Documentação

Robert e o Mundo das Frutas

Jhonata Nícollas Carvalho Querobim - 727342
Jonathan Gouvea da Silva - 727343
Paulo Henrique Dal Bello - 727351

Prof. Dr. Roberto Ferrari

São Carlos - 2017

Sumário

1. Apresentação

1.1. Estrutura

Pilha

1.2. Ferramenta

SFML (Simple Fast Media Library)

1.3. Nome do Jogo

Robert e o Mundo das Frutas

1.4. Tela Inicial

Figura 1.4.1: Tela Inicial

2. Autores

2.1. Identificação

- Jhonata Nícollas Carvalho Querobim
  - R.A.: 727342
  - E-mail: jhonataquerobim@gmail.com
- Jonathan Gouvea da Silva
  - R.A.: 727342
  - E-mail: jonathangouveasilva@gmail.com
- Paulo Henrique Dal Bello
  - R.A.:727342
  - E-mail: paulohdalbello@gmail.com

2.2. Participação

Inicialmente, todos os integrantes do grupo participaram ativamente no planejamento inicial do jogo e, durante todo o proceso, nas tomadas de decisão. A divisão se deu da seguinte maneira: o integrante Jhonata Querobim ficou responsável pela criação e arquivamento dos mapas/fases, o integrante Jonathan Gouvea teve como função desenvolver a interface gráfica e o integrante Paulo Dal Bello desenvolveu os efeitos sonoros e implementação da estrutura (pilha) . A implementação do jogo em si foi realizada em conjunto pelos integrantes Jonathan Gouvea e Paulo Dal Bello. A documentação foi redigida pelo integrante Jhonata Querobim.

Apesar de dividir os processos, todos os integrantes auxiliaram de alguma maneira em todas as tarefas do grupo.

3. Jogabilidade e Funcionamento

3.1. Introdução

Inicialmente a aplicação apresentará uma breve introdução da história do jogo com textos e ilustrações, uma maneira de aprofundar a imersão do jogador e otimizar sua experiência. Após isso, o jogo será iniciado automaticamente.

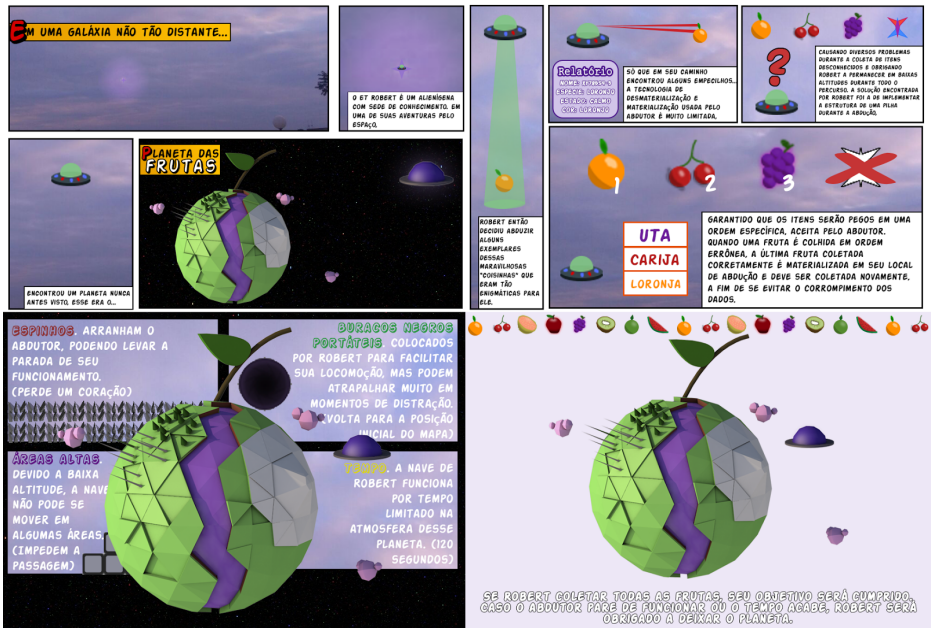

Figura 3.1.1: História Ilustrada

A função do jogador é ajudar o extraterrestre Robert a coletar as frutas na ordem apresentada para fins de estudo. O disco voador é controlado através das setas do teclado e o jogo possui diferentes mapas com frutas e obstáculos variados.

