Universidade Federal de São Carlos
Departamento de Computação

Disciplina: Estruturas de Dados
Professor Roberto Ferrari

JOGO 1- LETRIS v2.0

Integrantes:
Bruna Zamith
João Victor Pacheco
Marcos Faglioni
Rodrigo Salmen

Desenvolvedores:

Bruna Zamith	bruna.zamith@hotmail.com
João Victor Pacheco	jvp1805@gmail.com
Marcos Faglioni	marcosfagli@hotmail.com
Rodrigo Salmen	rodrigosalmen2012@hotmail.com

OBJETIVOS

A proposta inicial deste jogo era a de implementar um jogo no estilo Tetris, só que ao invés de blocos, que ao completarem uma linha, fazem com que a mesma desapareça, a ideia era a de fazerem colunas, as quais seriam pilhas e nessas colunas, o usuário colocava letras, de forma a montar palavras pré-determinadas no início do jogo. Assim que uma palavra fosse formada dentro de uma pilha, todas as letras dessa palavra seriam desempilhadas. Além disso, haveriam também caracteres estratégicos, que cairiam e estes seriam capazes de ser usados para desempilhar uma letra colocada errada.

METODOLOGIA

Para desenvolver os TAD's deste projeto, utilizamos a linguagem C++. Nele implementamos inicialmente uma classe Pilha.

Uma pilha é uma estrutura utilizada para armazenar elementos de modo que, novos elementos sempre entram nesse conjunto no topo da pilha e o unico elemento que pode ser retirado de uma pilha é o topo da mesma. Elas obedecem o critério *L.I.F.O (Last In First Out)*, ou seja, o ultimo elemento a entrar (ou ser armazenado) será o primeiro a sair (ou ser retirado).

A Pilha utilizada conta com os seguintes parâmetros e funcionalidades:

int Tamanho;	**retorna o tamanho da pilha.**
int Topo;	**retorna o valor do topo da pilha.**
T* Elementos;	**variável para os elementos da pilha (de tipo não definido).**
Pilha(**int** tam)	**construtor da pilha (age como um CriaPilha()).**
void Empilha(**T** X, **bool&** DeuCerto)	**Função para empilhar um elemento na pilha**
void Desempilha(**T** X, **bool&** DeuCerto)	**Função para desempilhar um elemento da pilha**
bool Vazio();	**Retorna TRUE se a pilha estiver vazia**
bool Cheia();	**Retorna TRUE se a pilha estiver cheia**

A função Pilha(int tam) recebe como parâmetro tam, que é o tamanho que a pilha criada deve ter. Caso tam receba algum numero invalido (menor que 0), por default, a pilha recebe o tamanho 10.

A função void Empilha(T X, bool& DeuCerto) verifica se a pilha está cheia, caso esteja, DeuCerto recebe false, senão, a variavel topo é incrementada, o elemento do topo da pilha recebe o valor X e DeuCerto recebe true.

A função void Desempilha(T X, bool& DeuCerto) verifica se a pilha está vazia, caso esteja, DeuCerto recebe false, senão, a variável topo é decrementada, X recebe o valor que está no elemento do topo e DeuCerto recebe true.

A função Vazio() verifica se topo é menor que zero, se for, isso significa que a pilha está vazia, e a função retorna true, senão, a função retorna false.

