Documentação

3R: Terra

Felipe Sampaio de Souza
Gabriel de Paiva Oliveira
Miguel de Souza Tosta
Pedro Henrique Migliatti

São Carlos - 2016

Sumário

1. Autores

a. Identificação

- Felipe Sampaio de Souza
  - RA: 619523
  - E-mail: sampaio.motion@gmail.com

- Gabriel de Paiva Oliveira
  - RA;
  - E-mail: gabriel.paiv.oliv@hotmail.com

- Miguel de Souza Tosta
  - RA: 619698
  - E-mail: miguel_st02@hotmail.com

- Pedro Henrique Migliatti
  - RA: 619744
  - E-mail: pedro.migliatti@gmail.com

b. Participação

A divisão de tarefas para a produção do jogo inicialmente foi que os integrantes Felipe Sampaio de Souza e Pedro Henrique Migliatti ficariam responsáveis pela desenvolvimento gráfico e os integrantes Gabriel de Paiva Oliveira e Miguel de Souza Tosta responsáveis pela implementação e integração da estrutura de dados utilizada (Fila).

A implementação e documentação do jogo foi realizada em conjunto por todos os membros do grupo.

2. Jogabilidade e Funcionamento do Jogo

a. Introdução

A motivação para esse jogo surgiu da ideia de um programa que fosse, além de entretenimento para o usuário, educacionalmente útil. Para isso pensamos em algo que ajuda os usuários a memorizarem as cores das lixeiras referentes a cada tipo de lixo reciclável, algo simples mas que muitas pessoas ainda tem dificuldade.

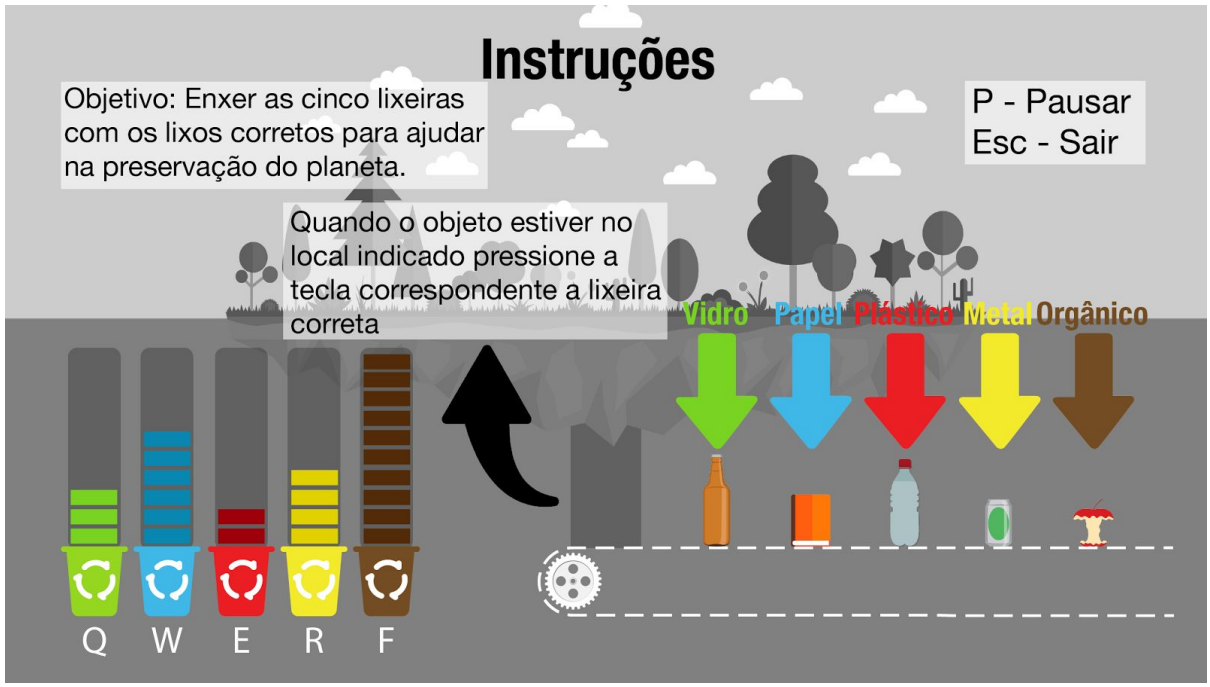

Imagem 1. Tela de instruções do jogo.