3.2. Regras

- As frutas deverão ser coletadas na ordem apresentada;
- A cada fruta coletada o jogador receberá UM ponto;
- Se a fruta abduzida não for compatível com a sequência, a última fruta correta será retirada da pilha, recolocada no mesmo lugar que fora abduzida e o jogador perderá DOIS pontos;
    - Se não houver nenhuma fruta no topo da pilha, o jogador perderá UMA vida;

- Se o personagem passar por um espinho, perderá UMA vida;
- Se o personagem passar por um buraco negro, voltará para o ponto de partida, mantendo as frutas e o tempo restante;
- Se todas as frutas forem coletadas na ordem correta, o jogador passará para o nível seguinte;
- A cada fase nova, o jogador receberá UMA vida;
- Se as vidas acabarem, o jogador perderá;
- Se o tempo acabar, o jogador perderá;
- Se o jogador passar todas as fases, então ele vencerá.

3.3. Mecânica

Após iniciar o jogo, o personagem estará localizado no canto superior esquerdo e a sequência no qual as frutas deverão ser coletadas aparecerá no placar. O jogador terá DEZ segundos para memorizar a ordem e não poderá se mover nesse tempo.

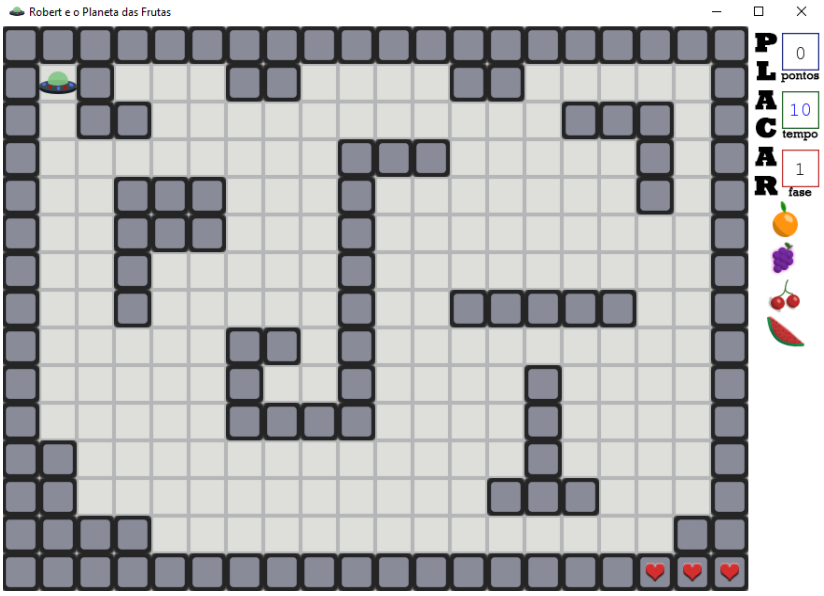

Figura 3.3.1: Tela Inicial - Mapa

Na imagem do placar abaixo, é possível reparar na sequência a ser seguida e o tempo para memorizá-la

Figura 3.3.2: Tela Inicial - Placar

Posteriormente, o jogador poderá utilizar as setas do teclado para movimentar o personagem e coletar os itens. A ordem das frutas funcionam como uma pilha, ou seja, o que for coletado será colocado no topo, assim como o que será retirado.

