FERNANDO HENRIQUE DA SILVA
LUCAS OLIVEIRA PEREIRA
MARCO YOSHIRO KUBOYAMA DE CAMARGO

JOGO DE NAVES - BATTLE SPACE

Relatório apresentado à Universidade Federal do ABC como parte dos requisitos para aprovação na disciplina de Metodologia e Algoritmos Computacionais do Curso de Bacharelado em Ciência e Tecnologia.

Professores: José Artur Quilici Gonzalez
Francisco José Fraga da Silva

Santo André - SP
2008

Sumário

1 INTRODUÇÃO

A escolha do tema para o projeto foi motivada pelo desafio de interesse comum aos componentes da equipe de criar um programa dinâmico onde o usuário pode interagir em tempo real com o aplicativo, diferentemente dos aplicativos que estacionários que estávamos acostumados a fazer.

1.1 Cenário do Projeto

Será desenvolvido um jogo com interface gráfica simples semelhante aos implementados como opção de entretenimento em dispositivos portáteis tais como celulares.

Neste jogo o usuário comandará uma nave espacial que não poderá entrar em contato com seus inimigos sob pena de após perder certo número de “vidas” finalizar aquela rodada. Para isso ela poderá se movimentar para todos os lados na tela e quando julgar conveniente atirar nas naves inimigas. Cada nave inimiga após ser “destruída” não volta ao jogo naquela jogada e sede ao usuário certa quantidade de pontos conforme o seu grau de dificuldade.Ao tocar naves inimigas ou ser atingido pelos tiros da mesmas uma determinada quantidade de vezes o jogo é encerrado, o nome do jogador é requisitado e o número de pontos obtidos pelo usuário assim como seu nome são armazenados e apresentados quando solicitados em ordem decrescente no “Hall da Fama”.

2 IMPLEMENTAÇÃO DO PROJETO

2.1 Tabela com os Arquivos do Projeto

Nome do Arquivo	Nome do Responsável
BattleSpace.java	Marco Yoshiro Kuboyama de Camargo
Jogo.java	Todos
Entidade.java	Lucas Pereira Oliveira
Tiro.java	Marco Yoshiro Kuboyama de Camargo
Nave.java	Lucas Pereira Oliveira
Jogador.java	Fernando Henrique Silva
Inimigo.java	Fernando Henrique Silva
Pontuacao.java	Marco Yoshiro Kuboyama de Camargo

2.2 Descrição das Classes e Métodos

2.2.1 BattleSpace.java

Classe que recebe o nome do aplicativo e contém o método main(), através de seus métodos que serão descritos abaixo controla a execução do jogo, a criação, exibição e funcionamento da abertura e do menu além de exibir o ranking. Esta classe implementa a interface ActionListener para capturar as ações do usuário via teclado e mouse.

2.2.1.1 Método main()

Cria a janela de abertura com uma imagem de fundo e um botão para iniciar o jogo.

2.2.1.2 Método criaMenu ()

Instancia os botões com as opções do menu, adiciona todos eles a uma barra de opções e ao final retorna essa barra.

2.2.1.3 Método getPlacar()

Ação estática que recebe dados através de um arquivo de texto e adiciona a um arrayList objetos do tipo Pontuação com os parâmetros lidos.

2.2.1.4 Método mostrarPlacar()

Também estático recorre a método supracitado, organiza o arrayList obtido e exibe em uma janela os dez melhores classificados em ordem decrescente de pontuação.

2.2.1.5 Método salvarPlacar (int pontos)

Recebe com parâmetro um inteiro que representa os pontos conseguidos pelo jogador na ultima jogada, pergunta ao usuário o nome e cria um novo objeto do tipo Pontuação e o adiciona ao arrayList, organiza essa lista e exporta a mesma no formato .txt.

2.2.1.6 Método actionPerformed(ActionEvent e)

Implementado obrigatoriamente por ser da interface ActionListener, recebe como parâmetro um evento, que no caso é o clique no botão iniciar e então fecha abertura e inicia o jogo.

2.2.2 Jogo.java

Classe filha de JFrame que implementa as interfaces KeyListener e ActionListener, é responsável por todo o funcionamento lógico e gráfico do jogo em si.

2.2.2.1 Método Jogo()

Construtor da classe, cria a janela e o menu da mesma após isso inicia o jogo.

2.2.2.2 Método criaMenu()

Gera o menu da janela do jogo com as opções “Novo Jogo”, “Placar” e “Sair”, recorrendo ao método de mesmo nome da classe BattleSpace.

2.2.2.3 Método actionPerformed (ActionEvent evento)

Implementado da interface ActionListener controla a entrada de dados através do menu.

2.2.2.4 Método iniciar ()

Começa um novo jogo limpando a tela, criando um novo objeto do tipo jogador, novos objetos inimigos e redesenhando a tela.

2.2.2.5 Métodos keyPressed(KeyEvent evento) e keyTyped(KeyEvent evento)

Métodos da interface KeyListener que reconhecem se alguma tecla foi ou está sendo pressionada.

2.2.2.6 Método keyReleased (KeyEvent evento)

Verifica quando uma tecla foi solta.

2.2.2.7 Método teclado (KeyEvent evento)

Recorre aos três métodos citados acima para verificar qual tecla está sendo utilizada e realiza a ação determinada para ela.

2.2.2.8 Método colisaoTiro()

Recebe como parâmetro um objeto Nave e verifica se esse objeto colide com um objeto do tipo Tiro.

2.2.2.9 Método acoesInimigos ()

Realiza a movimentação dos objetos do tipo inimigo e os faz atirar.

