Universidade Federal de São Carlos
Departamento de Computação
Estrutura de Dados

Documentação Trabalho 2

Renata Sarmet Smiderle Mendes, 726586, renatassmendes@hotmail.com

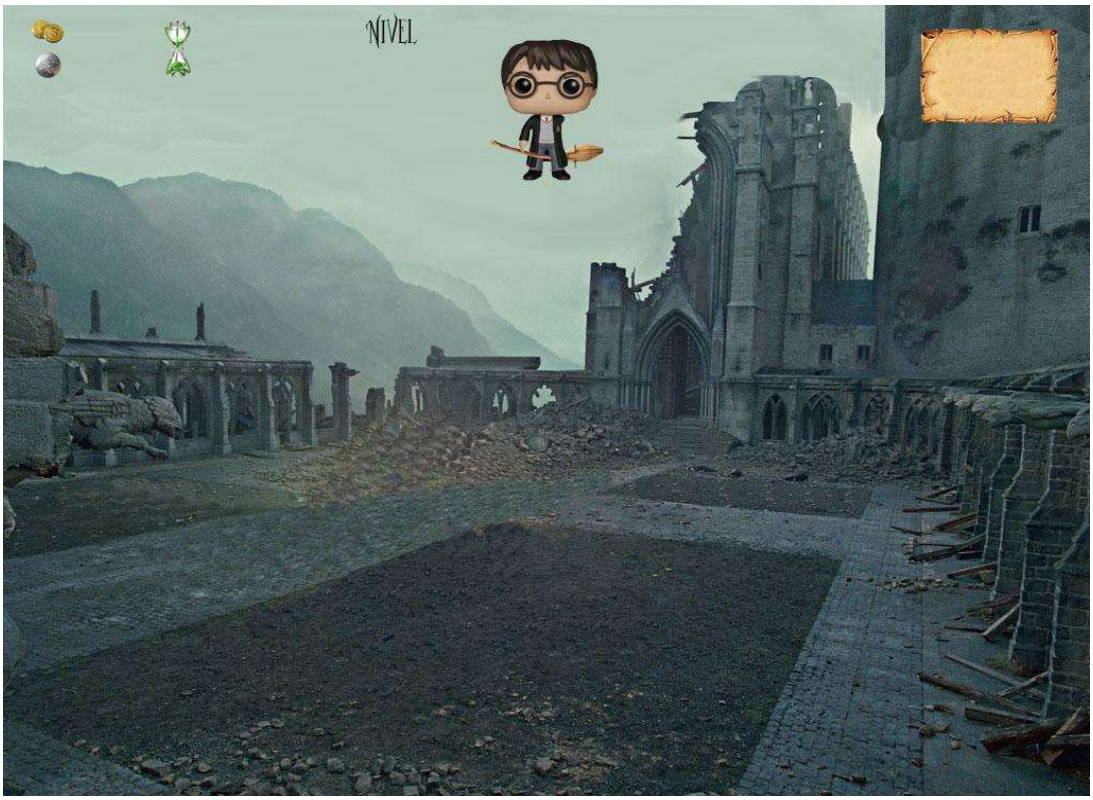
Rodrigo Pesse de Abreu, 726588, abreu.rodrigo97@gmail.com

Rafael Bastos Saito, 726580, rafaelbsaito@hotmail.com

1. Funcionamento do jogo

O jogo *Harry Potter e a Lista Sequencial* exige do usuário agilidade e reflexo. Basicamente, o jogo consiste em um personagem principal que movimenta seu braço como um pêndulo de forma constante. Atrelado a esse movimento, há a possibilidade do boneco lançar dois tipos de magias, em forma de “disparo”, para atingir algum elemento (item, horcrux, amigo ou inimigo) que esteja no plano. As magias são controladas pelo usuário, o qual designa o tipo e a hora de disparo da mesma.
Os objetivos do jogo são destruir todas as 6 horcruxes, que aparecem ao longo do jogo, e acumular dinheiro necessário para passar em cada fase. Após atingir tais objetivos, a fase final será o confronto contra o chefão (Voldemort).
Conforme o usuário termina cada partida, a quantidade de dinheiro e horcruxes é acumulada e, então, contabilizada;
O jogo é composto pelos seguintes elementos:

