Universidade Federal de São Carlos – UFSCar

Bacharelado em Ciência da Computação

Estrutura de Dados

Professor: Ferrari

Jogo: JvAdventure

Gabrieli Santos, 726523, gabrielisantos17.gs@gmail.com

João Gabriel Barbirato, 726546, e-mail: joaobarbirato@gmail.com

Leonardo de Oliveira Peralta, 726556, leonardo98ti@gmail.com

Sumário

1-Introdução

Promover atividades práticas facilita a transmissão e a fixação de conhecimento. Sendo assim, a disciplina de Estruturas de Dados ministrada no primeiro semestre de 2017 no Departamento de Computação da Universidade Federal de São Carlos (DC-UFSCar) estabelece a elaboração de outro jogo - envolvendo estruturas de dados - para fins avaliativos.

Ao longo da disciplina foi apresentado o tipo abstrato de dados denominado como lista, bem como suas características e exemplos de uso. Portanto, essa Documentação trata de um jogo envolvendo uma lista.

Essa estrutura permite o mapeamento de todos os elementos pertencentes à fila sem a necessidade de retirar qualquer elemento. Não obstante, permite que seja removido ou inserido um elemento em qualquer posição (ou de acordo com algum critério pré-estabelecido).

Além da estrutura, é necessária a utilização de interfaces gráficas. Assim, optou-se por utilizar a biblioteca SFML.

O jogo é inspirado no filme “Tron: O Legado” da Disney e na série de jogos de aventura do Megaman. Esse programa consiste em um cenário onde adversários tentam constantemente abater o jogador. Para isso, esse conta com uma lista de armas para se defender e sobreviver. O objetivo é sobreviver ao cenário repleto de inimigos por cinco minutos.

A principal estrutura escolhida foi uma Lista, tipo abstrato de dados que permite acesso a qualquer elemento, em qualquer posição. Implementou-se essa lista para facilitar as demandas do jogo, uma vez que o jogo requer a constante verificação de dois elementos que se interagem (tais como um projétil e um alvo) e a versatilidade da lista permite essa verificação sem muito custo computacional.