2.2.2.10 Método moverTiros ()

Organiza a movimentação dos objetos do tipo Tiro pela tela.

2.2.2.11 Método gameover ()

Pinta a tela de vermelho e salva a pontuação atual através do método da classe BattleSpace.

2.2.2.12 Método jogoLoop ()

Recorre a todos os métodos da sua classe para criar todo o jogo.

2.2.3 Entidade.java

Classe abstrata que modela todas as entidades do jogo, ou seja,é a classe mãe de todos os objetos do jogo, portanto, contem todos atributos e métodos básicos comuns a todos objetos. Ela herda a classe javax.swing.JLabel, pois usa vários atributos e métodos da mesma.

2.2.3.1 Método Entidade()

Construtor da classe, inicializa as variáveis através de parâmetros que recebe.

2.2.3.2 Método colisao(Entidade E)

Recebe como parâmetro uma entidade, verifica se houve uma colisão e retorna uma variável tipo boolean com valor true caso as duas entidades tenham colidido ou false caso contrário.

2.2.3.3 Métodos Getters e Setters

Implementa todos os métodos getters e setters da classe mãe.

2.2.4 Tiro.java

Filha da classe abstrata Entidade por isso herda todos os atributos e métodos da super classe, cria o atributo privado id e é responsável pelo gerenciamento de objetos do seu tipo através dos seus métodos.

2.2.4.1 Método Tiro()

Construtor desta classe, recorre ao construtor da classe mãe para inicializar as variáveis herdadas e depois inicializa seu atributo especifico.

2.2.4.2 Método Movimentar()

Método responsável pela movimentação de um objeto desta classe na tela do jogo.

2.2.4.3 Método getId()

Método que retorna variável id, necessário devido ao encapsulamento do atributo.

2.2.5 Nave.java

Classe abstrata que modela todas as naves do jogo, tanto a do jogador quanto as inimigas, herda Entidade e é mãe de Jogador e Inimigo, contém todos atributos e métodos abstratos comuns a todas as naves.

2.2.5.1 Método Nave()

Construtor da classe, inicializa todas as variáveis através de parâmetros.

2.2.5.2 Método atirar() e movimentar()

Métodos abstratos que são implementados nas filhas.

2.2.5.3 Métodos Getters e Setters

Métodos de retorno e alteração de dados necessários devido ao encapsulamento.

2.2.6 Jogador.java

Classe filha de Nave com os atributos específicos para controlar o número máximo de tiros, o número de tiros na tela e a barra de vida do jogador.

2.2.6.1 Método Jogador()

Construtor da classe que inicializa as variáveis e os atributos especificos.

2.2.6.2 Método movimentar (int x, int y)

Implementado da classe mãe responsável pela movimentação dos objetos deste tipo.

2.2.6.3 Método atirar()

Implementado da classe mãe cria objetos do tipo tiro e os movimenta pela tela.

2.2.6.4 Método tiroSumiu()

Verifica quando um tiro disparado por um objeto do tipo jogador sai da tela.

2.2.6.5 Métodos Getters e Setters

Métodos de retorno e alteração de dados necessários devido ao encapsulamento.

2.2.7 Inimigo.java

Também herda a classe Nave, implementa os métodos da mãe e cria seu atributo especifico tipo, para classificar os inimigos em diferente níveis.

2.2.7.1 Método Inimigo()

Construtor da classe que inicializa as variáveis herdadas e o atributo especifico.

2.2.7.2 Método características()

Define as característica de cada objeto deste tipo.

2.2.7.3 Método defineTipo()

Recorre ao método anterior e inicializa as características através do tipo passado como parâmetro.

2.2.7.4 Método movimentar (int x, int y)

Implementado da classe mãe responsável pela movimentação dos objetos deste tipo.

2.2.7.5 Método atirar()

Implementado da classe mãe cria objetos do tipo tiro e os movimenta pela tela.

2.2.6.6 Métodos Getter e Setter

Métodos de retorno e alteração de dados necessários devido ao encapsulamento.

2.2.8 Pontuacao.java

Responsável por gerenciar o ranking do aplicativo através dos seus métodos, ela implementa a interface Comparator e obrigatoriamente o método compare.

2.2.8.1 Método Pontuacao()

Construtor da classe que inicializa as variáveis nome e pontos.

2.2.8.2 Método compare()

Método da interface Comparator para ordenar os objetos.

2.2.8.3 Método Getter e Setter

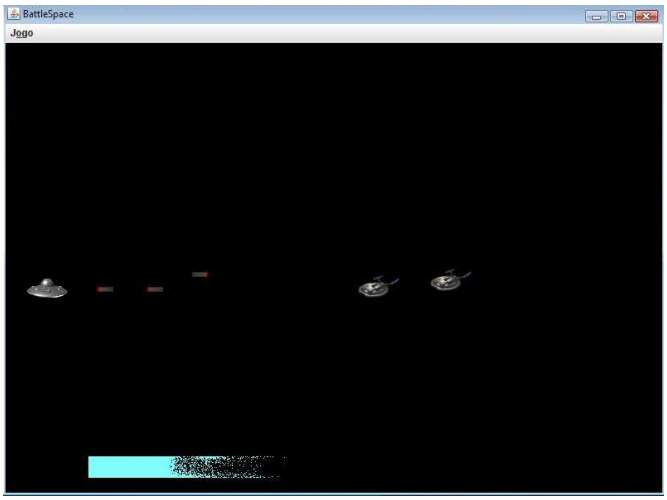
Métodos de retorno e alteração de dados necessários devido ao encapsulamento.

