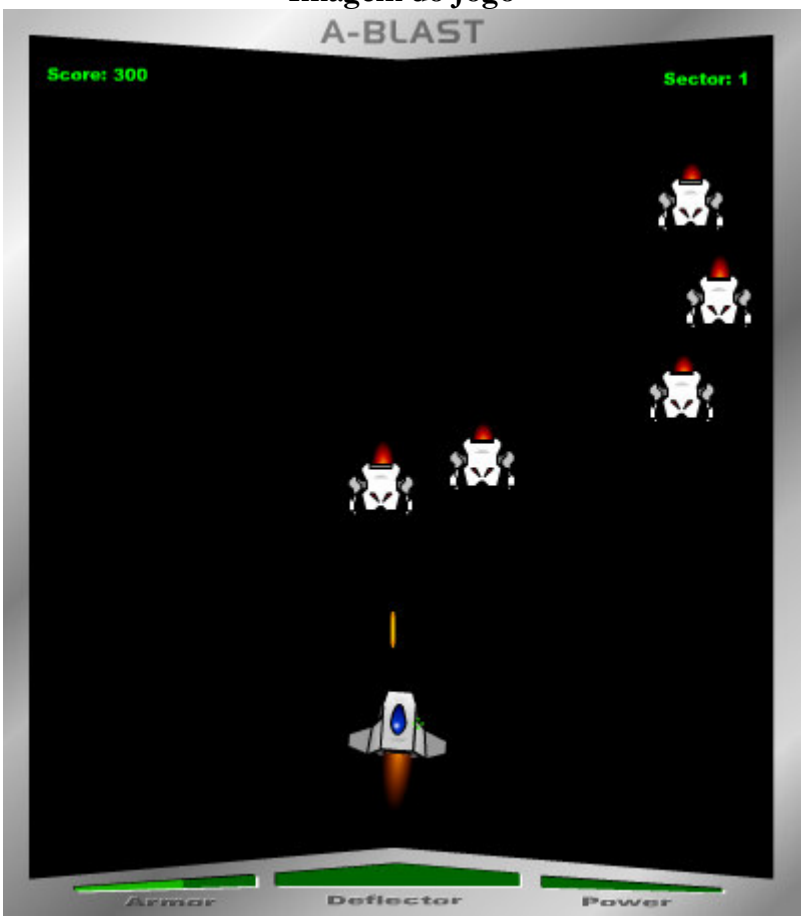
::: 1. Material de apoio – Desenvolvendo Jogos em Java

Uma boa maneira de consolidar o que já foi ensinado é sua aplicação no desenvolvimento de jogos. Para isso, serão apresentados os conceitos dados e mais alguns novos recursos para tornar o jogo ainda mais interessante.

2. Inspiração

A inspiração para este jogo pode ser encontrada no site abaixo:
http://www.absolu-flash.com/play-game/index.php?noP=2&rubrique=spacebattle

Imagem do Jogo



3. Jogo de Nave

Supondo que seja necessário o desenvolvimento de um pequeno jogo de nave espacial rodando localmente na máquina. Inicialmente, deverão ser analisadas suas características principais, são elas:

Ambiente: fundo espacial.
Usuário: Nave principal
Inimigo: Naves de cores variadas

Parte 1 do ambiente

O primeiro ponto é a criação do ambiente que representará o fundo com os elementos principais do qual o usuário irá visualizar. Isto vale não apenas em um jogo, mas também em qualquer aplicativo. Assim, em um jogo espacial, espera-se pelo menos um fundo preto.

Como é um jogo local e gráfico, será utilizado um JFrame.
Conforme já visto nas aulas passadas, para utilizar um JFrame basta a palavra chave extends seguido de JFrame.

Para utilizar o JFrame, deve-se importá-la do pacote SWING.

```
import javax.swing.JFrame;
```

No construtor da classe deverá ser incluído o título da janela(uso do método super), a cor de fundo(getContentPane().setBackground(cor)), o tamanho do frame e torná-lo visível.

