Universidade Federal Rural do Rio de Janeiro

Sistemas de Informação

DOCUMENTAÇÃO

Trabalho em Java

Linguagem de Programação III

Claudia Barreto de Oliveira

20200019331

Linguagem de Programação III

Rafael Bernardo Teixeira

Rio de Janeiro, 19 de dezembro de 2021.

1. Introdução

Neste trabalho foi necessário demonstrar conceitos aprendidos durante a disciplina de Linguagem de Programação III, sejam estes herança, polimorfismo, encapsulamento, interface e genéricos nos objetos. De todo modo, seguindo a proposta do trabalho, elaborei um .jar (executável java) que apresenta um jogo do tipo "cookie clicker" que consiste em receber os clicks do mouse para computar valores e dar seguinte a operação do jogo. Toda a parte de codificação foi baseada em documentações do próprio java e de bibliotecas utilizadas.

Ademais, foi utilizada a biblioteca Processing que torna capaz a interação entre usuário e a interface do jogo em si.

Quanto às artes do jogo em si, também foi utilizado Adobe Photoshop, Paint e bases retiradas da internet para elaboração dos cenários e dos personagens.

2. Apresentação

O trabalho consiste em ser apresentado como um jogo, tendo como inspiração a série americana "Mr.Robot : Sociedade Hacker", que apresenta Elliot Alderson como um jovem adulto, analista de segurança da informação durante o dia, e um hacker durante a noite. Pensando na série, criei o "Mr.cat: Sociedade Gato" que apresenta o personagem principal Mr.Cat entrando para um exército de gatos hackers cujo objetivo é dominar o cibermundo.

Fig 1: exemplo de personagem

O jogo é do gênero “cookie clicker” conceito que surgiu em 2013 onde os jogadores clicando recebem um cookie e podem gastar o cookie com novos recursos que aumentam progressivamente a quantidade de cookies totais. No contexto do jogo criado para o trabalho em si, o jogador ao efetuar um click recebe um “exploit” que funciona como a moeda deste jogo. Ao receber “exploits” o jogador poderá ter acesso a mais ferramentas tais quais: Arduino, RaspBerry PI, Rubber Ducky que são componentes de hardwares muito famosos e amplamente utilizados no contexto da Tecnologia da Informação, juntamente a Computador com Kali Linux, BotNet e Exploit o ESTADO que consiste no fim do jogo.

O jogo em si tem curta duração e tem como essência apresentar alguns conceitos aprendidos na disciplina além de demonstrar as possibilidades da manipulação desta linguagem de programação.

3. Codificação

Quanto a parte da codificação em si, é necessário apresentar o código para melhor entendimento.

```
import Melhorias.*;
import processing.core.*;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

> Em primeiro momento é necessário fazer o import das classes do próprio java e da biblioteca processing.

> Criar a classe para abrir a interface do jogo

```
public class Principal extends PApplet {

    public static void main(String args[]) {
        PApplet.main("Principal");
    }

    PImage imgMenu;
    PImage imgInstrucoes;
    PImage imgFundo;
    PImage imgEndgame;


    enum STATUS { NO_MENU, TUTORIAL, EM_JOGO, ENDGAME };
    STATUS status = STATUS.NO_MENU;
    Map<String, PImage> imagens = new HashMap<>();

    private int quantidadeExploits = 0;
    private List<Melhoria> listaMelhorias = new ArrayList<>();
    DecimalFormat formatar = new DecimalFormat("0");

    private int contagemQuadros = 0; //considerando que cada segundo tem 60 draw()

    @Override
    public void settings() {
        size(1280, 720);
    }
```

> O método setup é responsável por inicializar as variáveis que são utilizadas no programa. Nele carregamos as PImage, que serão utilizadas na interface.
Também, foi criada uma lista contendo **objetos** que **herdam** valores e métodos da classe **Melhorias.**

```
    @Override
    public void setup() {
        imgMenu = loadImage("imagens/hackercats.png");
        imgInstrucoes = loadImage("imagens/instrucoes.png");
        imgFundo = loadImage("imagens/gatinhacker.png");
        imgEndgame = loadImage("imagens/fim.png");

        listaMelhorias.add(new Arduino());
        listaMelhorias.add(new RaspberryPI());
        listaMelhorias.add(new Rubberducky());
        listaMelhorias.add(new PCcomKALILINUX());
        listaMelhorias.add(new BotNet());
        listaMelhorias.add(new ExploitarEstado());

        for(Melhoria i : listaMelhorias){
            imagens.put(i.getDescricao(), loadImage("imagens/" + i.getFoto()));
        }
    }
```

> Como o programa baseia-se em tempo (segundos), uma variável foi criada para contar quantas vezes a função draw foi chamada. Considera-se que a função draw é chamada a cada frame e por padrão 60 frames por segundo.

> Condições:

Dependendo o status do jogo (EM_MENU,TUTORIAL,EM_JOGO,ENDGAME) elementos diferentes são renderizados.