2.3 Modelagem

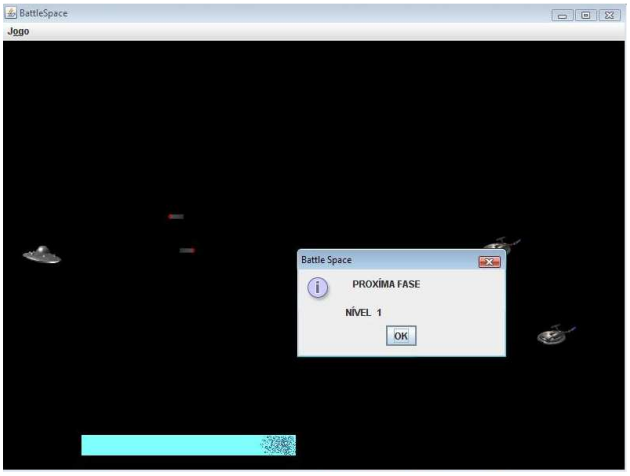
2.4 Explicação do Funcionamento do Aplicativo

Quando o programa é executado, uma janela de abertura é aberta, contendo o logotipo do jogo e um botão “iniciar”

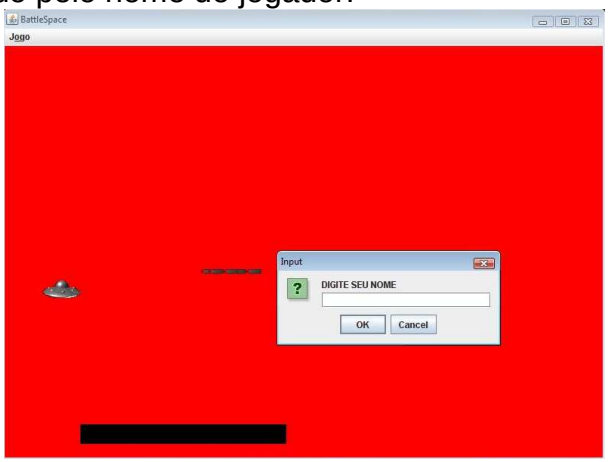
Se o usuário clicar no botão “iniciar”, o jogo é iniciado. O objetivo de jogo é conquistar pontos destruindo as espaçonaves inimigas, usando as setas do teclado para controlar a nave do jogador e o botão Ctrl para disparar.



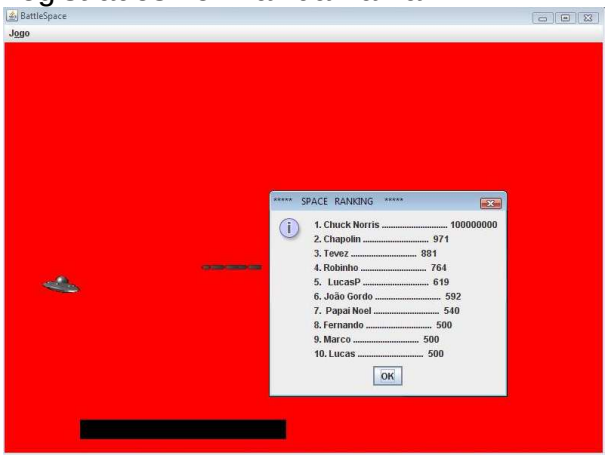
Depois de derrotar um certo numero de inimigos o jogador “passa de fase”, a “vida” do jogador é restaurada e os inimigos ficam mais fortes e numerosos.



No canto esquerdo inferior da tela há uma barra azul. Conforme o jogador vai sofrendo danos causados pelos tiros inimigos ou colisões com os inimigos suicidas a barra azul vai sendo pintada de preto, representando o dano sofrido pelo jogador, quando a barra fica totalmente negra, isso significa que a “vida” do jogador acabou e, portanto, ele foi derrotado. Então uma janela é aberta perguntando pelo nome do jogador.



Caso o jogador conquiste pontos o suficiente, seu nome e sua pontuação ficaram registrados no “Hall da Fama”.



O Menu do jogo pode ser acessado a qualquer momento durante a partida ou depois que o jogador for derrotado, através do menu é possível iniciar uma nova partida selecionando “Novo”, ver o “Hall da Fama” selecionando “Placar” ou sair do jogo selecionando “Sair”. Também é possível sair do jogo apertando “esc”;

3 CONCLUSÃO

Através dos exemplos de funcionamento do aplicativo nota-se que os objetivos propostos tanto na introdução quanto no cenário do projeto foram plenamente atingidos, o jogo funciona como o especificado, o placar com nome e pontuação em ordem decrescente são apresentados quando o comando no menu é acionado e esse dados são automaticamente importados e exportados mantendo um registro atualizado. A modelagem do projeto teve que ser ligeiramente alterada em com a criação da classe “Pontuacao”, que gerencia o placar do jogo, no entanto a essência da UML original foi integralmente preservada.

O desenvolvimento do projeto teve a participação ativa e sempre crítica de todos os componentes do grupo que ao se reunirem conseguiram trocar idéias e discutir opiniões para a melhoria do aplicativo sempre com muito interesse e força de vontade. No entanto em senso comum decidimos enaltecer a participação do integrante Fernando Henrique Silva que muitas vezes auxiliou com exatidão na identificação e correção de defeitos em trechos do programa que não eram de sua responsabilidade.