- **Plano:** No plano são inseridos 20 elementos, sendo eles distribuídos entre: itens, horcruxes, amigos e/ou inimigos; Em cada rodada, os itens são distribuídos de forma aleatória. As horcruxes podem ser vistas apenas uma vez em cada rodada, não necessariamente aparecendo. Ou seja, há a possibilidade de sua aparição, sendo única na fase, porém não necessariamente irá aparecer.



Plano do jogo sem elementos

- **Magias:** Há duas possibilidades de magias, sendo elas: Accio (“magia de puxar”) e Bombarda (“magia de destruir”);

Magia Accio

Magia Bombarda

- **Elementos:** Os elementos distribuídos no plano diferem-se entre: Itens, Horcruxes, amigos e inimigos; Os elementos reagem de forma diferente quando atingidos por cada magia;

- **Itens:** quando atingidos pela Accio, o usuário ganha dinheiro e o item em questão é removido; quando atingidos pela Bombarda, o usuário não ganha dinheiro e o item em questão é removido;

Itens

- **Amigos:** quando atingidos pela Accio o usuário ganha dinheiro e o amigo é removido do plano; quando atingidos pela Bombarda o usuário perde dinheiro e o amigo é removido do plano;

Amigos de Harry

- **Horcruxes:** quando atingidos pela Accio o usuário perde o jogo; quando atingidos pela Bombarda o usuário contabiliza mais uma horcrux destruída;

Lista de Horcruxes

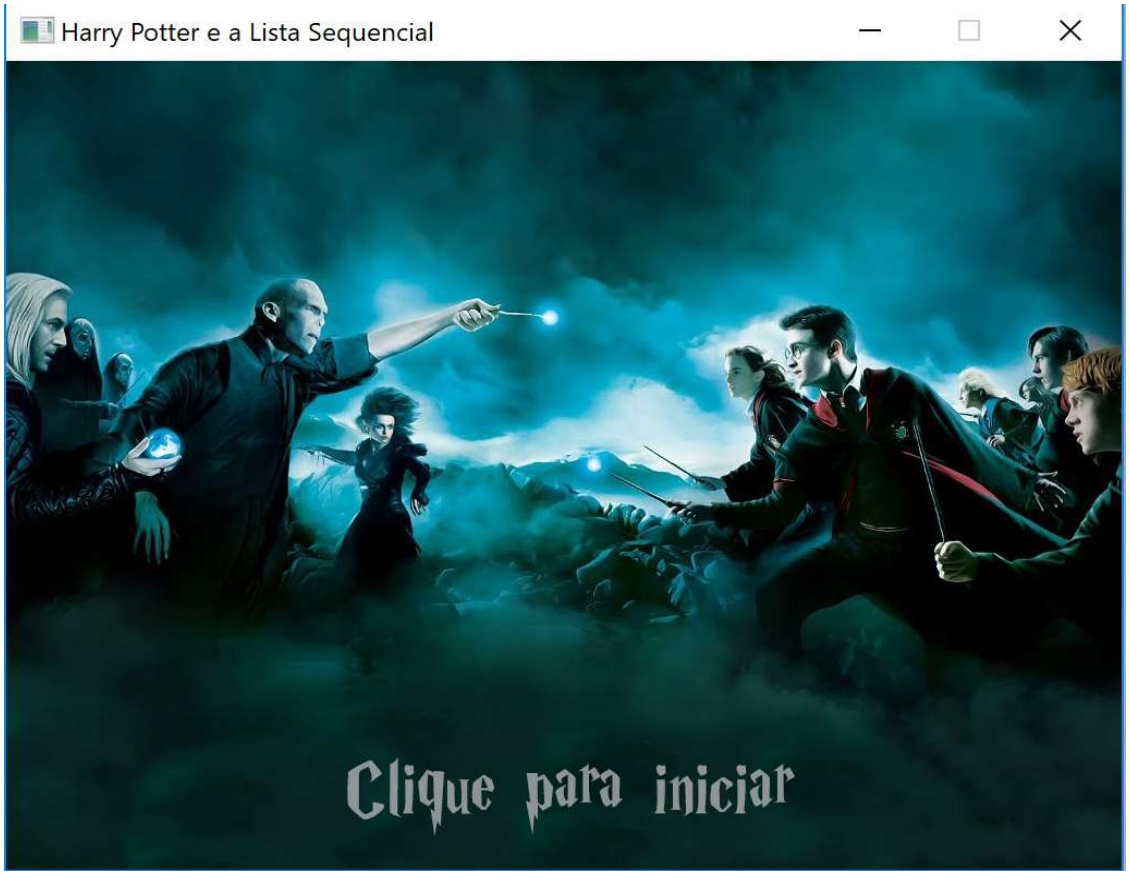
- **Inimigos:** quando atingidos pela Accio o usuário perde dinheiro e o inimigo é removido do plano; quando atingidos pela Bombarda o usuário ganha dinheiro e o inimigo é removido do plano.