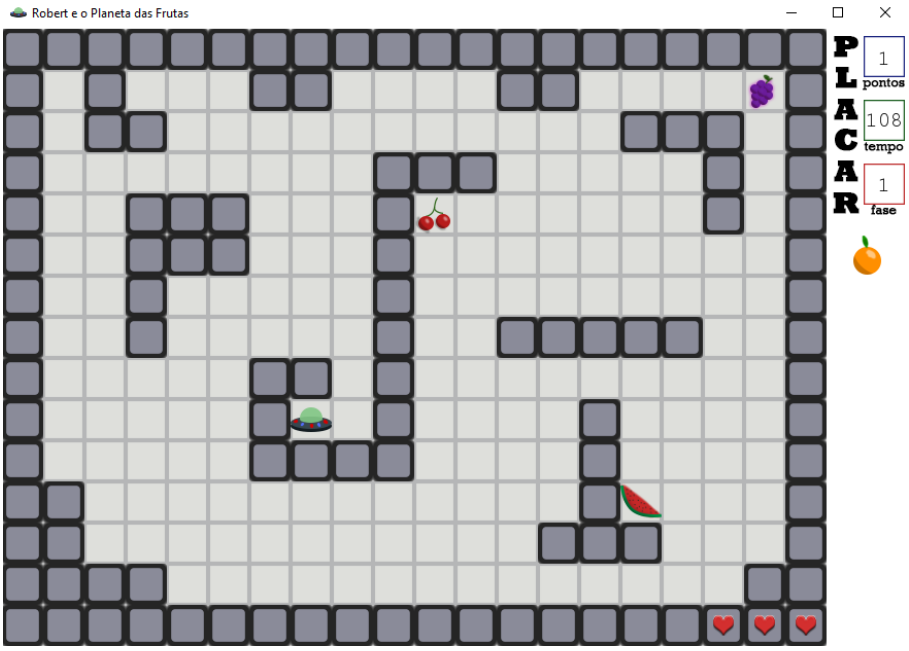

Figura 3.3.3: Jogo em Andamento - Loronjo coletado

Figura 3.3.4: Fase 1 Completa

A dificuldade das fases é distribuída de maneira crescente. A partir da segunda fase, os mapas apresentam espinhos e na terceira aparecem buracos negros. A quantidade de frutas também variam de maneira crescente em cada fase. Sempre que o personagem passa por um espinho o jogador perde UMA vida e os buracos negros redirecionam o personagem ao ponto de partida, porém mantém as frutas e o tempo restante.