4 REFERÊNCIAS BIBLIOGRÁFICAS

DEITEL M. Harvei e DEITEL, Paul J. JAVA: Como Programar. 6ª Edição. São Paulo: Bookman, 2005.

APÊNDICE

A. Arquivo “BattleSpace.java”

```
/*
* UFABC – BCT - BC0502
* BattleSpace.java
*
* Responsável: MARCO YOSHIRO KUBOYAMA DE CAMARGO
*
* Data: 08/08/2008. (data da atualização mais recente)
*/

import java.awt.event.ActionEvent;
import java.io.*;
import javax.swing.*;
import java.awt.Color;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Collections;

public class BattleSpace implements ActionListener{

	private static Jogo jogo;
	private static ArrayList <Pontuacao> pontuacao;
	private static String arqPlacar = "placar.txt";
      static JFrame abertura;

	// main
	public static void main (String[] args) {
		pontuacao = new ArrayList <Pontuacao>();

	  abertura = new JFrame ();
	  abertura.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
		abertura.setBounds(0, 0 , 800, 600);
		abertura.getContentPane().setBackground(Color.BLACK);


	  JButton I = new JButton("INICIAR");
	  I.addActionListener(new BattleSpace());
	  I.setBounds(350, 450, 100,20);
		abertura.add(I);
	  abertura.add(new JLabel (new ImageIcon ("BattleSpace.png")));
		abertura.setVisible(true);

	}

	/**
	 * Cria um menu com as opções:
	 *    Novo = Inicia um novo jogo
	 *    Placar = Mostra o placar
	 *    Sair = Sai do jogo
	 * @return Barra de menu
	 */
	public static JMenuBar criaMenu () {

		// Barra de menu
		JMenuBar menubar = new JMenuBar();

		// Menu
		JMenu menu = new JMenu ("Jogo");
```

```
		menu.setMnemonic ('o'); // Tecla de atalho

		// Submenus
		JMenuItem submenu[] = new JMenuItem[3];

		submenu[0] = new JMenuItem ("Novo");
		submenu[0].setMnemonic ('N');

		submenu[1] = new JMenuItem ("Placar");
		submenu[1].setMnemonic ('P');

		submenu[2] = new JMenuItem ("Sair");
		submenu[2].setMnemonic ('S');

		// Adiciona os submenus ao menu
		for (int i=0; i < submenu.length; i++)
			menu.add ( submenu[i] );

		// Adiciona o menu a barra de menu
		menubar.add (menu);

		return menubar;
	}

	/**
	 * Pega todas as pontuações do arquivo
	 * e armazena no arraylist pontuacao
	 */
	public static void getPlacar () {
		try {
			BufferedReader in = new BufferedReader (
				new FileReader (arqPlacar));

			pontuacao.clear();
			String linha = "" , campos[ ] = null;

			while ( (linha = in.readLine()) != null ) {
				campos = linha.split (":");
				// é necessário a linha ter somente dois campos
				if (campos.length == 2)
					pontuacao.add (new Pontuacao (campos[1],
							Integer.parseInt (campos[0])));
			}
			in.close();
		}
		// se o arquivo não existe, ele é criado
		catch (FileNotFoundException erro) {
			try {
				BufferedWriter out = new BufferedWriter (
						new FileWriter (arqPlacar));
				out.close();
			} catch (IOException ioerro) { System.out.println (erro); }
		}
		catch (IOException erro) {
			System.out.println ("Erro na leitura dos dados");
		}
	}

	/**
	 * Mostra todas as pontuações
```

```
         */
        public static void mostrarPlacar () {
                getPlacar();
                // ordena da maior pontuação para a menor
                Collections.sort( pontuacao, new Pontuacao("",0) );

                String msg = "";
                for (int i=0; i <10; i++)
                msg +=  String.format((i+1)+". %s\t .....\t %4d\n",
                     pontuacao.get(i).getNome(), pontuacao.get(i).getPontos() );
                JOptionPane.showMessageDialog  (null,  msg,  "*****       SPACE      RANKING
*****",1);
        }

        /**
         * Salva a pontuação do jogador no placar
         * @param pontos pontuação que irá para o placar
         */
        public static void salvarPlacar (int pontos) {
                // pede o nome do jogador
                String nome = "";
                while (nome.isEmpty()) {
                        try { nome = JOptionPane.showInputDialog ("DIGITE SEU NOME"); }
                        catch (Exception erro) { nome = ""; }
                }
                // remove os ":" para não dar erro na hora de ler o arquivo
                nome = nome.replaceAll(":","");

                getPlacar();
                // adiciona a nova pontuação
                pontuacao.add (new Pontuacao (nome, pontos));

                // ordena da maior pontuação para a menor
                Collections.sort( pontuacao, new Pontuacao("",0) );

                // salva no arquivo
                try {
                        BufferedWriter out = new BufferedWriter (
                                        new FileWriter (arqPlacar));
                        String msg = "";
                        for (int i=0; i < pontuacao.size(); i++)
                                msg += pontuacao.get(i).getPontos() + ":" +
                                                pontuacao.get(i).getNome()   + "\n";
                        out.write (msg);
                        out.close();
                }
                catch (IOException erro) {
                        System.out.println ("Erro na escrita dos dados");
                }
        }

    public void actionPerformed(ActionEvent e) {
        jogo = new Jogo ("BattleSpace", 800, 600);
        abertura.setVisible(false);
    }
}
```

