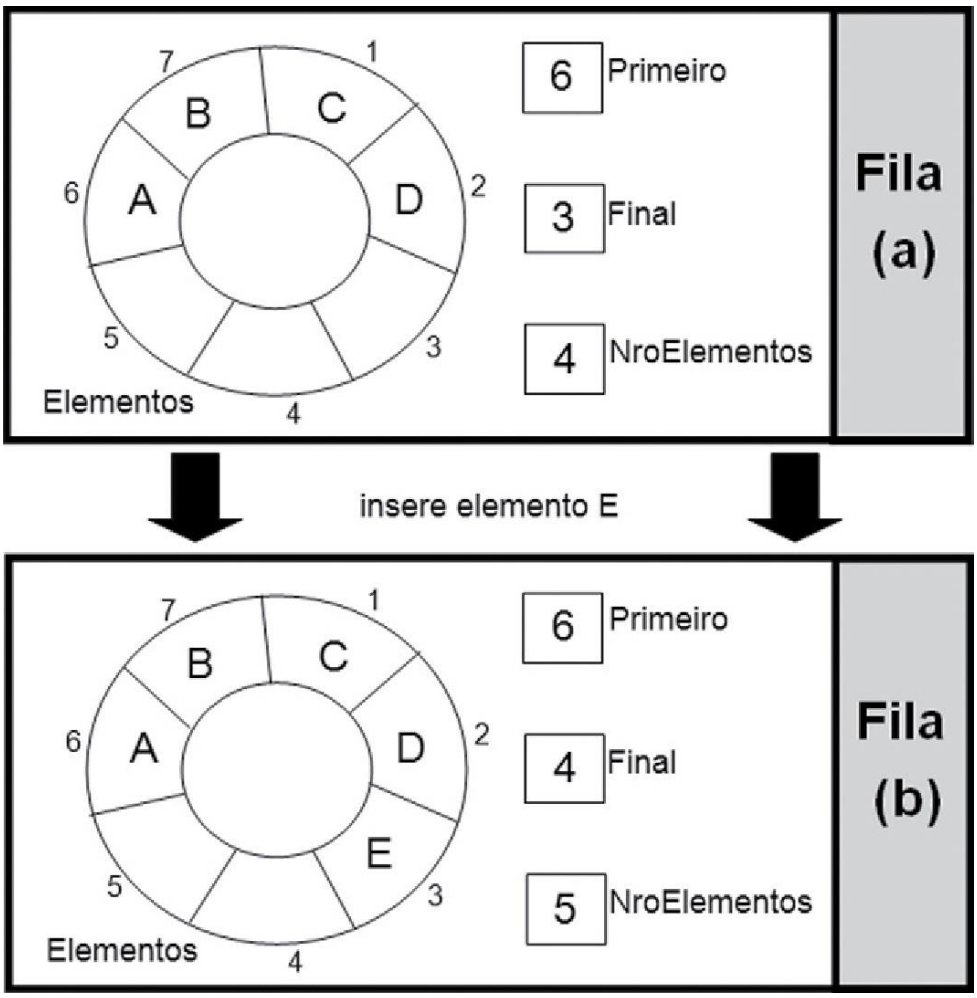
E a classe Sal foi utilizada para fazer a animação da linha de sal:

```
class Sal{
    private:
        double x, y, xInicial, yInicial, raio, vX, vY, dx;
    public:
        Sal(int minX, int maxX);
        void desenha();
        void move();
        int getY();
};
```

4. Implementação da fila:

No projeto escolhemos implementar a estrutura de dados do tipo Fila, como dito anteriormente, e utilizamos um template para proporcionar mais portabilidade e reusabilidade do código, podendo assim servir para outros fins além desse jogo.

O tipo de Fila escolhido foi circular, não encadeada e com alocação sequencial e estática de memória:



Classe Fila:

```
template <class T>
class Fila{
    private:
        T *vetor;
        int primeiro, ultimo, tam, numElementos;
    public:
        Fila();
        Fila(int);
        bool vazia();
        bool cheia();
        bool insere(T);
        bool remove();
        T primeiroElem();
        int getTam();
        int getNumElementos();
        T operator[](int);
};
```

Implementação:

Fila<T>::Fila():

```
template <class T>
Fila<T>::Fila(int tam){
    vetor = new T[tam];
    primeiro = -1;
    ultimo = -1;
    this->tam = tam;
    numElementos = 0;
}
```

Construtor vazio utilizado na classe 'Caminho' onde criamos atributos do tipo Fila.