Figura 3.3.5: Espinhos

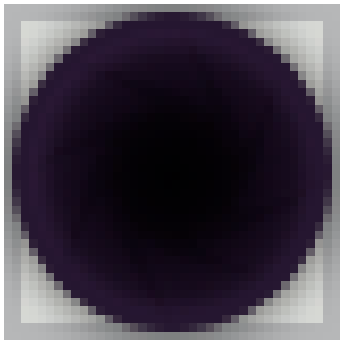

Figura 3.3.6: Buraco Negro

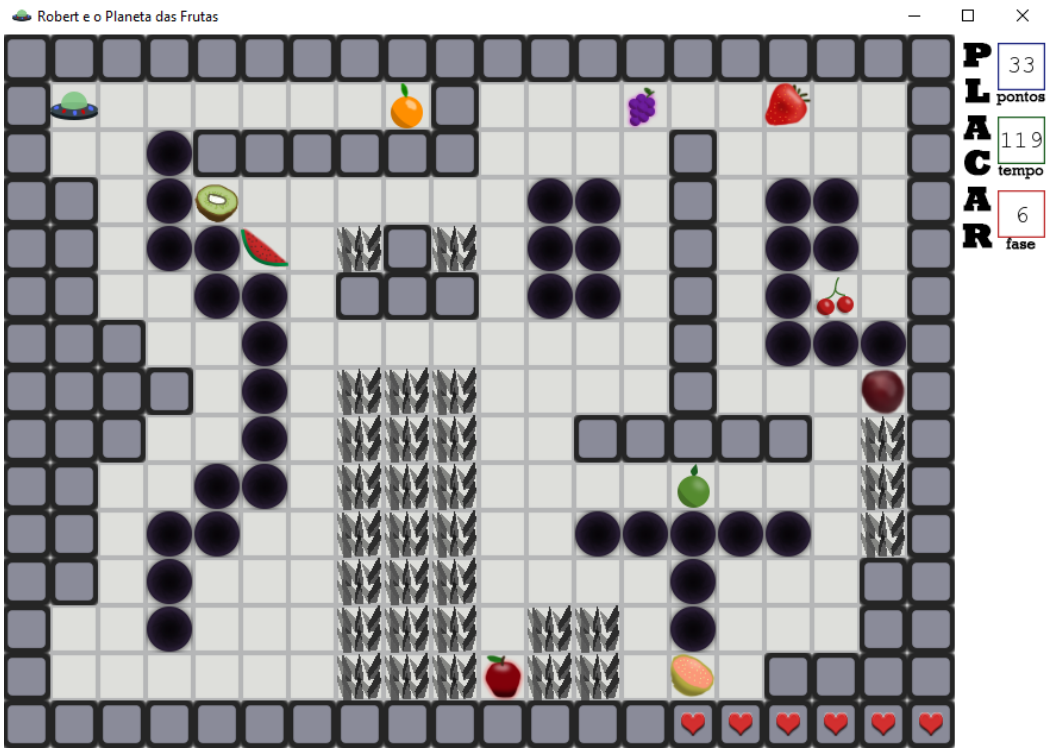

Figura 3.3.7: Representação de Dificuldade - Fase 6

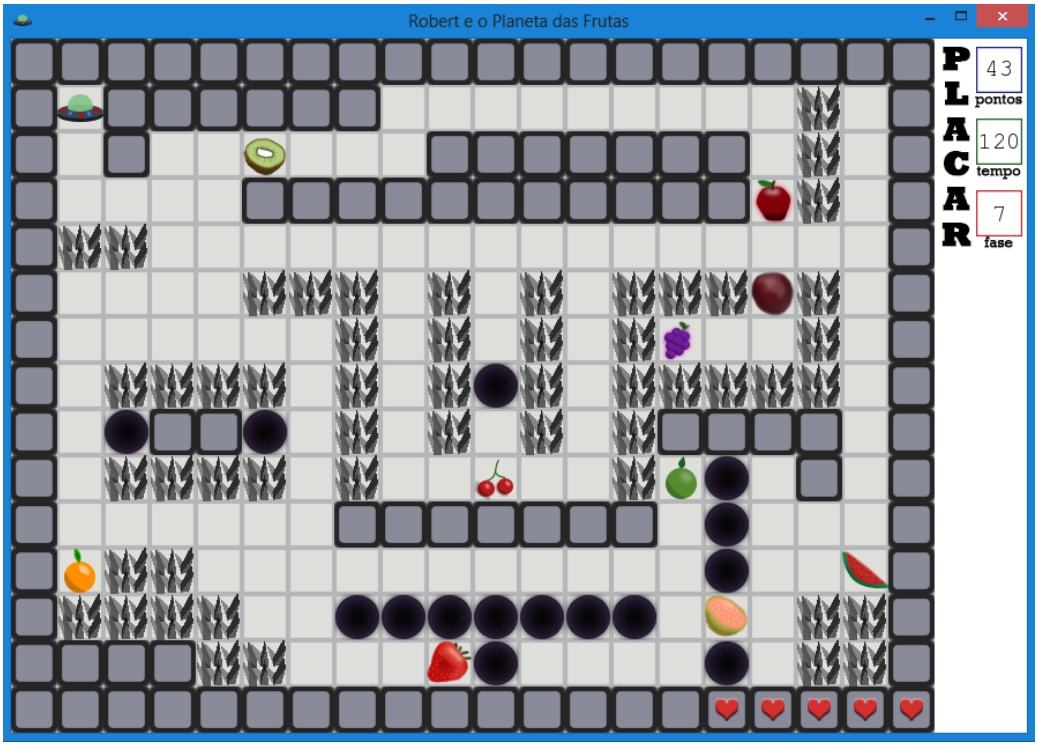

Figura 3.3.8: Representação de Dificuldade - Fase 7

O jogo acaba quando o jogador coletar todas as frutas de todas as fases, acabar o tempo ou perder todas as vidas. A vitória é dada pela primeira alternativa, enquanto as outras duas representam o Game Over.

Figura 3.3.9: Tela de Game Over

Figura 3.3.10: Tela de Vitória

4. Estrutura e Implementação

4.1. Ferramenta

A ferramenta escolhida para o desenvolvimento do jogo foi o SFML com Code::Blocks IDE. O SFML, é uma biblioteca multiplataforma voltada para a implementação em C++ e a versão utilizada foi a 2.4.9. A versão do GCC compiler do Code::Blocks foi a 4.9.2. Para a implementação do jogo foram necessárias a inclusão de algumas bibliotecas do SFML e suas dependentes:

Módulo	Dependentes	Módulo	Dependentes
sfml-graphics-s	○ sfml-window-s ○ sfml-system-s ○ opengl32 ○ freetype ○ jpeg	sfml-audio-s	○ sfml-system-s ○ openal32 ○ flac ○ vorbisenc ○ vorbisfile ○ vorbis ○ ogg
sfml-window-s	○ sfml-system-s ○ opengl32 ○ winmm ○ gdi32	sfml-system-s	○ winmm

Tabela 4.1.1: Bibliotecas e Dependências SFML

4.2. Estrutura

Para atender a proposta do jogo, a estrutura utilizada na implementação do jogo foi uma pilha. Com o intuito de garantir a portabilidade do projeto, a pilha foi implementada como uma template, assim poderia ser utilizada para diversos propósitos além desta aplicação específica.