B. Arquivo “Jogo.java”

```
/*
 * UFABC – BCT - BC0502
 * Jogo.java
 *
 * Responsável:  MARCO YOSHIRO KUBOYAMA DE CAMARGO
 *               FERNANDO HENRIQUE SILVA
 *               LUCAS PEREIRA OLIVEIRA
 *
 * Data: 10/08/2008. (data da atualização mais recente)
 */

import java.awt.Color;
import java.awt.event.*;
import javax.swing.*;
import java.util.ArrayList;

public class Jogo extends JFrame implements KeyListener , ActionListener {

        // Jogador, inimigos e tiros
        private Jogador jogador;
        private ArrayList <Inimigo> inimigo = new ArrayList <Inimigo>(3);
        private ArrayList <Tiro> tiro = new ArrayList <Tiro>();

    private JLabel lifebar;
    private int fase;

        // Cronômetro para excecutar alguns métodos periodicamente
        Timer cronometro;

        /**
         * Construtor, cria uma janela com menus
         * @param titulo título da janela
         * @param largura largura da tela em pixels
         * @param altura altura  da tela em pixels
         */
        public Jogo (String titulo, int largura, int altura) {

                // Cria janela
                super (titulo);

                // arruma o modo como as coisas aparecem manualmente
                setLayout(null);

                // Ajusta a tela
                setBounds(0, 0, largura, altura);

                setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

                // Permite a janela processar eventos do teclado
                addKeyListener (this);

                criaMenu();

                // A cada 40 milisegundos excecuta um método
                cronometro = new Timer (40, new ActionListener() {
                        public void actionPerformed (ActionEvent evt) {
                                jogoLoop(); } } );
```

```
                // Inicia o jogo
                iniciar();
        }

        /**
         * Cria o menu da janela
         */
        public void criaMenu () {
                // Adiciona a barra de menu a janela
                setJMenuBar (BattleSpace.criaMenu());

                // Adiciona eventos aos menus
                JMenu m = getJMenuBar().getMenu(0);
                m.addActionListener(this);
                for (int i=0; i < m.getItemCount(); i++)
                        m.getItem(i).addActionListener (this);
        }

        // Eventos relacionados ao menu da janela
        public void actionPerformed (ActionEvent evento){
                Object o = evento.getSource();
                JMenu m = getJMenuBar().getMenu(0);

                if (o == m.getItem(0)) // Novo
                        iniciar();
                else if (o == m.getItem(1)) // Placar
                        BattleSpace.mostrarPlacar();
                else if (o == m.getItem(2)) // Sair
                        System.exit(0);
        }

        /**
      * Inicia um jogo novo
      */
        public void iniciar () {

                // Limpa tudo
          fase = 0;
                if (jogador != null) remove (jogador);
                for (int i=0; i < inimigo.size(); i++) remove (inimigo.get(i));
          if (lifebar != null) remove (lifebar);
                inimigo.clear();
                for (int i=0; i < tiro.size(); i++) remove (tiro.get(i));
                tiro.clear();
                getContentPane().setBackground (Color.BLACK); // Cor de fundo


                // Jogador
          jogador = new Jogador (0, 0, "ufo.gif", 1, 25, 20, 5);
          lifebar= new JLabel(new ImageIcon("vida20.png"));
          lifebar.setBounds(100,
500,lifebar.getIcon().getIconWidth(),lifebar.getIcon().getIconHeight());
          add (jogador);
          add (lifebar);
                lifebar.setVisible(true);
                // Cria inimigos
                for (int i=0; i < 3; i++)
                        inimigo.add (new Inimigo ( getWidth() ,
                                                (int) (Math.random() * getHeight()) , 0));
                for (int i=0; i < inimigo.size(); i++)
```

```
                add (inimigo.get(i));

        // Reinicia cronômetro
        cronometro.restart();

        // Desenha tela inicial
        repaint();
        setVisible(true);
}

// Quando uma tecla é pressionada
public void keyPressed (KeyEvent evento) { teclado (evento); }
public void keyTyped   (KeyEvent evento) { teclado (evento); }
// Quando uma tecla é solta
public void keyReleased (KeyEvent evento) {  }

/**
 * Ações do jogador via teclado
 */
public void teclado (KeyEvent evento) {

        int limite = 0;

        /**
         * 1) Verifica a tecla pressionada
         * 2) Movimenta o jogador
         * 3) Não deixa o jogador sair da tela
         */
        switch ( evento.getKeyCode() ) {

                // Baixo
                case KeyEvent.VK_DOWN:
                case KeyEvent.VK_NUMPAD2:
                case KeyEvent.VK_S:
                case KeyEvent.VK_J:
                        jogador.movimentar (0, jogador.getVelocidade());
                        limite = getHeight() - jogador.getA()*4;
                        if (jogador.getY() > limite)
                                jogador.setY (limite);
                        break;

                // Cima
                case KeyEvent.VK_UP:
                case KeyEvent.VK_NUMPAD8:
                case KeyEvent.VK_W:
                case KeyEvent.VK_K:
                        jogador.movimentar (0, -jogador.getVelocidade());
                        if (jogador.getY() < limite)
                                jogador.setY (limite);
                        break;

                // Esquerda
                case KeyEvent.VK_LEFT:
                case KeyEvent.VK_NUMPAD4:
                case KeyEvent.VK_A:
                case KeyEvent.VK_H:
                        jogador.movimentar (-jogador.getVelocidade(), 0);
                        if (jogador.getX() < limite)
                                jogador.setX (limite);
                        break;
```

```
                    // Direita
                    case KeyEvent.VK_RIGHT:
                    case KeyEvent.VK_NUMPAD6:
                    case KeyEvent.VK_D:
                    case KeyEvent.VK_L:
                            jogador.movimentar (jogador.getVelocidade(), 0);
                            limite = getWidth()/2 - jogador.getL();
                            if (jogador.getX() > limite)
                                    jogador.setX (limite);
                            break;

                    // Atira
                    case KeyEvent.VK_COMMA:
            case KeyEvent.VK_CONTROL:
                            if (jogador.getNTiros() < jogador.getMaxTiros()) {
                                    tiro.add (jogador.atirar());
                                    add( tiro.get( tiro.size() - 1 ) );
                                    setVisible(true); // Para o tiro aparecer na tela
                            }
                            break;

                    // Sai do Jogo
                    case KeyEvent.VK_ESCAPE: System.exit(0); break;
                    default: break;
            }
}

/**
 * Verifica se houve alguma colisão com os tiros e alguma nave.
 * Se houve, tira um pouco da vida (-1) e remove o tiro da tela
 * @param nave nave que pode ter colidido com o tiro
 */
public void colisaoTiro(Nave nave) {
            for (int i = 0; i < tiro.size(); i++) {
                    if (tiro.get(i).colisao(nave)) {
                            nave.setVida (nave.getVida() - tiro.get(i).getDano());
                lifebar.setIcon(new ImageIcon(jogador.getLifebar()));
                            tiro.get(i).setImagem("");
                            // Se o tiro for do jogador
                            if (tiro.get(i).getId()) {
                   jogador.tiroSumiu();
                   jogador.setPontos(jogador.getPontos() + nave.getPontos());
                            }
                else { // se o tiro for do inimigo
                   if (nave instanceof Inimigo)
                      nave.setVida (nave.getVida() + tiro.get(i).getDano());
                }
                            tiro.remove(i);
                    }
            }
}

// Excecuta as ações dos inimigos
public void acoesInimigos () {
            // tiro dos inimigos
            for (int i=0; i < inimigo.size(); i++) {
            if ((int) (Math.random() * (1+inimigo.get(i).getTipo())*7) == 3) {
                            tiro.add (inimigo.get(i).atirar());
                            add (tiro.get( tiro.size() - 1));
```

```
                }
            }
            setVisible(true); // Para o tiro aparecer na tela

            // Move os inimigos
            for (int i=0; i < inimigo.size(); i++)
                inimigo.get(i).movimentar (jogador.getX(), jogador.getY());
        }

        // Move os tiros
        public void moverTiros () {
            for (int i=0; i < tiro.size(); i++) {
                tiro.get(i).movimentar();
                // Remove o tiro se sair da tela
                if (tiro.get(i).getX() > getWidth() || tiro.get(i).getX() < 0) {
                    tiro.get(i).setImagem("");
                    // Se o tiro for do jogador
                    if (tiro.get(i).getId())
                        jogador.tiroSumiu();
                    tiro.remove(i);
                }
            }
        }

        /**
         * Loop do jogo
         * É excecutado de acordo com o cronômetro
         */
        public void jogoLoop () {
            // Move os tiros e excecuta ações dos inimigos
            moverTiros();
            acoesInimigos();

            // Tiros e jogador
            colisaoTiro (jogador);
            // Tiros e todos os inimigos
            for (int i=0; i < inimigo.size(); i++) {
                colisaoTiro (inimigo.get(i));

                // Se algum inimigo bateu no jogador, o inimigo morre
                if (jogador.colisao( inimigo.get(i) )) {
                    jogador.setPontos( jogador.getPontos()
                                        + inimigo.get(i).getPontos() );
                    inimigo.get(i).setVida(0);
                    jogador.setVida (jogador.getVida() - 1);
                lifebar.setIcon(new ImageIcon(jogador.getLifebar()));
                break;
                }
            }

            // Verifica se algum inimigo morrreu
            for (int i=0; i < inimigo.size(); i++) {
                if (inimigo.get(i).getVida() < 1) {
                    inimigo.get(i).setLocation (getWidth(), (int) (Math.random() *
getHeight()));
                    inimigo.get(i).setTipo ((int) (Math.random() * 5));
                fase++;
                // quando passar de fase (número múltiplo de 10)
                // adiciona um inimigo e restaura a vida do jogador
                if (fase%10 == 0) {
```

```
                        jogador.setVida(20);
                        inimigo.add(new Inimigo(getWidth(),
                                (int) (Math.random() * getHeight()), 0));
                        add(inimigo.get(i));
                        setVisible(true);
                        JOptionPane.showMessageDialog(null,"    PROXÍMA FASE    \n\n NÍVEL
" + fase/10,"Battle Space",1);                          }
                          }
                }

                // Verifica se o jogador morrreu
                if (jogador.getVida() < 1)
                        gameover();
        }

        /**
         * O que acontece quando o jogo acaba
         */
        public void gameover () {
                cronometro.stop();

                // Pinta o fundo de vermelho para indicar que o jogo acabou
                getContentPane().setBackground (Color.RED);

                BattleSpace.salvarPlacar (jogador.getPontos());
        }
}
```