Inimigos

- **Player:** O player é o direcionador do disparo da magia. Conforme o movimento da varinha, o usuário deve disparar a magia no ângulo adequado para atingir o elemento no plano; Quando o usuário clicar com a tecla "F" a magia bombarda será lançada, quando clicado na tecla "J" a magia accio será lançada;

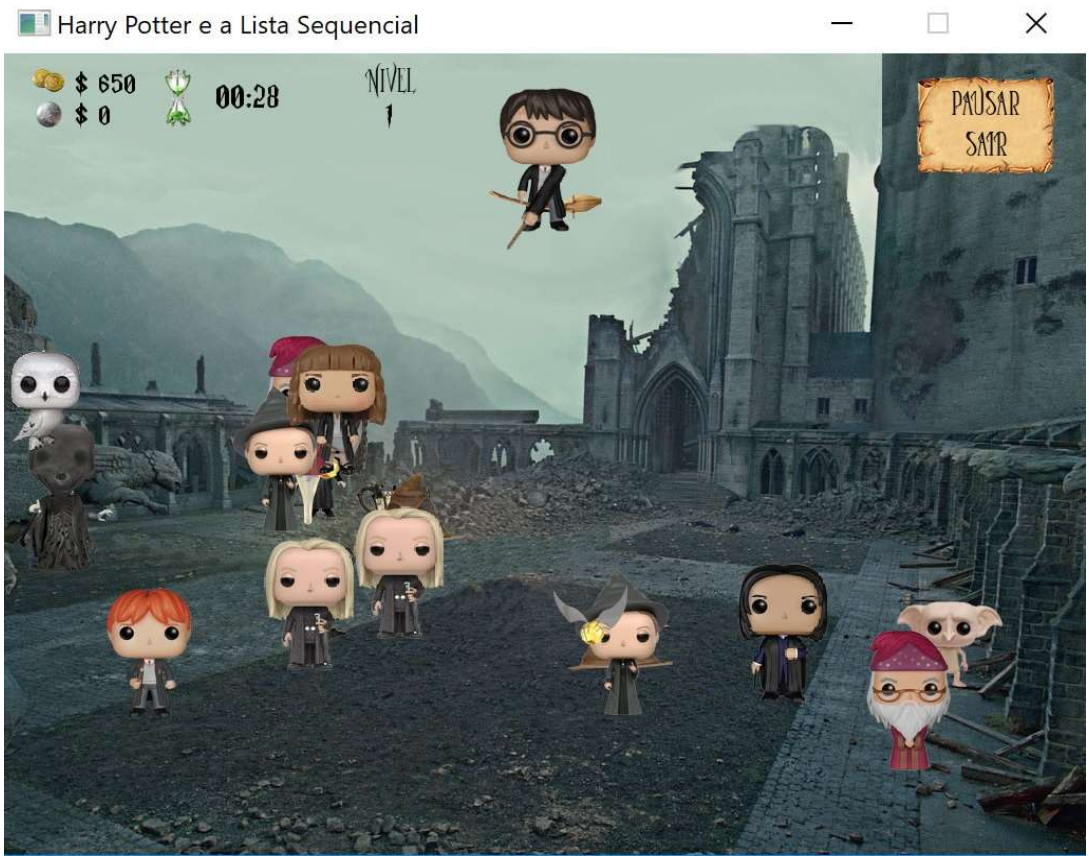
Harry Potter - representa player

A sequência de telas do jogo são:

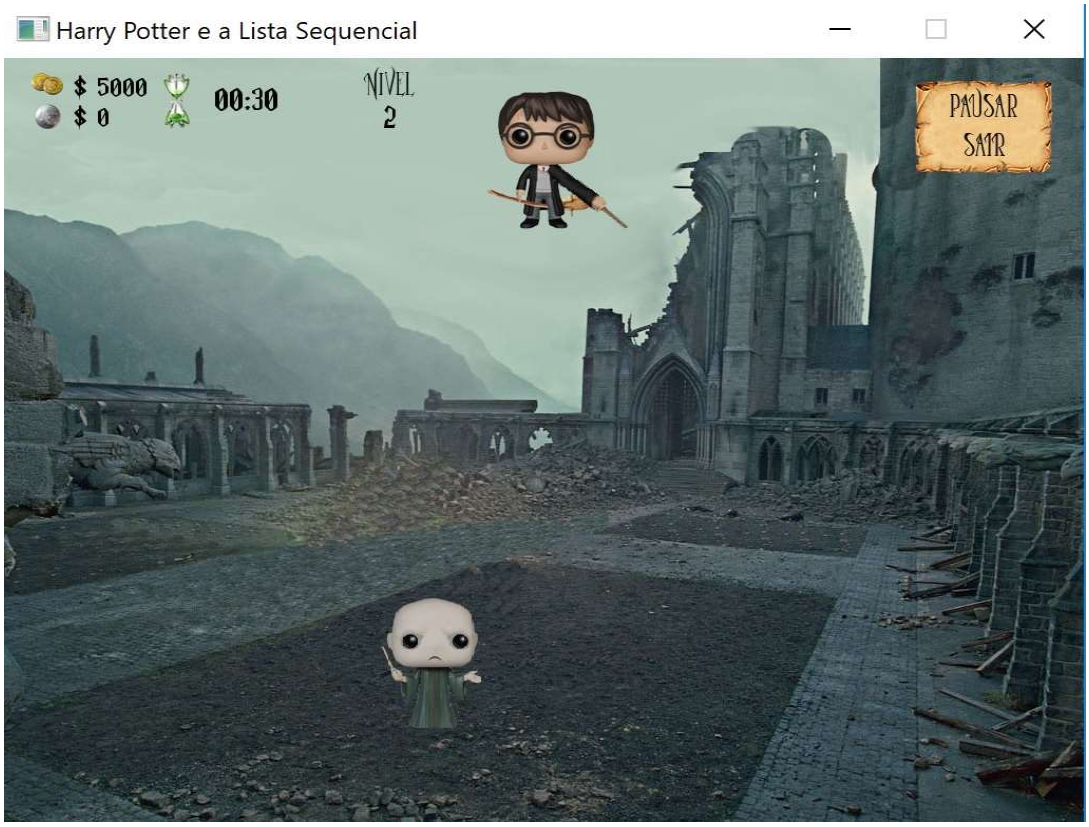


Tela Inicial

Menu



Jogo em andamento



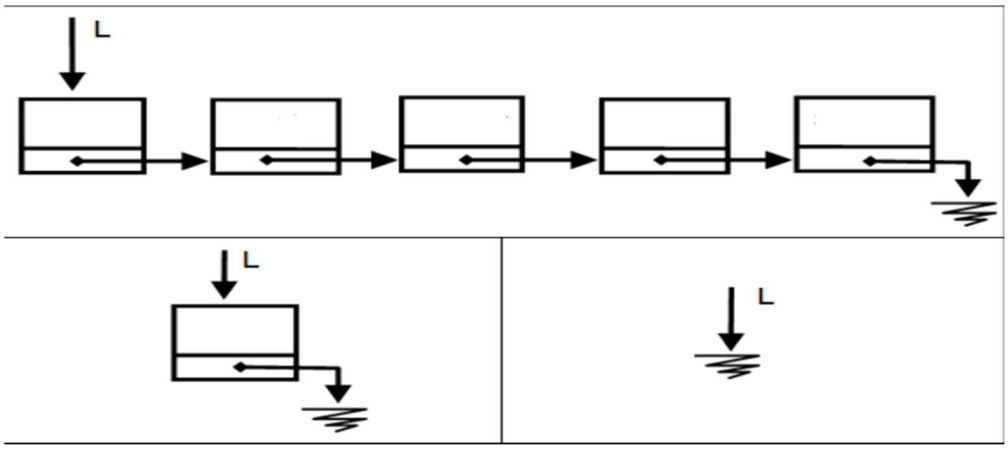
Chefão

2. Estruturação do código

Para a implementação do jogo foi utilizada a estrutura de Lista vista em sala, implementada em C++

- **Lista Simples Encadeada**

Foi utilizado a estrutura de dados Lista para diversas situações dentro do jogo.



Lista Geral

Existem, no jogo, quatro Listas: Lista geral, Lista plano, Lista de itens ganhar e Lista de itens destruídos.
A Lista geral é, como o nome diz, a mais genérica. Ela é utilizada para listar todos os elementos do jogo: inimigos, amigos, itens e horcruxes. Uma vez inserido os dados no início do jogo, não é mais alterada. Ela é responsável por listar todos os elementos possíveis para serem colocados, posteriormente, no plano. A Lista geral é do tipo ListaSimples. A Lista plano é do tipo Plano e a Lista de itens ganhar e Lista de itens destruídos é do tipo ItensGanhar. Ambos os tipos são subclasses de ListaSimples e serão melhor explicados posteriormente.

```
class ListaSimples
{
public:
    ListaSimples();
    ~ListaSimples();

    void Cria(Nodetype *x);
    bool Vazia();
    void ExibeLista();
    void Insere(Nodetype *x);
    void InsereADireita(Nodetype *x);
    void ProcuraRemove(int x, bool &DeuCerto);
    void ProcuraRemove(string x, bool &DeuCerto);
    void DeletaTudo();


    Nodetype* PegaElementoAleatorio() const; // pega um elemento na lista aleatorio
    Nodetype* PegaElementoAleatorioTodosTipoX(int x) const; // pega um elemento aleatorio do tipo X
    Nodetype* PegaElementoN(int n) const; // pega o N-esimo elemento da lista
    int QuantidadeElementos() const; // conta a quantidade de elementos da lista
    Nodetype* PegaElementoInfo(string info, bool &DeuCerto) ; // pega o elemento da lista pela Info

private:
    Nodetype *P;
    void Remove(Nodetype *Premove, bool &DeuCerto); // metodo privado pois quem deve ser chamado eh o procura remove
};
```