4.3. Funções

Pilha() - Construtor Padrão que cria um objeto Pilha com o ponteiro para NULL e o topo -1, ou seja, uma pilha vazia.

```
template <class T>
Pilha<T>::Pilha()
{
    PilhaPtr = NULL;
    Topo = -1;
}
```

Figura 4.3.1: Funções da Pilha - Pilha()

~Pilha() - Destrutor padrão que desaloca do ponteiro PilhaPtr caso a pilha não esteja vazia

```
template <class T>
Pilha<T>::~Pilha()
{
    if(PilhaPtr != NULL)
        delete [] PilhaPtr;
}
```

Figura 4.3.2: Funções da Pilha - ~Pilha()

Insere() - Verifica se a Pilha está vazia. Caso afirmativo, insere o valor do parâmetro na Pilha, caso contrário, realoca o conteúdo da Pilha para uma pilha auxiliar, desalocando a memória da Pilha principal para alocar novamente com o tamanho adequado e, finalmente, retornando com os valores e inserindo o elemento passado pelo parâmetro no topo.

```
template <class T>
void Pilha<T>::Insere(const T &elemento)
{
    if(Topo != -1)
    {
        T *PtrAux;
        int i;
        PtrAux = new T[++Topo];
        for(i=0;i<Topo;i++)
            PtrAux[i] = PilhaPtr[i];
        delete [] PilhaPtr;
        PilhaPtr = new T[Topo+1];
        for(i=0;i<Topo;i++)
            PilhaPtr[i] = PtrAux[i];
        PilhaPtr[i] = elemento;
        delete [] PtrAux;
    }
    else
    {
        PilhaPtr = new T[1];
        Topo++;
        PilhaPtr[0] = elemento;
    }
}
```

Figura 4.3.4: Funções da Pilha - Insere()

Retira() - Uma função bool que verifica se a fila está vazia. Caso afirmativo, a função retorna false. Caso contrário, o elemento passado por parâmetro receberá o item do topo da pilha e esse valor será retirado da mesmo, que será realocada para ficar com o tamanho adequado.

```
template <class T>
bool Pilha<T>::Retira(T &elemento)
{
    if(Topo > 0)
    {
        T *PtrAux;
        int i;
        elemento = PilhaPtr[Topo];
        PtrAux = new T[Topo];
        for(i=0;i<Topo;i++)
            PtrAux[i] = PilhaPtr[i];
        delete [] PilhaPtr;
        PilhaPtr = new T[Topo];
        for(i=0;i<Topo;i++)
            PilhaPtr[i] = PtrAux[i];
        Topo--;
        delete [] PtrAux;
    }
    else
    {
        if(Topo == 0)
        {
            elemento = PilhaPtr[Topo];
            Topo--;
            delete [] PilhaPtr;
            PilhaPtr = NULL;
        }
        else
            return false;
    }
    return true;
}
```

Figura 4.3.5: Funções da Pilha - Retira()

EstaVazia() - Função bool que retorna true caso a Pilha esteja vazia e false caso contrário.

```
template <class T>
bool Pilha<T>::EstaVazia() const
{
    return Topo == -1;
}
```

Figura 4.3.6: Funções da Pilha - EstaVazia()

4.4. Pilha em Jogo

No jogo, existe a pilha utilizada para armazenar a sequência de frutas e a pilha que representa as frutas coletadas no decorrer do jogo. A pilha de frutas coletadas recebe um elemento novo quando uma fruta correspondente à pilha de sequência é coletada e tem um elemento retirado sempre que não estiver vazia e uma fruta coletada não equivale a sequência correta.

4.5. Funções da main

setEnderecos() - Atribui valores para um vetor “Endereco” que armazena as imagens utilizadas no projeto;

setTexturas() - Carrega as texturas de cada elemento do vetor “Endereco”;

setSprites() - Carrega os sprites de cada elemento do vetor “Endereco”;

desenhaMapa() - Carrega o sprite e desenha o mapa da fase determinada;

desenhaPersonagem() - Carrega o sprite e desenha o personagem no mapa;

adicionaFrutas() - Adiciona uma fruta coletada corretamente na pilha de frutas;

desenhaPlacar() - Imprime nos locais adequados a pontuação, tempo, e número da fase. Além disso, carrega o sprite e desenha as vidas do personagem e a pilha de frutas coletadas;

inicia() - Inicia a tela do jogo, carregando o spirte e desenhando a pilha da sequencia de frutas durante 10 segundos;

leMapa() - Lê o mapa e define as bordas como “áreas altas” (paredes);

movimenta() - Faz a movimentação do personagem, verificando a validade das frutas quando coletadas.