C. Arquivo “Entidade.java”

```
/*
 * UFABC – BCT - BC0502
 * Entidade.java
 *
 * Responsável: LUCAS PEREIRA OLIVEIRA
 *
 * Data: 07/08/2008. (data da atualização mais recente)
 *
  */

import javax.swing.ImageIcon;
import javax.swing.JLabel;

public abstract class Entidade extends JLabel {

        // Quantidade de dano
        private int dano;

        // velocidade
        int velocidade;

        /**
         * Construtor
         * @param x posição inicial no eixo x
         * @param y posição inicial no eixo y
         * @param _imagem representação visual
         * @param _d quantidade de dano
         * @param _v velocidade em pixels
         */
        public Entidade (int x, int y, String _imagem, int _d, int _v) {
```

```
		setIcon (new ImageIcon(_imagem));
		setBounds (x , y, getIcon().getIconWidth(), getIcon().getIconHeight());
		dano = _d;
		velocidade = _v;
	}

	/**
	 * Verifica se houve colisão
	 * @param outro
	 * @return true  colidiu
	 * @return false não colidiu
	 */
	public boolean colisao(Entidade outro)
	{
		//colisão horizontal
		if (getX()+getL()>outro.getX()
			&&outro.getX()+outro.getL()>getX()
			//colisão vertical
			&& getY()+getA()>outro.getY()
			&& outro.getY()+outro.getA()>getY())
				return true;

		return false;
	}

	// gets
	// getX() e getY() são herdados do JFrame
	public int getL () { return getIcon().getIconWidth(); }
	public int getA () { return getIcon().getIconHeight(); }
	public int getDano () { return dano; }
	public int getVelocidade () { return velocidade; }

	// sets
	public void setX (int x) { setLocation (x, getY()); }
	public void setY (int y) { setLocation (getX(), y); }
	public void setImagem (String _imagem) { setIcon (new ImageIcon(_imagem)); }
	public void setDano (int _d) { dano = _d; }
	public void setVelocidade (int _v) { velocidade = _v; }
}
```