b. Controles

Todos os comandos do jogo são via teclado, na tela de inicio do jogo qualquer tecla pode ser pressionada para dar inicio a partida. Com a partida em andamento temos os seguintes comandos (que estão graficamente representados na Imagem 1):

Tecla	Função
Q	Captura lixos do tipo vidro que são destinados a lixeira verde.
W	Captura lixos do tipo papel que são destinados a lixeira azul.
E	Captura lixos do tipo plastico que são destinados a lixeira vermelha.
R	Captura lixos do tipo metal que são destinados a lixeira amarela.
F	Captura lixos do tipo orgânico que são destinados a lixeira marrom.
P	Pausa a partida.
Esc	Fecha o jogo.

c. Dinâmica do Jogo

Inicialmente o cenário do jogo começa todo em tons de cinza e sem nenhum animal e conforme o jogador for completando os marcadores ele sobe de nível e com isso o cenário vai ficando colorido e são acrescentados animais (cada marcador completo significa um nível a mais). Para desbloquear o cenário todo e vencer no jogo o jogador deve completar todos os marcadores, passando assim por todos os níveis.

Cada vez que o jogador aperta uma tecla correspondente a lixeira errada ou quando não houver nenhum lixo na região demarcada seu nível é decrementado, o marcador da lixeira correspondente aquela tecla é zerado e um pedaço do cenário volta a ser cinza, caso o cenário já esteja todo cinza apenas o marcador é zerado.

Caso o jogador deixe algum lixo passar da área demarcada esse lixo sumirá e o nível do jogador é decrementado, o marcador da lixeira correspondente a esse lixo é zerado e um pedaço do cenário volta a ser cinza, caso o cenário já esteja todo cinza apenas o marcador é zerado.

Os lixos são apresentados para o jogador em formato de fila em cima de uma esteira que os levam até a área de captura. A cada nível que o jogador sobe a velocidade da esteira aumenta e a dificuldade do jogo também, consequentemente.

Ao completar todos os níveis o jogador irá para a tela final onde ele recebe os parabéns e pode ver o tempo que demorou para finalizar a partida.

Abaixo se encontram algumas imagens do jogo em execução.