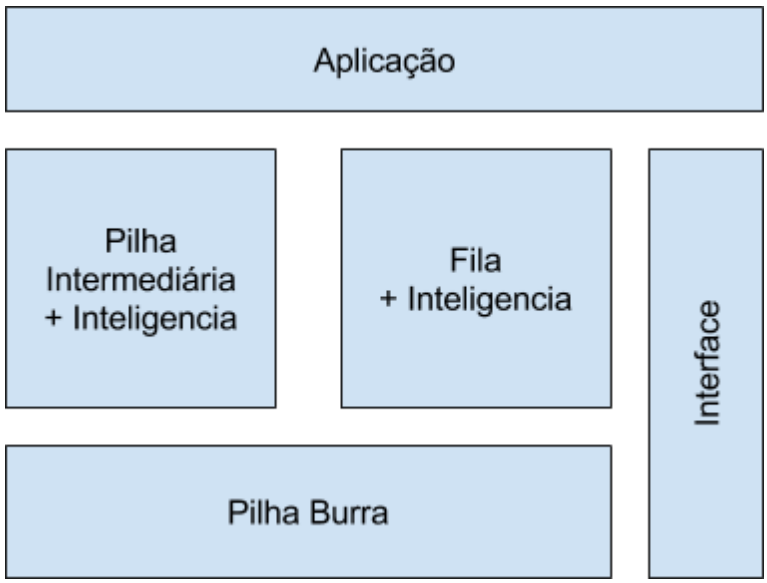
A função Cheia() verifica se o topo é maior ou igual a Tamanho -1, caso seja, isso significa que a pilha está cheia, e a função retorna true, senão, a função retorna false.

Na figura a seguir podemos ver a implementação da Classe Pilha:

```cpp
template <class T>

class Pilha
{
public:
     Pilha(int tam);
     ~Pilha();
     void Empilha(T X, bool& DeuCerto);
     void Desempilha(T& X, bool& DeuCerto);
     bool Vazio();
     bool Cheia();

     int Tamanho;
     int Topo;
     T* Elementos;
};

template <class T>
Pilha<T>::Pilha(int tam)
{
     if (tam > 0)
          Tamanho = tam;
     else
          Tamanho = 10;

     Topo = -1;
     Elementos = new T[Tamanho];
}

template <class T>
Pilha<T>::~Pilha()
{
     delete[] Elementos;
}

template <class T>
void Pilha<T>::Empilha(T X, bool& DeuCerto)
{
     if (Cheia())
     {
          DeuCerto = false;
     }
     else
     {
          Topo++;
          Elementos[Topo] = X;
          DeuCerto = true;
     }
}

template <class T>
void Pilha<T>::Desempilha(T& X, bool& DeuCerto)
{
     if (Vazio())
     {
          DeuCerto = false;
     }
     else
     {
          X = Elementos[Topo];
          Topo--;
          DeuCerto = true;
     }
}

template <class T>
bool Pilha<T>::Vazio()
{
     if (Topo < 0)
          return true;
     else
          return false;
}

template <class T>
bool Pilha<T>::Cheia()
{
     if (Topo >= Tamanho - 1)
          return true;
     else
          return false;
}
```

DESENVOLVIMENTO

Arquitetura de Software



Tem-se inicialmente implementada a pilha burra supracitada, a partir delas, foi implementado a inteligencia para essas pilhas, com destaque principal para a função que verifica se uma palavra está presente na pilha. A função recebe por parâmetro uma pilha e um vetor de strings, seu objetivo é procurar as palavras do vetor na pilha e caso a encontre tanto de cima para baixo, quanto de baixo para cima, e eliminar a palavra, caso a mesma seja encontrada.