D. Arquivo “Tiro.java”

```
/*
 * UFABC – BCT - BC0502
 * Tiro.java
 *
 * Responsável: MARCO YOSHIRO KUBOYAMA DE CAMARGO
 *
 * Data: 08/08/2008. (data da atualização mais recente)
 */

public class Tiro extends Entidade {

	/**
	 * Identificação do tiro
	 * true  = Jogador
	 * false = Inimigo
	 */
	private boolean id;
```

```
        private double precisao =0;
        boolean imprecisao;

            /**
             * Construtor
             * @param x posição inicial no eixo x
             * @param y posição inicial no eixo y
             * @param _imagem representação visual
             * @param _d quantidade de dano causada pelo tiro
             * @param _v velocidade em pixels
             * @param _id indica de quem é o tiro
             */
            public Tiro (int x, int y, String _imagem, int _d,
                                        int _v, boolean _id) {

                        super (x, y, _imagem, _d, _v);
                        id = _id;

                        // Inverte o sentido do tiro se for do inimigo
                        if ( ! id )
                          setVelocidade( - getVelocidade() );
            }
        public Tiro (int x, int y, String _imagem, int _d,
                                        int _v, boolean _id,double _precisao,double erro) {

                        super (x, y, _imagem, _d, _v);

               if (erro>0.5) imprecisao=true;
               precisao=_precisao;
                        id = _id;

                        // Inverte o sentido do tiro se for do inimigo
                        if ( ! id )
                          setVelocidade( - getVelocidade() );
            }

            /**
             * Movimenta o tiro
             */
             public void movimentar () {
            if(!id) //inimigo
            {
               if (imprecisao)
               setLocation (getX() + getVelocidade() , getY()+(int)(getVelocidade()*precisao));
               else
                setLocation (getX() + getVelocidade() , getY()-(int)(getVelocidade()*precisao));
            }
            else//jogador
            {
               int abc=(int)(Math.random()*10);
               if(Math.random()<0.5) abc=-abc;
               setLocation (getX() + getVelocidade() , getY()+abc);
            }
             }

             // gets
             public boolean getId () { return id; }
}
```

E. Arquivo “Nave.java”

```
/*
 * UFABC – BCT - BC0502
 * Nave.java
 *
 * Responsável: LUCAS PEREIRA OLIVEIRA
 *
 * Data: 08/08/2008. (data da atualização mais recente)
  */
public abstract class Nave extends Entidade {
	/**
	 * Define a vida da nave.
	 * Se for igual ou menor a zero a nave é destruída.
	 */
	protected int vida;

	/**
	 * Define a pontuação da nave ou a quantidade de
	 * pontos que é dada ao Jogador quando um inimigo é destruído.
	 */
	private int pontos;

	/**
	 * Construtor
	 * @param x posição inicial no eixo x
	 * @param y posição inicial no eixo y
	 * @param _imagem representação visual
	 * @param _d quantidade de dano causada pelo tiro
	 * @param _v velocidade em pixels
	 * @param _vida vida da nave
	 */

	public Nave (int x, int y, String _imagem, int _d, int _v, int _vida) {
		super (x, y, _imagem, _d, _v);
		vida = _vida;
		pontos = 0;
	}
	public abstract void movimentar (int x, int y);
	/**
	 * Faz a nave atirar
	 * @return Tiro da nave
	 */
	public abstract Tiro atirar();
	// gets
	public int getVida () { return vida; }
	public int getPontos() { return pontos; }
	// sets
	public void setVida (int _vida) { vida = _vida; }
	public void setPontos (int _pontos) { pontos = _pontos; }
}
```