Código Lista Simples

- **Nó**

A classe Nodetype foi implementada para compor a lista.

```
class Nodetype
{
public:
    //CONSTRUTOR
    Nodetype();

    //DESTRUTOR
    ~Nodetype();

    //FUNCOES
    virtual void carregar(std::string nomearquivo);
    virtual void desenhar(sf::RenderWindow& renderWindow);
    virtual bool colidiu(feitico& _feitico);

    virtual float get_x() const;
    virtual float get_y() const;

    virtual void set_posicao(float x, float y);
    virtual void set_origem(float x, float y);
    virtual void move(float x, float y);

    virtual sf::Rect<float> get_bounding_rect();

    Nodetype* get_next() const;
    void set_next(Nodetype *);
    string get_info() const;
```

Classe nó - parte 1

```
40      void set_info(string);
41      int get_id() const;
42      void set_id(int);
43      void ExibeInformacoes() const;
44      void CopiaNode(Nodetype *original);
45      void set_tipo(int);
46      int get_tipo() const;
47      void set_valor(int);
48      int get_valor() const;
49
50      sf::Sprite _sprite;
51
52  private:
53      int tipo; //1 - INIMIGO, 2 - AMIGO, 3 - HORCRUX, 4 - PEGAR
54      Nodetype *next;
55      int id;
56      int valor;
57      string info;
58      bool carregou;
59      sf::Texture _imagem;
60      std::string nome_arquivo;
61  };
```

Classe nó - parte 2

- **Plano**

A classe Plano é uma subclasse de Lista Simples, portanto, ela possui todas as funcionalidades de uma lista normal, além de ter suas características específicas. Ela é a lista que aparece para o usuário durante o jogo. Como principal funcionalidade que se destaca é na inserção. Logo que se inicia a fase, são inseridos 20 itens no plano. À medida que o jogador vai acertando os itens com os feitiços, estes vão sendo removidos da lista Plano. Quando esta ficar vazia ou o tempo acabar, a fase termina.

```
10  class Plano : public ListaSimples{
11  public:
12      Plano();
13      ~Plano();
14      void InsereNplano(int n, ListaSimples * listaGeral, ItensGanhar *itensGanhar);
15      void InsereFaseFinal(ListaSimples* listaGeral);
16      void desenha_todos_plano(sf::RenderWindow& window);
17  private:
18      float x;
19      float a, b;
20      int contador;
21  };
```

Classe Plano

- **Insere no Plano**

Ao decorrer do jogo, a função InsereNplano é chamada. Tal função é responsável por pegar elementos aleatórios da lista geral de itens e inserir no

plano, que aparecerá para o jogador naquela rodada, seguindo os critérios já ditos no primeiro tópico deste documento.

```
36  void Plano::InsereNplano(int n, ListaSimples * listaGeral, ItensGanhar *itensGanhar)
37  {
38      int i, qtd = 0, r;
39      float _x, _y;
40
41      Nodetype *no;
42      Nodetype *noPtr;
43
44      // 50% de chance de ter 1 horcrux na fase e 50% de chance de nao ter nenhuma (pode ser alterado, apenas mudando o 2 ali)
45      r = (rand() % 2);
46      if(r==0){
47          // Inserindo horcrux
48          no = itensGanhar->PegaElementoAleatorio();
49          noPtr = new Nodetype();
50          noPtr->CopiaNode(no);
51          noPtr->set_id(0);
52          noPtr->carregar(noPtr->get_info());
53          _x = (rand() % 919);
54          _y = (rand() % 300 + 321);
```

Insere Plano parte 1

```
62      for (i = qtd; i<n; i++)
63      {
64          no = listaGeral->PegaElementoAleatorioTodosTipoX(5);
65          noPtr = new Nodetype();
66          noPtr->CopiaNode(no);
67          noPtr->set_id(i);
68          noPtr->carregar(noPtr->get_info());
69          _x = (rand() % 919);
70          _y = (rand() % 300 + 321);
71          noPtr->set_posicao(_x, _y);
72
73          if (noPtr->get_info() == "imagens/pomo1.png")
74          {
75              if (noPtr->get_x() > 924)
76              {
77                  noPtr->set_posicao(noPtr->get_x() - 100, noPtr->get_y());
78              }
79              if (noPtr->get_x() < 100)
80              {
81                  noPtr->set_posicao(noPtr->get_x() + 100, noPtr->get_y());
82              }
83          }
84
85          noPtr->set_origem(noPtr->_sprite.getLocalBounds().width/2, noPtr->_sprite.getLocalBounds().height/2);
86
87          Insere(noPtr);
88      }
```