Ela faz isso, desempilhando as letras da pilha e empilhando numa pilha auxiliar até encontrar a primeira letra da primeira palavra. Se ela encontrar, ela enfileira a letra numa fila auxiliar e continua fazendo isso para as letras seguintes. Caso ela comece a encontrar uma palavra, e no meio da mesma, apareça uma letra diferente da que deveria aparecer para completar a palavra, ela desenfileira a fila inteira e volta a procurar novamente a primeira letra (e.g. ele econtra as letras CARR e na quinta letra, ele encontra J. Ele percebe que J é diferente de O e portanto não forma a palavra carro, portanto ele desenfileira todos os elementos que já estavam na pilha, CARR, e joga eles para a pilha). Quando a pilha primária está vazia, ela desempilha os elementos da pilha auxiliar e empilha na pilha primária, para voltar a ficar como estava antes, com todas as letras ou sem as letras da palavra que foram desempilhadas.

```
#include<iostream>
#define TAM_LETRAS 50 //tamanho do quadrado das letras
#define DIST_PILHAS 15 //distancia entre as pilhas
#define PALAVRASDISPONIVEIS 10 //qntd de palavras no banco de palavras
#define PALAVRASSELECIONADAS 3 //palavras selecionadas por jogo
#define TAM_PALAVRA 5 //qntd de letras das palavras do banco de palavras

//Esta na Pilha
bool estaNaPilha(Pilha<char> *palavras, string selecionadas[PALAVRASSELECIONADAS]) {
     bool DeuCerto;
     bool AchouPalavra = false;
     int k = 0;
     char X, Y;
     Pilha<char> Paux(palavras->Tamanho);
     Pilha<char> Paux2(palavras->Tamanho);
     Pilha<char> Paux3(palavras->Tamanho);

     for (int i = 0; i<PALAVRASSELECIONADAS; i++) {
          while (palavras->Vazio() == false && k != TAM_PALAVRA) {
               palavras->Desempilha(X, DeuCerto);
               if (X == selecionadas[i][k]) {
                    k++;
                    Paux2.Empilha(X, DeuCerto);
               }
               else {
                    k = 0;
                    Y = X;
                    while (Paux2.Vazio() == false) {
                         Paux2.Desempilha(X, DeuCerto);
                         Paux3.Empilha(X, DeuCerto);
                    }
                    while (Paux3.Vazio() == false) {
                         Paux3.Desempilha(X, DeuCerto);
                         Paux.Empilha(X, DeuCerto);
                    }
                    Paux.Empilha(Y, DeuCerto);
               }
          }
          if (k == TAM_PALAVRA) {
               while (Paux.Vazio() == false) {
                    Paux.Desempilha(X, DeuCerto);
                    palavras->Empilha(X, DeuCerto);
               }
               AchouPalavra = true;
          }
          else {
               k = 0;
               while (Paux2.Vazio() == false) {
                    Paux2.Desempilha(X, DeuCerto);
                    Paux3.Empilha(X, DeuCerto);
               }
               while (Paux3.Vazio() == false) {
                    Paux3.Desempilha(X, DeuCerto);
                    Paux.Empilha(X, DeuCerto);
               }
               while (Paux.Vazio() == false) {
                    Paux.Desempilha(X, DeuCerto);
                    if (X == selecionadas[i][k]) {
                         k++;
                         Paux2.Empilha(X, DeuCerto);
                    }
                    else {
                         k = 0;
                         Y = X;
                         if (X == selecionadas[i][k]) {
                              while (Paux2.Vazio() == false) {
                                   Paux2.Desempilha(X, DeuCerto);
                                   Paux3.Empilha(X, DeuCerto);
                              }
                              while (Paux3.Vazio() == false) {
                                   Paux3.Desempilha(X, DeuCerto);
                                   palavras->Empilha(X, DeuCerto);
                              }
                              Paux2.Empilha(Y, DeuCerto);
                              k++;
                         }
                         else {
                              while (Paux2.Vazio() == false) {
                                   Paux2.Desempilha(X, DeuCerto);
                                   Paux3.Empilha(X, DeuCerto);
                              }
                              while (Paux3.Vazio() == false) {
                                   Paux3.Desempilha(X, DeuCerto);
                                   palavras->Empilha(X, DeuCerto);
                              }
```

```
87                              palavras->Empilha(Y, DeuCerto);
88                          }
89                      }
90                  }
91                  if (k < TAM_PALAVRA) {
92                      while (Paux2.Vazio() == false) {
93                          Paux2.Desempilha(X, DeuCerto);
94                          Paux3.Empilha(X, DeuCerto);
95                      }
96                      while (Paux3.Vazio() == false) {
97                          Paux3.Desempilha(X, DeuCerto);
98                          palavras->Empilha(X, DeuCerto);
99                      }
100                 }
101                 else if (k == TAM_PALAVRA) {
102                     AchouPalavra = true;
103                 }
104             }
```