```
  @Override
  public void draw() {
      if(contagemQuadros == 60){
         adicionaExploits();
         contagemQuadros = 1;
      } else { contagemQuadros += 1; }

      clear();

      if(status == STATUS.NO_MENU){
         image(imgMenu, 0, 0);

         fill(200);

         rect(500, 610, 200, 50, 10,10,10,10);
         textSize(30);

         fill(50);
         text("Jogar", 560, 645);
      } else if(status == STATUS.TUTORIAL) {
         image(imgInstrucoes, 0, 0);

         fill(200);

         rect(500, 610, 200, 50, 10,10,10,10);
         textSize(30);
```

```
            fill(50);
            text("Começar", 540, 645);

        } else if(status == STATUS.EM_JOGO) {

            image(imgFundo, 0, 0);

            int x = 130; int y = 60;
            for(Melhoria i : listaMelhorias){
                fill(255);
                rect(x,y, 150, 120, 8,8,8,8);
                fill(0);
                textSize(16);
                text(i.getDescricao() + " (" + i.getQuantidade() + ")", x + 5, y + 20);

                image(imagens.get(i.getDescricao()), x + 44, y + 35);

                text("Custo: " + formatar.format(i.getCustoCompra() + i.getQuantidade() *
i.getCustoCompra() * 0.1), x + 35, y + 110);
                x += 160;
            }

            fill(255);
            textSize(20);
            text("Quantidade Exploits: " + quantidadeExploits, 150, 345);

        } else {
            image(imgEndgame, 0, 0);
        }

    }


    @Override
```

> A função mousePressed é executada toda vez que o usuário clica com o mouse e com ela é possível reconhecer a posição que o usuário clicou, onde é possível entender se o usuário clicou dentro ou fora de algum botão.

```
public void mousePressed() {
    if(status == STATUS.NO_MENU) {
      if (mouseX >= 500 && mouseX <= 700 && mouseY >= 610 && mouseY <= 660) {
        // MUDA TELA
        status = STATUS.TUTORIAL;
      }
    } else if(status == STATUS.TUTORIAL) {
      if (mouseX >= 500 && mouseX <= 700 && mouseY >= 610 && mouseY <= 660) {
        // MUDA TELA
        status = STATUS.EM_JOGO;
      }
    } else {
      //CONTABILIZA CLICKS
      int x = 130; int y = 100;
      for(Melhoria i : listaMelhorias){
        if(mouseX >= x && mouseX <= x + 150 && mouseY >= y && mouseY <= y +
120 ){
          if(quantidadeExploits >= (i.getCustoCompra() + i.getQuantidade() * 1.1)){
            i.compra();
            quantidadeExploits -= (i.getCustoCompra() + i.getQuantidade() * 1.1);
            if(i.getDescricao() == "Exploit ESTADO") {
              status = STATUS.ENDGAME;
            }
          }
        }
        x += 160;
      }
      quantidadeExploits += 1;
    }
  }
```

```
    private void adicionaExploits(){
        for(Melhoria i : listaMelhorias){
            quantidadeExploits += i.getQuantidade() * i.getExploitsPorSegundo();
        }
    }

}
```

> Essa classe é abstrata e é herdada por todas as outras classes do tipo melhoria. Ela estabelece todos os métodos padrões para as melhorias.

```
package Melhorias;

import processing.core.PImage;

public abstract class Melhoria {

    private int exploitsPorSegundo;
    private String descricao;
    private int quantidade = 0;
    private int custoCompra;

    private String foto;

    public Melhoria(int exploitsPorSegundo, String descricao,  int custoCompra, String foto) {
        this.exploitsPorSegundo = exploitsPorSegundo;
        this.descricao = descricao;
        this.custoCompra = custoCompra;
        this.foto = foto;
```

```
    }

    public void compra(){
        this.quantidade += 1;
    }

    public int getExploitsPorSegundo() {
        return exploitsPorSegundo;
    }

    public String getDescricao() {
        return descricao;
    }

    public int getQuantidade() {
        return quantidade;
    }

    public int getCustoCompra() {
        return custoCompra;
    }

    public String getFoto() {
        return foto;
    }
}
```

> MELHORIAS

Exemplo:

```
package Melhorias;

public class ExploitarEstado extends Melhoria{
    public ExploitarEstado() {
        super(0, "Exploit ESTADO", 20000, "hack.png");
    }

    @Override
    public void compra(){
        System.out.println("ACABOU");

        //polimorfismo
    }
}
```