Insere Plano parte 2

- **Itens Ganhar**

A classe ItensGanhar também é subclasse de ListaSimples, portanto, possui todos os métodos de uma lista normal, além de suas especificidades. Essa classe é utilizada para 2 listas no jogo: lista de itens ganhar e lista de horcruxes destruídas. À medida que as horcruxes vão sendo destruídas pelo jogador, além de ser removida da lista plano, também é removida do itens ganhar e é inserida na lista de horcruxes destruídas, que aparece nas transições entre fases. A lista de itens ganhar é utilizada, em especial, na própria inserção no plano, pois não é permitido aparecer horcruxes que já foram destruídas, portanto a lista geral não seria muito útil nesses casos. Já a lista de horcruxes destruídas é utilizada, principalmente, na verificação da passagem para a fase final. Quando o jogador completa uma quantidade de 6

elementos nesta lista, significa que ele já destruiu todas as horcruxes disponibilizadas e já está pronto para o "chefão".

```
class ItensGanhar : public ListaSimples{
public:
    ItensGanhar();
    ~ItensGanhar();
    void InicializaItensGanhar(ListaSimples *listaGeral);
    void ProcuraRemove(string info, bool &DeuCerto, ItensGanhar *destruidos, ListaSimples *todosItens);
};
```

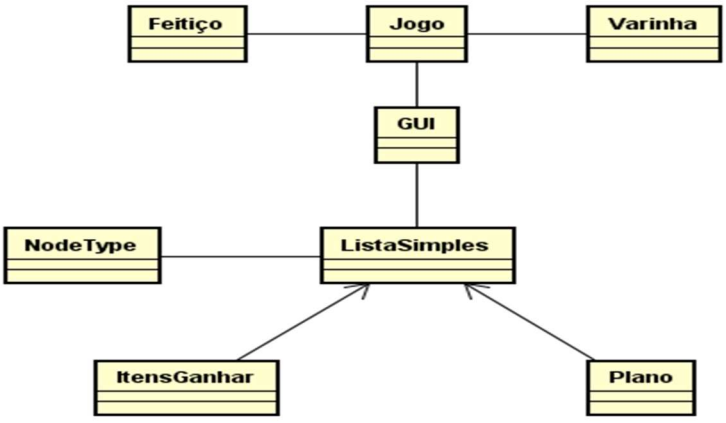
Classe ItensGanhar

As demais classes foram implementadas para objetivos específicos no jogo, em especial para preencher os quesitos da interface gráfica.

3. Diagramas

a. Diagrama da arquitetura do software

b. Diagrama das classes do software



4. Conclusão

O desenvolvimento do trabalho 2, da matéria de Estrutura de Dados do departamento de computação, ministrada por Roberto Ferrari, foi mais direcionada que o trabalho um, pela familiarização com a ferramenta visual sfml. Porém, como a complexidade do trabalho dois supera a do primeiro, foram encontradas diversas barreiras e desafios até a finalização do jogo.

A distribuição entre o grupo para a execução do projeto, foi repartido de tal maneira: Renata e Rodrigo com a parte do desenvolvimento do código, implementação das listas, etc; Renata e Rafael com o desenvolvimento do sfml; Rafael com a parte de design.

Em especial, neste trabalho foi utilizada a estrutura de dados Lista, implementada com nós, possibilitando um dinamismo no tamanho, diferentemente do trabalho 1. Concluiu-se, então, a partir dessa comparação entre os dois trabalhos, que essa maneira, implementada neste trabalho, é mais interessante. Isso fez o grupo decidir implementar o terceiro jogo seguindo o padrão deste.