A função abaixo é responsável por imprimir os elementos da pilha no console. A cada interação, ela desempilha a pilha P, para que X receba o valor do elemento do topo, imprime o valor na tela utilizando a chamada cout, e empilha o valor X numa pilha auxiliar para não perdê-lo

```
14  void imprimePilha(Pilha<char> *P) {
15      Pilha <char> Paux(P->Tamanho);
16      char X;
17      bool DeuCerto;
18
19      while (P->Vazio() == false) {
20          P->Desempilha(X, DeuCerto);
21          Paux.Empilha(X, DeuCerto);
22          cout << X << " ";
23      }
24      while (Paux.Vazio() == false) {
25          Paux.Desempilha(X, DeuCerto);
26          P->Empilha(X, DeuCerto);
27      }
28  }
29
30
```

Após a impressão de todos os elementos da pilha, a função percorre outro laço, para retornar os elementos da pilha auxiliar para a pilha principal, utilizando novamente as funções desempilha e empilha.

```
C:\Users\João Victor\Documents\letris\Letris.versao2\Release\Letris.versao2.exe
Letras Restantes: 288
Empilhou em 2
Palavras selecionadas:
1. placa
2. parto
3. preto
Letra caindo: o
a
*
p p r
c *
l

Letras Restantes: 287
Empilhou em 2
Palavras selecionadas:
1. placa
2. parto
3. preto
Letra caindo: *
a
*
o p p r
c *
l

Letras Restantes: 286
```

A seguinte função, se refere ao caractere especial - * - que serve para desempilhar a pilha inteira, caso esse seja colocado numa pilha.

```
void Destroi(float x, Pilha<char> *pilha0, Pilha<char> *pilha1, Pilha<char> *pilha2, Pilha<char> *pilha3, Pilha<char> *pilha4, Pilha<char> *pilha5, sf::Sprite botao[]) {
    int distanciaTotal = TAM_LETRAS + DIST_PILHAS;
    bool DeuCerto = true;
    int j;
    char k;
    if (x == 0) {
        while (DeuCerto)
            pilha0->Desempilha(k, DeuCerto); //desempilha da pilha
        //remove as sprites
        for (j = 0; j < NUM_LETRAS; j++) {
            if (botao[j].getPosition().x == x && botao[j].getPosition().y!=0) {
                botao[j].setColor(sf::Color::Black);
                botao[j].setPosition(sf::Vector2f(0, 0));
            }
        }
    }
    if (x == distanciaTotal) {
        while (DeuCerto)
            pilha1->Desempilha(k, DeuCerto); //desempilha da pilha
                                             //remove as sprites
        for (j = 0; j < NUM_LETRAS; j++) {
            if (x == botao[j].getPosition().x) {
                botao[j].setColor(sf::Color::Black);
                botao[j].setPosition(sf::Vector2f(0, 0));
            }
        }
    }
    if (x == distanciaTotal * 2) {
        while (DeuCerto)
            pilha2->Desempilha(k, DeuCerto); //desempilha da pilha
                                             //remove as sprites
        for (j = 0; j < NUM_LETRAS; j++) {
            if (x == botao[j].getPosition().x) {
                botao[j].setColor(sf::Color::Black);
                botao[j].setPosition(sf::Vector2f(0, 0));
            }
        }
    }
    if (x == distanciaTotal * 3) {
        while (DeuCerto)
            pilha3->Desempilha(k, DeuCerto); //desempilha da pilha
                                             //remove as sprites
        for (j = 0; j < NUM_LETRAS; j++) {
            if (x == botao[j].getPosition().x) {
                botao[j].setColor(sf::Color::Black);
                botao[j].setPosition(sf::Vector2f(0, 0));
            }
        }
    }
    if (x == distanciaTotal * 4) {
        while (DeuCerto)
            pilha4->Desempilha(k, DeuCerto); //desempilha da pilha
                                             //remove as sprites
        for (j = 0; j < NUM_LETRAS; j++) {
            if (x == botao[j].getPosition().x) {
                botao[j].setColor(sf::Color::Black);
                botao[j].setPosition(sf::Vector2f(0, 0));
            }
        }
    }
    if (x == distanciaTotal * 5) {
        while (DeuCerto)
            pilha5->Desempilha(k, DeuCerto); //desempilha da pilha
                                             //remove as sprites
        for (j = 0; j < NUM_LETRAS; j++) {
            if (x == botao[j].getPosition().x) {
                botao[j].setColor(sf::Color::Black);
                botao[j].setPosition(sf::Vector2f(0, 0));
            }
        }
    }
}
```