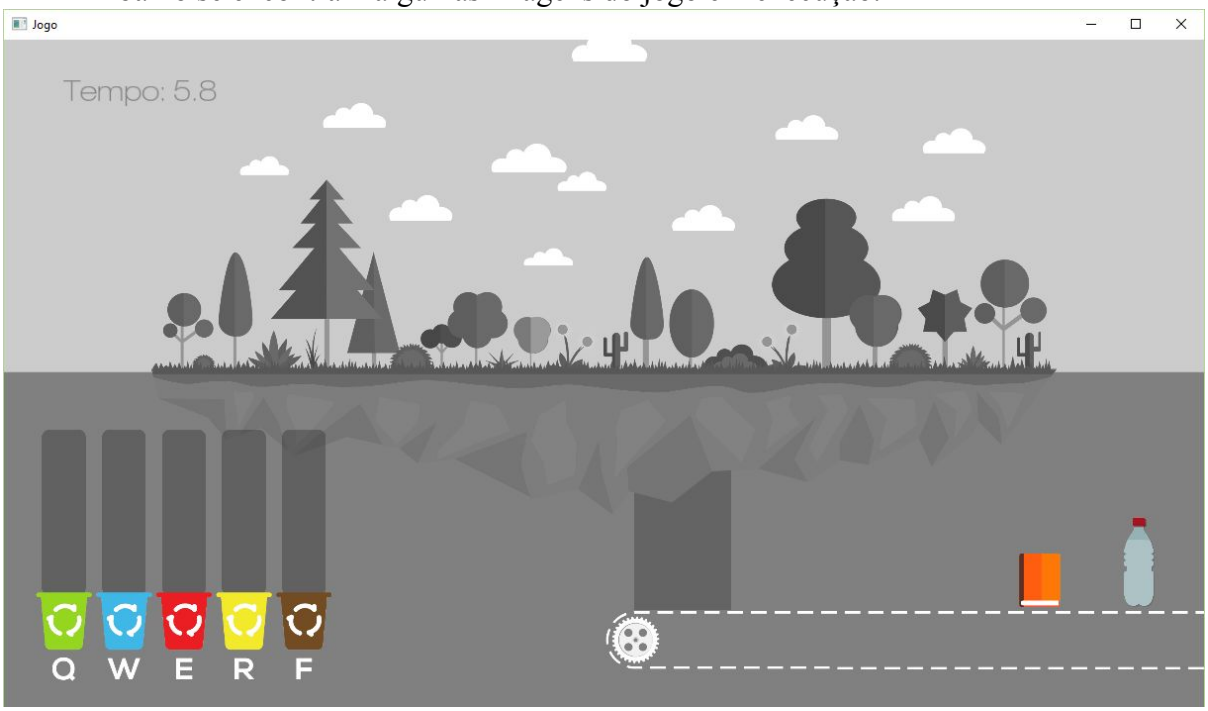

Imagem 2. Inicio da partida.

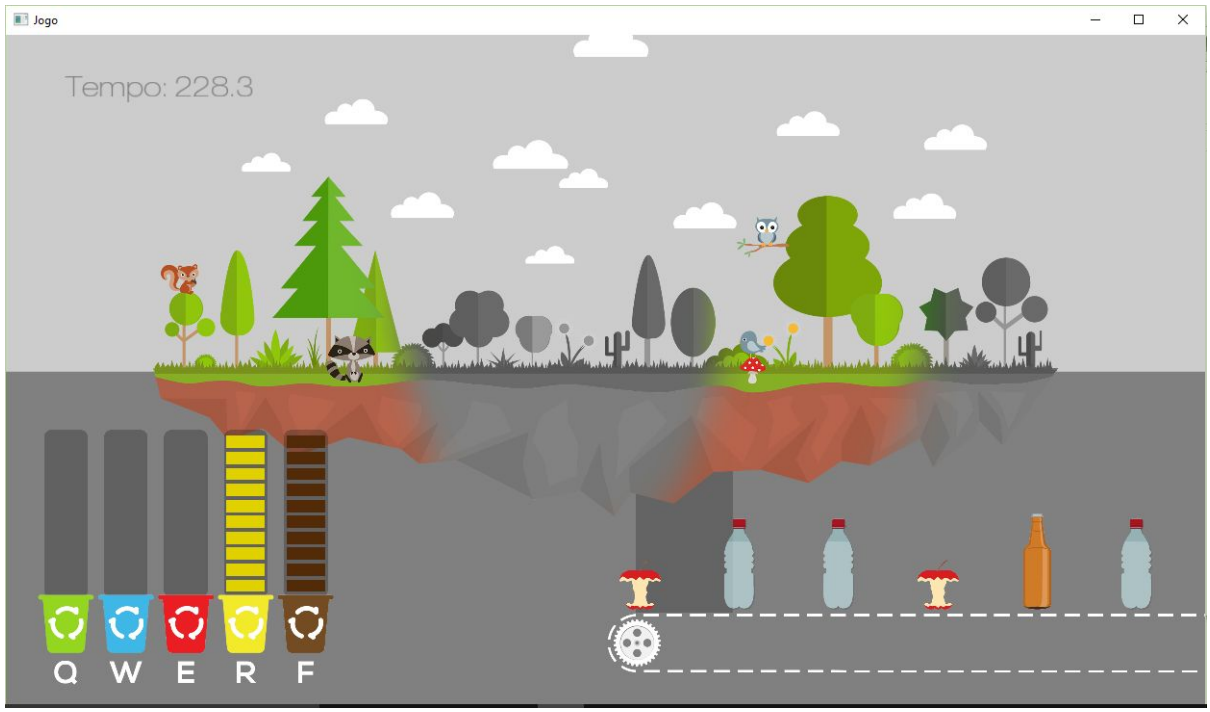

Imagem 3. Jogo no segundo nível.