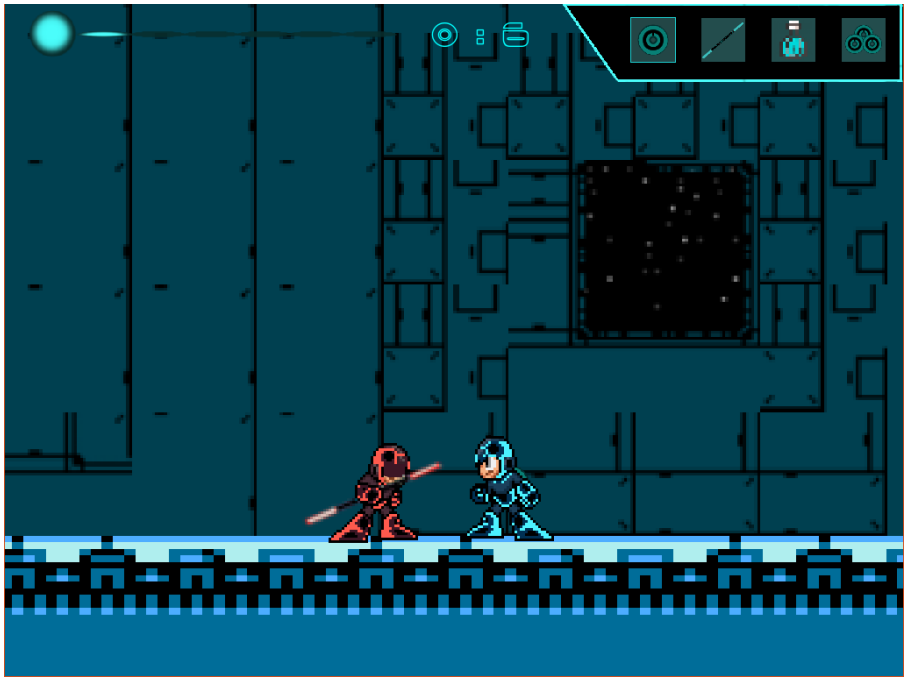
2-Desenvolvimento

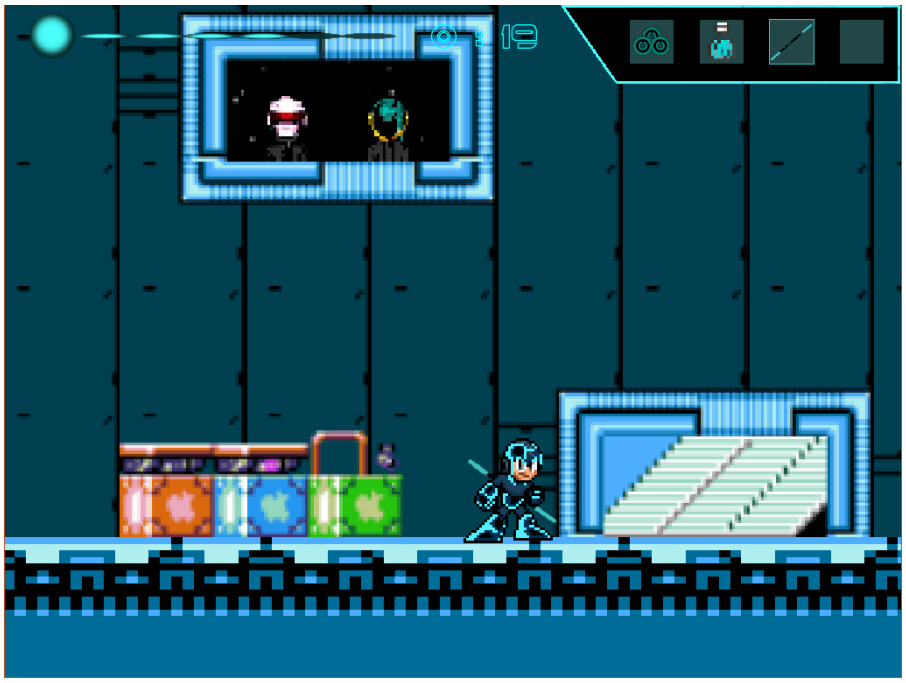
a) Prints da execução

Ao iniciar o jogo, o usuário se depara com a tela de menu principal:

Caso o usuário queira saber como funciona o jogo, clica-se em Regras e têm-se as seguintes telas:

Ao clicar em Jogar, o usuário entra no cenário e interage com as entidades principais do programa, como inimigos e itens.





É fácil perceber que, no início do jogo há a primeira e mais visível lista, a de itens.

Caso o jogador consiga sobreviver a todos os desafios no jogo, aparece a seguinte tela:

Caso contrário, a seguinte tela aparece:

b) Estrutura da Lista

A lista utilizada foi implementada num arquivo chamado Lista.hpp. Esse arquivo contempla uma Implementação de lista utilizando operações de baixo nível em nós. A imagem abaixo ilustra os atributos e os métodos (operadores) da lista implementada.

```
14  template<class Gen>
15  class Lista{
16      private:
17          int quant;
18          struct Node<Gen> *Primeiro;
19          struct Node<Gen> *Atual;
20          struct Node<Gen> *header;
21      public:
22          Lista();
23          ~Lista();
24          void PegaOProximo(Gen&,bool&);
25          void PegaOPrimeiro(Gen&,bool&);
26          void cria();
27          void limparLista();
28          int getQuant();
29          void insere(Gen&,bool&);
30          bool estaNaLista(Gen& x);
31          void insereAEsquerdaDeP(Gen&,bool&);
32          void removeP(Gen&,bool&);
33          void remove(Gen& x,bool& deuCerto);
34          void removePCaixa(Item& x,bool& deuCerto);
35          bool estaNaListaCaixa(Gen& x);
36          void atualizaP(Gen& x,bool& deuCerto);
37  };
```

Muito embora tenha-se usado *templates*, houve necessidade de implementar diferentes assinaturas do método para facilitar seu uso.