Basicamente, essa funcionalidade serve para ser utilizada quando o usuário incluiu letras erradas na pilha e quer eliminá-la. Esse caractere é capaz de

eliminá-la, chamando a função desempilha, e retirando todos os elementos que estavam na pilha.

Podemos ver um exemplo da aplicação deste operador na figura abaixo, onde temos numa pilha as letras P,E,G,E, e queremos desempilhar tudo, portanto colocamos o (*) nesta pilha e assim, desempilhamos ela completamente

Aqui temos o inicio da função Main(), que mostra a tela inicial, seleciona-se opção de instruções, ou inicia o jogo, selecionando a dificuldade e então chama a função ChamaTelaJogo() e caso a pessoa perca o jogo, chama a função TelaPerdeu() que imprime ao usuário, que ele perdeu o jogo e pergunta qual a próxima ação que ele quer tomar.

```
60  int main() {
61      sf::RenderWindow window(sf::VideoMode(600, 600), "Letris 2"); //Inicializa a tela
62      int tela;
63      int pontuacao = 0;
64      int dificuldade;
65      tela = menuInicial(window);
66      if (tela == 1) {
67          dificuldade = telaDificuldade(window);
68          tela = ChamaTelaJogo(window, dificuldade, pontuacao);
69          telaPerdeu(window, pontuacao);
70      }
71      if (tela == 2) {
72          tela = tela1Instucoes(window);
73          tela = tela2Instucoes(window);
74          tela = tela3Instucoes(window);
75          tela = tela4Instucoes(window);
76          dificuldade = telaDificuldade(window);
77          tela = ChamaTelaJogo(window, dificuldade, pontuacao);
78          telaPerdeu(window, pontuacao);
79      }
80      return 0;
81  }
```

A seguir, temos a função ChamaTelaJogo, função onde incialmente se carrega as fontes qeu serão utilizadas e cria todas as pilhas com tamanho fixo Tamy(tamanho de Y), após isso, carrega as texturas das letras, o logo e determina o banco de palavras, passando as palavras disponíveis no jogo. Neste caso, optamos por palavras com 5 letras. Também são determinadas as palavras selecionadas para o jogo. Dentro da função ChamaTelaJogo, também temos também todas as funções responsáveis por imprimir os caracteres e fazer a movimentação deles na tela. Maiores detalhes podem ser consultados no arquivo .h que acompanha o jogo na pasta com as implementações.

```
72  int qtd_letras2;
73      sf::Font font;
74      font.loadFromFile("fonte.TTF"); //lê a fonte
75      sf::Font font2;
76      font2.loadFromFile("fonte2.TTF");
77      Pilha<char> pilha0(TAMY);
78      Pilha<char> pilha1(TAMY);
79      Pilha<char> pilha2(TAMY);
80      Pilha<char> pilha3(TAMY);
81      Pilha<char> pilha4(TAMY);
82      Pilha<char> pilha5(TAMY);
83      sf::Texture texture[QTD_LETRAS];
84      texture[0].loadFromFile("letraa.png"); //lê as texturas
85      texture[1].loadFromFile("letrac.png"); //lê as texturas
86      texture[2].loadFromFile("letrae.png"); //lê as texturas
87      texture[3].loadFromFile("letrag.png"); //lê as texturas
88      texture[4].loadFromFile("letral.png"); //lê as texturas
89      texture[5].loadFromFile("letrao.png"); //lê as texturas
90      texture[6].loadFromFile("letrap.png"); //lê as texturas
91      texture[7].loadFromFile("letrar.png"); //lê as texturas
92      texture[8].loadFromFile("letras.png"); //lê as texturas
93      texture[9].loadFromFile("letrat.png"); //lê as texturas
94      texture[10].loadFromFile("letraespecial.png"); //lê as texturas
95      texture[11].loadFromFile("letrat.png"); //lê as texturas
96      texture[12].loadFromFile("letrap.png"); //lê as texturas
97      texture[13].loadFromFile("letraa.png"); //lê as texturas
98      texture[14].loadFromFile("letrao.png"); //lê as texturas
99      texture[15].loadFromFile("letrar.png"); //lê as texturas
100
101     float tempo = 0.5;
102     //logo
103     sf::Texture logo;
104     logo.loadFromFile("logopng.png"); //lê o logo
105     sf::Sprite ologo;
106     ologo.setTexture(logo); //textura do primeiro
107     ologo.setPosition(350, 10); //posicao do primeiro
108                                 //ologo.setScale(sf::Vector2f(1, 1)); //escala do primeiro
```

```
109     //Do Letris 1
110     string bancoPalavras[PALAVRASDISPONIVEIS] = { "porta","pasto","placa","posto","prego","peste","porco","parto","preto", "prato" }; //banco de palavras
111     string selecionadas[PALAVRASSELECIONADAS]; //retorna as palavras selecionadas dentro do banco de palavras
```