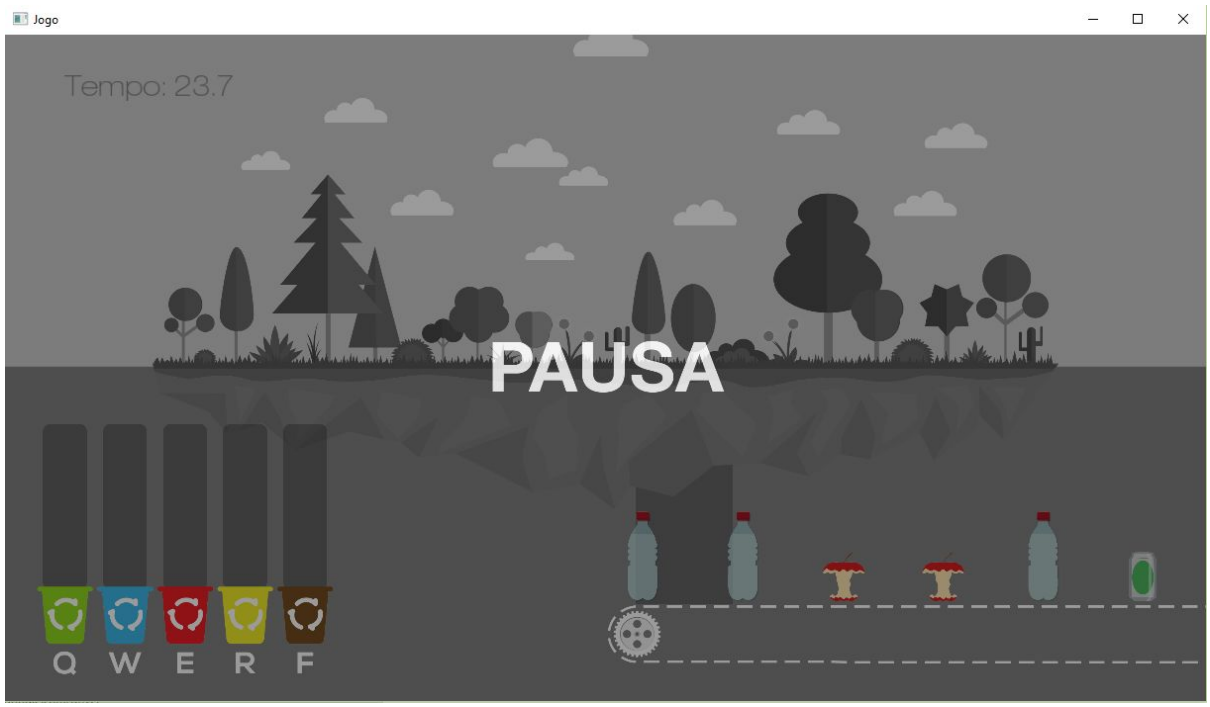

Imagem 4. Jogo pausado.

Imagem 5. Tela de vitória

d. Implementação e ferramentas

Para a execução da parte gráfica do jogo, foi utilizada a biblioteca *SFML (Simple Fast Media Library)*, uma biblioteca multiplataforma focada na implementação em C++, por esse motivo, a linguagem utilizada para implementação do projeto foi C++.

Observação: Para que o executável funcione, ele necessita estar na mesma pasta que os arquivos ".dll" da biblioteca *SFML*.

3. Estrutura: Fila

a. Introdução

Para a execução da mecânica desse jogo, foi escolhido o uso de uma fila como a estrutura de dados. Para maior portabilidade, a mesma foi implementada como um template, de tal forma que o código consiga ser utilizado para outros fins, além desse jogo.

b. Implementação Fila

```
1   #ifndef FILA_H
2   #define FILA_H
3
4   #include "Lixo.h"
5
6
7   template <class T>
8
9   class Fila
10  {
11  public:
12
13      Fila(int tamanho = 10);
14      ~Fila();
15      bool vazia();
16      bool cheia();
17      bool insere(T x);
18      bool retira(T &x);
19
20  protected:
21      T* elementos;
22      int inicio;
23      int ultimo;
24      int NElementos;
25      int tamanho;
26  };
27
28  #endif // FILA_H
```