A facilidade de percorrer os elementos da lista apenas com o movimento de um ponteiro (métodos *PegaOPrimeiro* e *PegaOProximo*) permitiu que o jogo tivesse constantes verificações de interações entre entidades do software. Por exemplo, o trecho a seguir mostra o procedimento de verificação para a perda de vida do protagonista, ou seja, quando um dos elementos da lista de projéteis (discos) do antagonista interage com o protagonista:

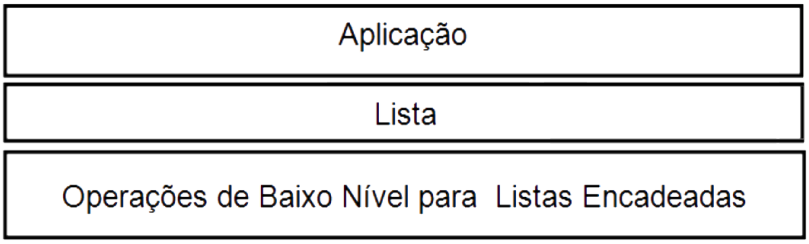
```
        //percorre os discos do heroi
         int quantDiscosInimigo = discosInimigos.getQuant();
        for(i=0;i<quantDiscosInimigo;i++){
             Disco discoAuxInimigo;
            if(i == 0){
                discosInimigos.PegaOPrimeiro(discoAuxInimigo,deuCerto);
            }else{
                discosInimigos.PegaOProximo(discoAuxInimigo,deuCerto);
            }
            discosInimigos.removeP(discoAuxInimigo,deuCerto);
            if(discoAuxInimigo.getPosicao().x <= view.getCenter().x + 600 && discoAuxInimigo.getPosicao().x >view.getCenter().x - 600 ){
                 if(discoAuxInimigo.Bateu(tron.animatedSprite) ){
                     tron.perdeVida();
                 }else{
                    App.draw(discoAuxInimigo.desenho);
                    discoAuxInimigo.mover(tron.getAndando(),tron.getDirecao(),tron.getPulando());
                    discosInimigos.insere(discoAuxInimigo,deuCerto);
                }
            }
        }
```

Outro exemplo de uso dos métodos da lista está na lista de itens. Essa é visível durante todo o jogo, uma vez que o protagonista está constantemente removendo, inserindo ou alterando itens na lista, podendo remover qualquer item em qualquer lugar da lista. O código abaixo ilustra essa interação:

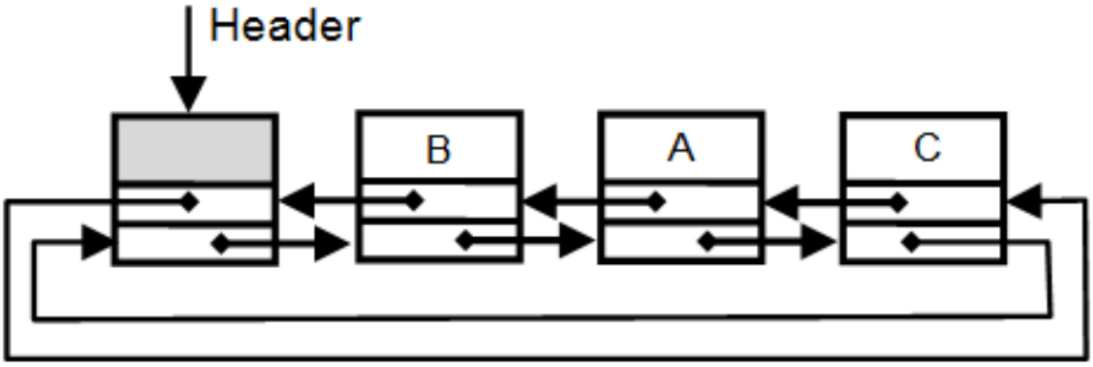
```
void BarraItens::setPosicaoItens(int a){
    bool deuCerto;
    Item auxItem;
    int i=0;
    for(i=0;i<4;i++){
        idItemCaixa[i]=0;
    }
    i=0;
    itens.PegaOPrimeiro(auxItem,deuCerto);
    while(deuCerto){
        idItemCaixa[i]=auxItem.getId();
        auxItem.idCaixa=i;
        auxItem.setPosicao(centralizar(caixaItem[i].getPosition(),auxItem.getTamanho(),auxItem.getId()));
        itens.atualizaP(auxItem,deuCerto);
        i++;
        itens.PegaOProximo(auxItem,deuCerto);
    }

};
```