F. Arquivo “Jogador.java”

```
/*
 * UFABC – BCT - BC0502
 * Jogador.java
 *
 * Responsável: FERNANDO HENRIQUE SILVA
 *
 * Data: 08/08/2008. (data da atualiação mais recente)
 */
```

```
public class Jogador extends Nave {

	// Limita o número máximo de tiros
	private int maxTiros;

	// Número de tiros que estão na tela
	private int nTiros;

    private String lifebar;

	/**
	 * Construtor
	 * @param x posição inicial no eixo x
	 * @param y posição inicial no eixo y
	 * @param _imagem representação visual
	 * @param _d quantidade de dano causada pelo tiro
	 * @param _v velocidade em pixels
	 * @param _vida vida da nave
	 * @param m limite do número máximo de tiros
	 */
	public Jogador (int x, int y, String _imagem,
                                        int _v, int _d, int _vida, int m) {
		super (x, y, _imagem, _v, _d, _vida);
		maxTiros = m;
		nTiros = 0;
	}

	/**
	 * Movimenta a nave
	 * @param x incrementa posição no eixo x
	 * @param y incrementa posição no eixo y
	 */
	public void movimentar (int x, int y) {
		if(vida > 0)
			setLocation (getX() + x , getY() + y);
	}

	/**
	 * Faz a nave atirar
	 * @return Tiro da nave
	 */
	public Tiro atirar () {
		nTiros++;
		return new Tiro (getX() + getL(), getY()+getA()/2, "tiro1.png",
                                getDano(), getVelocidade()*2 ,true);
	}

	/**
	 * Modifica o limite dos tiros
	 * @param l novo limite
	 */
	public void setMaxTiros (int l) { maxTiros = l; }

	// Um tiro saiu da tela ou atingiu um inimigo
	public void tiroSumiu () { nTiros--; }

	// gets
	public int getMaxTiros () { return maxTiros; }
	public int getNTiros () { return nTiros; }
    public String getLifebar()
```

```
    {
        return "vida"+this.getVida()+".png";
    }


}
```

G. Arquivo “Inimigo.java”

```
/*
 * UFABC – BCT - BC0502
 * Inimigo.java
 *
 * Responsável: FERNANDO HENRIQUE SILVA
 *
 * Data: 09/08/2008. (data da atualiação mais recente)
 */
public class Inimigo extends Nave {

        /**
         * Determina: dano causado pelo tiro, velocidade e vida
         * Exemplo:
          */
        private int tipo;

        /**
         * Define as características do inimigo
         * @param _d quantidade de dano causada pelo tiro
         * @param _v velocidade em pixels
         * @param _vida vida da nave
         * @param _p pontuação da nave
         */
        private void caracteristicas (int _d, int _v, int _vida, int _p) {
                setDano(_d);
                setVelocidade(_v);
                setVida(_vida);
                setPontos(_p);
        }

        /**
         * Define as características do inimigo através de seu tipo
         * @param _tipo tipo do inimigo
         */
        private void defineTipo (int _tipo) {
                tipo = _tipo;
                switch (tipo) {
                        case 0:  caracteristicas (1,  8, 2, 1); break;
                        case 1:  caracteristicas (1, 10, 2, 2); break;
                        case 2:  caracteristicas (2, 15, 2, 3); break;
                        case 3:  caracteristicas (3,  5, 1, 4); break;
                        default: caracteristicas (2, 10, tipo, tipo); break;
                }
        }

        /**
         * Construtor
         * @param x posição inicial no eixo x
```

```
	 * @param y posição inicial no eixo y
	 * @param _tipo tipo do inimigo
	 */
	public Inimigo (int x, int y, int _tipo)
	{
		super (x, y, "alfa.png", 1, 1, 1);
		defineTipo(_tipo);
	}
	/**
	 * Movimenta o inimigo baseado na posição do jogador
	 * @param x posição do jogador no eixo x
	 * @param y posição do jogador no eixo y
	 */
	public void movimentar (int x, int y) {
		// Persegue o jogador

		if (tipo < 3) {
			// jogador a direita do inimigo
			if (x > getX()) x = getVelocidade();
			else x = - getVelocidade();

			// jogador acima do inimigo
			if (y > getY())
		  y = getVelocidade();
			else
		  y = - getVelocidade();

			setLocation (getX() + x , getY() + y);
		}
		else
		{
			// inimigo fica a uma distacia do jogador
			// porém na mesma "altura"
			setLocation (x + 400 , y);
		}
	}
	/**
	 * Faz a nave atirar
	 * @return Tiro da nave
	 */
	public Tiro atirar () {
		return new Tiro (getX() - getL()/2, getY()+getA()/2, "tiro2.png",
				getDano(), getVelocidade()*2,
false,Math.random()/8,Math.random());
	}
	// gets
	public int getTipo () { return tipo; }
	// sets
	public void setTipo (int _tipo) { defineTipo(_tipo); }
}
```

H. “Pontuacao.java”

```
/*
 * UFABC – BCT - BC0502
 * Pontuacao.java
 *
 * Responsável: MARCO YOSHIRO KUBOYAMA DE CAMARGO
 *
 * Data: 03/08/2008. (data da atualização mais recente)
 */
```

```
import java.util.Comparator;

public class Pontuacao implements Comparator <Pontuacao> {

	private String nome;
	private int pontos;

	/**
	 * Construtor, inicializa as variáveis de acordo com os parâmetros
	 * @param nomeIn nome de quem possui a pontuação
	 * @param pontosIn quantidade de pontos
	 */
	public Pontuacao (String nomeIn, int pontosIn) {
		nome = nomeIn;
		pontos = pontosIn;
	}

	// ordem DEcrescente (maior pontuação vem primeiro)
	public int compare (Pontuacao p1, Pontuacao p2) {
		if (p1.pontos < p2.pontos) return 1;
		else if (p1.pontos == p2.pontos) return 0;
		else return -1;
	}

	// gets
	public String getNome() { return nome; }
	public int getPontos() { return pontos; }
}
```