5. Diagrama da Arquitetura

5.1. Estrutura

De acordo com os conceitos vistos em aula, o jogo foi desenvolvido utilizando a “Solução A” de Arquitetura da Estrutura, representado pela seguinte figura:

Figura 5.1.1 - Diagrama da Arquitetura da Estrutura

5.2. Implementação

A única classe utilizada no projeto foi a “Pilha”, já que a “inteligência” foi implementada na aplicação (main), portanto, o diagrama de implementação possui somente a classe Pilha e a Main, como ilustra a imagem abaixo:

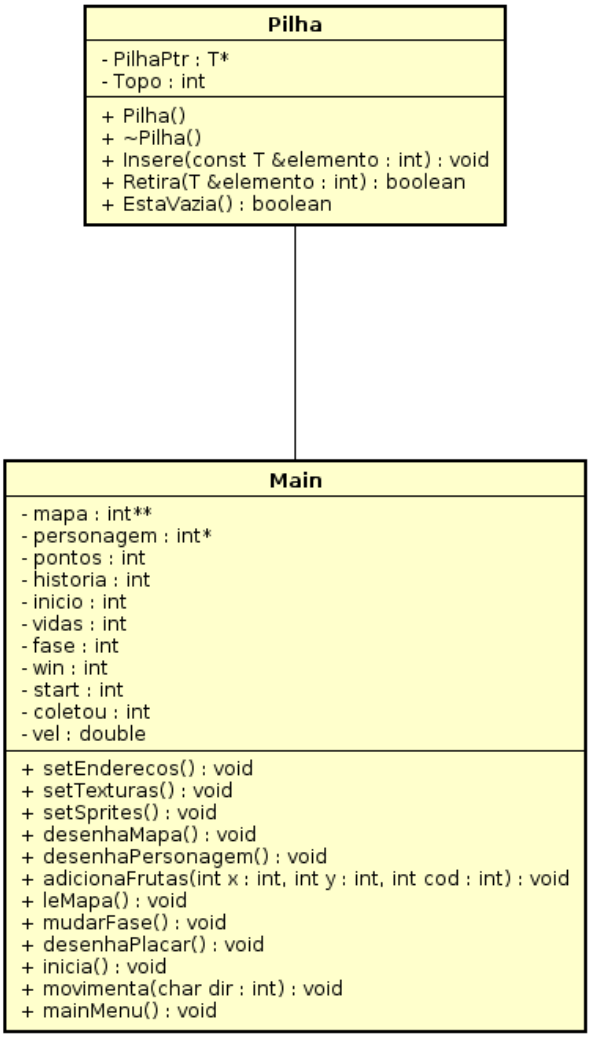

Figura 5.2.1: Diagrama de Implementação

6. Conclusão

Inicialmente, o desenvolvimento de um jogo que utiliza uma estrutura de Pilha auxiliou na fixação dos conceitos vistos em sala, tais como os Tipo Abstratos de Dados e além disso, facilita a compreensão de outras estruturas, como as Filas e Listas, uma vez que seus respectivos funcionamentos e implementações são análogas.

No decorrer do desenvolvimento do projeto, poucas dificuldades que poderiam prejudicar o andamento do jogo, sendo que os únicos problemas memoráveis se deram no início, tais como o planejamento inicial do conceito do jogo e instalação e configuração adequada da ferramenta SFML. Apesar disso, o grupo foi capaz de contornar as dificuldade e não foi prejudicado com o surgimento de qualquer problema.

No geral, a oportunidade de utilizar uma ferramenta gráfica para desenvolver um jogo com estrutura gráfica e sonora inclusas agregou conhecimento e experiência para o grupo, expandindo a lógica e implementação para novas áreas.

Por fim, a finalização do projeto é gratificante uma vez que os integrantes foram capazes de construir um jogo da maneira como foi planejado e, por conta disso, o grupo está satisfeito com o resultado final.

7. Referências

SFML. Tutorials for SFML 2.4. Disponível em: <https://www.sfml-dev.org/tutorials/2.4/>. Acesso em: 20/04/2017;

Sons:

Bensound. Disponível em: <http://www.bensound.com/>. Acesso em: 13/05/2017;

ZapSplat. Disponível em: <http://www.zapsplat.com/> Acesso em: 13/05/2017;

Imagens:

Dos Autores;

NASA. Nasa Multimedia. Disponível em: <https://www.nasa.gov/multimedia/imagegallery>. Acesso em: 16/05/2017.

8. Anexos

8.1. Lista de Figuras

8.2. Lista de Tabelas