Imagem 6. Código do arquivo Fila.h

Fila(int tamanho) - Construtor de Fila que recebe um valor inteiro como argumento, indicando o tamanho da fila que será criada. Note que a fila é alocada dinamicamente, sendo necessário desalocar a memória no destrutor.

```
template <typename T>
Fila<T>::Fila(int tamanho)
{
    this->inicio = 0;
    this->ultimo = 0;
    this->NElementos = 0;

    if (tamanho < 1)
    {
        this->tamanho = TAMDEF;
    }
    else
    {
        this->tamanho = tamanho;
    }

    this->elementos = new T[this->tamanho];
}
```

~Fila() - Desaloca memória alocada em tempo de execução no construtor.

```
template <typename T>
Fila<T>::~Fila()
{
    delete []elementos;
}
```

vazia() - Verifica se a fila está vazia (não possui nenhum elemento), retornando **true** caso esteja vazia e **false** caso contrário.

```
template <typename T>
bool Fila<T>::vazia()
{
    if (this->NElementos == 0)
        return true;
    else
        return false;
}
```

cheia() - Verifica se a fila está cheia (atingiu tamanho máximo), retornando **true** caso esteja cheia e **false** caso contrário.

```
template <typename T>
bool Fila<T>::cheia()
{
    if (this->NElementos == this->tamanho)
        return true;
    else
        return false;
}
```

insere() - Verifica se a fila está cheia, caso não esteja, um novo elemento será introduzido nela e retornará um valor booleano **true** indicando que a operação foi efetuada com sucesso, caso contrário, retornará um valor **false** indicando que não foi possível inserir o elemento.

```
template <typename T>
bool Fila<T>::insere(T x)
{
    if (!this->cheia())
    {
        this->elementos[this->ultimo] = x;
        if (this->ultimo == this->tamanho - 1)
            this->ultimo = 0;
        else
            this->ultimo++;

        this->NElementos++;

        return true;
    }
    else
        return false;

}
```

retira() - Verifica se a fila está vazia, caso não esteja, o primeiro elemento será removido dela e retornará um valor booleano **true** indicando que a operação foi efetuada com sucesso, caso contrário, retornará um valor **false** indicando que não foi possível remover o elemento.

```
template <typename T>
bool Fila<T>::retira(T &x)
{
    if (!this->vazia())
    {
        x = this->elementos[this->inicio];

        if(this->inicio == this->tamanho - 1)
            this->inicio = 0;
        else
            this->inicio++;

        this->NElementos--;

        return true;
    }
    else
        return false;
}
```

c. FilaDeLixo

```
1   #ifndef FILADELIXO_H
2   #define FILADELIXO_H
3
4   #include "Fila.h"
5
6   class FilaDeLixo : public Fila<Lixo>
7   {
8   public:
9       FilaDeLixo(int tamanho);
10      ~FilaDeLixo();
11      bool pegaOPrimeiro(Lixo& x);
12      bool pegaOUltimo(Lixo& x);
13      void move(float y);
14  };
15
16  #endif // FILADELIXO_H
```

Para o uso do template Fila citado anteriormente, foi criada uma classe chamada FilaDeLixo que herda de Fila e armazena objetos do tipo lixo.

FilaDeLixo(int tamanho) - Construtor vazio que envia o tamanho da fila para a classe base.

```
FilaDeLixo::FilaDeLixo(int tamanho) : Fila(tamanho)
{

}
```

~FilaDeLixo() - Destrutor vazio

```
FilaDeLixo::~FilaDeLixo()
{

}
```

pegaOPrimeiro(Lixo& x) - Verifica se a fila está vazia. Caso não esteja, x recebe o primeiro elemento da fila e retorna um booleano com o valor **true**, caso contrário, retorna um booleano com o valor **false**.

```
bool FilaDeLixo::pegaOPrimeiro(Lixo& x)
{
    if(this->vazia())
        return false;
    else
    {
        x = this->elementos[this->inicio];
        return true;
    }

}
```

pegaOUltimo(Lixo& x) - Pega o último elemento inserido na fila caso ela não esteja vazia.

```
bool FilaDeLixo::pegaOUltimo(Lixo& x)
{
    if(this->vazia())
        return false;
    else
    {
        if(this->ultimo == 0)
            x = this->elementos[this->tamanho-1];
        else
            x = this->elementos[this->ultimo-1];
        return true;
    }

}
```

move(float y) - Move todos os objetos contidos na fila.

```
void FilaDeLixo::move(float y)
{
    int pos = this->inicio;
    int qtd = 0;

    sf::Vector2f movement(1.f, 0.f);

    while(qtd < this->tamanho)
    {
        this->elementos[pos].move(movement * (-y));

        if (pos == this->tamanho - 1)
            pos = 0;
        else
            pos++;

        qtd++;
    }
}
```

4. **Diagrama da Arquitetura do Jogo**
    a. **Escolha da estrutura**

A

b.