Fila<T>::Fila(int tam):

```
12  template <class T>
13  Fila<T>::Fila(int tam){
14      vetor = new T[tam];
15      primeiro = -1;
16      ultimo = -1;
17      this->tam = tam;
18      numElementos = 0;
19  }
```

Construtor que recebe como argumento um inteiro (tamanho máximo) e inicializa a Fila como vazia.

bool Fila<T>::vazia():

```
template <class T>
bool Fila<T>::vazia(){
    if(numElementos == 0)
        return true;
    return false;
}
```

Método que verifica a condição da fila, se está vazia ou não, se estiver vazia retorna verdadeiro, senão retorna falso.

bool Fila<T>::cheia();

```
template <class T>
bool Fila<T>::cheia(){
    if(numElementos == tam)
        return true;
    return false;
}
```

Método que assim como o anterior verifica a condição da Fila, dessa vez se a Fila está cheia, se o número de elementos da Fila for igual ao seu tamanho significa que a Fila estará cheia, portanto retornará o valor 'verdadeiro', caso o contrário retorna 'falso'.

bool Fila<T>::insere(T elem);

```
template <class T>
bool Fila<T>::insere(T elem){
    if(cheia())
        return false;
    if(primeiro == -1)
        primeiro++;
    ultimo++;
    if(ultimo == tam)
        ultimo = 0;
    vetor[ultimo] = elem;
    numElementos++;
    return true;
}
```

Método responsável por inserir um novo elemento na Fila. Para isso primeiramente verificamos se a Fila já está cheia, pois se estiver não será possível inserir novos elementos e portanto retornando o valor 'falso'; se a Fila não estiver cheia um novo elemento é inserido ao final da Fila retornando o valor 'verdadeiro'.

bool Fila<T>::remove();

```
template <class T>
bool Fila<T>::remove(){
    if(vazia())
        return false;
    primeiro++;
    if(primeiro == tam)
        primeiro = 0;
    numElementos--;
    return true;
}
```

Método responsável por remover o primeiro elemento da Fila. Primeiro é verificado se a Fila não está vazia, retornando o valor 'falso' se estiver, pois

não seria possível remover um elemento de uma Fila que não possui nenhum; caso contrário remove o primeiro elemento da Fila retornando assim o valor 'verdadeiro'.

T Fila<T>::primeiroElem() e int Fila<T>::getTam():

```
template <class T>
T Fila<T>::primeiroElem(){
    return vetor[primeiro];
}

template <class T>
int Fila<T>::getTam(){
    return tam;
}
```

A função primeiroElem() retorna o primeiro elemento da Fila e getTam() retorna o tamanho atual da Fila.

int Fila<T>::getNumElementos():

```
template <class T>
int Fila<T>::getNumElementos(){
    return numElementos;
}
```

Retorna o número de elementos máximo da Fila.

T Fila<T>::operator[](int i):

```
template <class T>
T Fila<T>::operator[](int i){
    if(i < tam-primeiro)
        return vetor[primeiro+i];
    else
        return vetor[i-(tam-primeiro)];
}
```

Utilizamos a sobrecarga do operador [ ] para acessar qualquer elemento da Fila. Precisamos desse método para acessar todos os elementos eficientemente e, por exemplo, desenhar na tela todos elementos da Fila, como caracóis ou comidas.

5. Conclusão:

O grupo evoluiu muito durante o desenvolvimento do jogo. Foi bem divertido e ao mesmo tempo foi possível aprender como é feito um jogo: movimentação, animação, efeitos sonoros, atualização das imagens na tela, etc.

Além disso, aprendemos a desenvolver uma estrutura de dados do tipo Fila, implementando-a como um tipo abstrato de dado.

A única dificuldade encontrada foi na busca de horários em que todos os participantes pudessem se reunir para finalizar o trabalho. Mas isso foi superado, gerando resultados bem satisfatórios no final.