As pilhas são criadas, todos com o tamanho fixo Tamy (tamanho de Y), algumas variáveis de apoio são declaradas, a matriz do jogo é declarada e inicializada.

Na função abaixo, temos a função palavrasSelecionadas(), que recebe as palavras do banco de palavras, e através de uma função rand(), ela determina quais as palavras serão selecionadas para o jogo. Também temos a função SorteiaLetras() que funciona da mesma forma que a palavrasSelecionadas(), usando uma função rand para determinar quando cai uma letra ou quando cai o caracter *.

```
51    //Palavras Selecionadas
52    void palavrasSelecionadas(string bancoPalavras[PALAVRASDISPONIVEIS], string selecionadas[PALAVRASSELECIONADAS])
53    {
54        int contador = 0;
55        int randomNumber;
56        int randomPalavras[PALAVRASSELECIONADAS] = { 0,0,0 };
57        bool verifica = 0;
58        time_t seconds;
59        time(&seconds);
60        srand((unsigned int)seconds);
61
62        randomPalavras[0] = rand() % 10;
63
64        while (verifica != 1) {
65            randomNumber = rand() % 10;
66            if (contador == 0 && (randomNumber != randomPalavras[0])) {
67                randomPalavras[1] = randomNumber;
68                contador = 1;
69                randomNumber = rand() % 10;
70            }
71            if (contador == 1 && (randomNumber != randomPalavras[0] && randomNumber != randomPalavras[1])) {
72                randomPalavras[2] = randomNumber;
73                verifica = 1;
74            }
75
76        }
77        selecionadas[0] = bancoPalavras[randomPalavras[0]];
78        selecionadas[1] = bancoPalavras[randomPalavras[1]];
79        selecionadas[2] = bancoPalavras[randomPalavras[2]];
80    }
81
82    //Sorteia letras
83    char sorteiaLetras(char letras[QTD_LETRAS]) {
84        int a = rand() % QTD_LETRAS;
85        if (rand() % 20 < PALAVRASSELECIONADAS)
86            return '*';
87        else
88            return letras[a];
89    }
```

CONCLUSÕES

Neste trabalho aprendemos a implementação de Tipos Abstratos de Dados, especificamente pilhas e filas. Aprendemos a manipular algumas funções de interface gráfica através do SFML e conseguimos a partir disso implementar um jogo no estilo Tetris, só que ao invés de preencher linhas, o que faz-se é formar palavras.