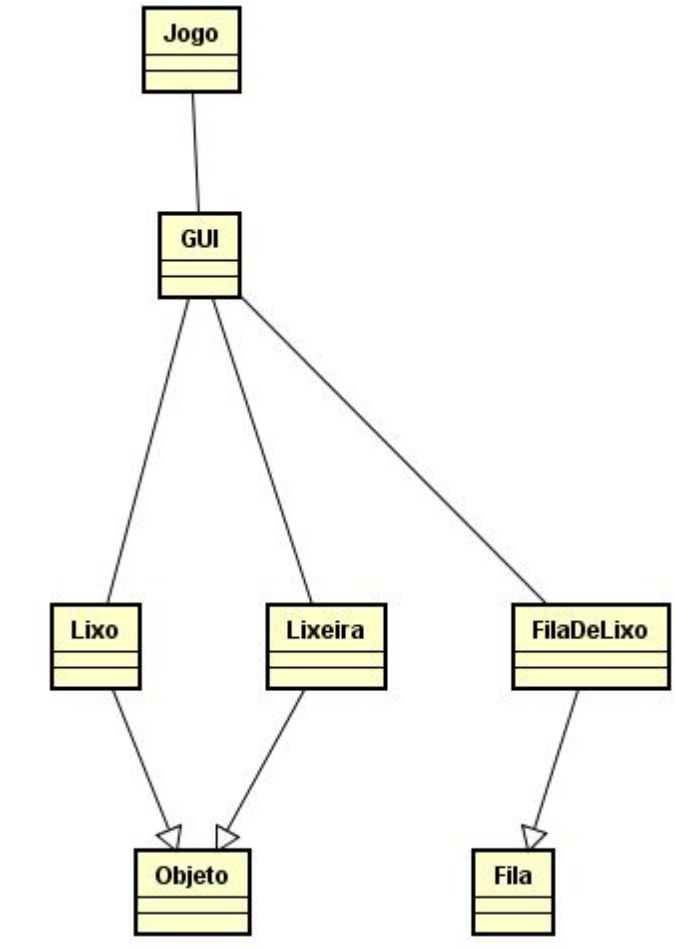

Imagem 7. Diagrama de classes.

5. Conclusão

Como esse foi o primeiro trabalho dessa disciplina, o primeiro trabalho desse grupo junto e a primeira vez que alguns de nós mexeram com uma biblioteca gráfica, houveram algumas dificuldades que aos poucos e com muita conversa foram superadas com o passar do tempo. Entre essas dificuldades a que mais demandou atenção foi a organização e a divisão de trabalho, principalmente pelo tamanho do projeto.

Apesar das dificuldades, foi um projeto em que todos os membros se empenharam muito e tiveram muita satisfação em participar. Além de poder aplicar na prática as estruturas de dados que aprendemos na matéria, a experiência e o aprendizado ganho com o trabalho foram muito grandes.

Aprender a utilizar a biblioteca gráfica, poder discutir métodos e lógicas para implementar o jogo e solucionar os problemas na hora de programar foram as partes mais interessantes do projeto, pois foram nessas horas que aprendemos mais.

Por fim, o mais gratificante para todos os participantes desse projeto foi poder vê-lo funcionando, algo que nós apenas imaginávamos e de inicio não tínhamos muita certeza de como implementar.

6. Referências

- Site da Biblioteca Gráfica Utilizada (SFML): http://www.sfml-dev.org/
- Livro: SFML Game Development (Haller, J.; Hansson, H. V.; Moreira, A.)
- Tutorial SFML: http://www.sfml-dev.org/tutorials/2.3/
- Softwares: Code::Blocks 16.01, pacote Adobe CS6
- Imagens: www.freepik.com/
- Áudios: www.freesound.org
- Música: www.incompetech.com/music/