c) Diagrama e arquitetura do software

Nesse projeto, a lista não segue nenhum critério – em termos de implementação do TAD Lista – para inserção ou remoção de elementos. Dos elementos da aplicação, manipula-se indiretamente as operações da lista, as quais foram implementadas utilizando operações de baixo nível. Portanto, a arquitetura do software funciona conforme o diagrama abaixo:



d) Implementação da Lista



Trata-se de uma lista generalizada, duplamente encadeada, com nó *header*, com quantidade de elementos, *template* e operações de baixo nível. No jogo, instanciam-se quatro listas: os discos atirados pelos inimigos, os discos atirados pelo protagonista, os itens que aparecem no cenário e os itens que o protagonista possui.

e) Implementação

Utilizou-se a linguagem C++ e a biblioteca de interface gráfica SFML. Implementou-se uma solução seguindo o paradigma de programação orientada a objetos. As classes foram escritas em arquivos HPP (arquivos *header* para a linguagem C++) para melhor organização de códigos. Entre esses estão as telas, gerenciadas na função principal (*main*).

O projeto conta com 16 classes, 1 arquivo CPP (da função principal *main*) (ou seja, arquivos HPP) e 89 imagens PNG.

Além disso, todos os códigos estão disponíveis abertamente junto dessa Documentação, bem como no repositório https://github.com/gabrielissantos/JvTron-2.0.

3-Conclusão

Diferentemente do projeto anterior, o grupo conseguiu implementar constantemente com a utilização da biblioteca gráfica SFML utilizando sistemas Linux. Além disso, por experiência do projeto anterior, a utilização das classes específicas da biblioteca gráfica tornou-se mais fluida e fácil.

Não obstante, o TAD lista apresentou-se de fácil utilização dos seus operadores. Mesmo com uma implementação de baixo nível de nós, a estrutura apresentou-se versátil e trouxe vantagens computacionais que permitiram o surgimento de novas funcionalidades no projeto.

A ideia do jogo parecia simples. Contudo, as tentativas falhas de implementação, as constantes falhas de performance e a complexidade que o jogo tomou atrasaram o desenvolvimento.

Planejou-se para o desenvolvimento a seguinte divisão: o João como responsável pela parte gráfica e pela parte de *frontend* (interfaces de interação e exibição, como menu e telas de regras) e Leonardo como *backend* (implementação da lógica e do funcionamento), enquanto que Gabrieli cuidava, em primeira instância, do desenvolvimento do próximo projeto e das músicas presentes no jogo. Essa divisão conseguiu perdurar por todo o período de desenvolvimento.

Assim, o maior desafio foi desenvolver dois projetos de ideias complexas em um período curto de tempo. Felizmente, conseguiu-se realizar o projeto e desenvolver uma aplicação funcional utilizando conceitos de Estruturas de Dados vistos em aula.