Para mudar a cor de fundo, será necessário ainda importar a classe Color do pacote AWT.

```
import java.awt.Color;
```

No método main deverá ter a chamada da criação do objeto da classe e sem seguida setDefaultCloseOperation() para certificar que o processo do aplicativo será finalizado quando o usuário fechar a janela. A seguir, é apresentado o código do que já foi apresentado.

Código completo até o momento:

```
import java.awt.Color;
import javax.swing.*;

public class Jogo extends JFrame
{
	public Jogo()
	{
		super("Jogo de nave");

		//Muda a cor de fundo para preto
		getContentPane().setBackground(Color.black);

		//Muda o tamanho da janela
		setSize(510,510);

		//Deixa a janela visivel
		setVisible(true);
	}

	public static void main(String[] args)
	{
		Jogo p = new Jogo();
		p.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

Resultado da tela

Parte 2 do ambiente

Agora que já foi modelado a janela principal do jogo, bem como o fundo preto, é necessário desenhar o contorno da tela. Para isto, basta utilizar o método fillPolygon para desenho do polígono.

Como será desenhada uma figura geométrica deverá ter o método paint, juntamente com a classe Graphics. A classe Graphics deve ser importada do pacote AWT.

```
import java.awt.Graphics;

public void paint(Graphics fundo){}
```

Acrescenta-se ainda o uso do método paint da classe herdada de JFrame com a palavra super.paint();

```
super.paint(fundo);
```

A matriz com os pontos x e Y

```
int mt_x[] = {0, 510, 510, 0, 0, 25, 480, 480, 255, 25, 25};
int mt_y[] = {29, 29, 510, 510, 29, 50, 50, 480, 450, 480, 50};

A cor do preenchimento
fundo.setColor(Color.GRAY);
```

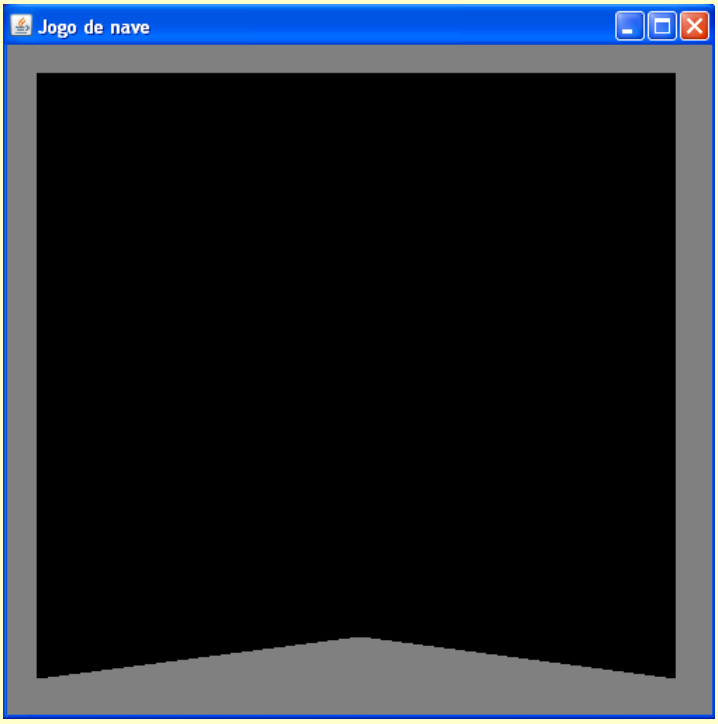
e por fim o método fillPolygon

```
fundo.fillPolygon(mt_x,mt_y,11);
```

Código completo até o momento:

```
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.*;

public class Jogo extends JFrame
{
	public Jogo()
	{
		super("Jogo de nave");

		//Muda a cor de fundo para preto
		getContentPane().setBackground(Color.black);

		//Muda o tamanho da janela
		setSize(510,510);

		//Deixa a janela visivel
		setVisible(true);
	}

	public void paint(Graphics fundo)
	{
		int mt_x[] = {0, 510, 510, 0, 0, 25, 480, 480, 255, 25, 25};
		int mt_y[] = {29, 29, 510, 510, 29, 50, 50, 480, 450, 480, 50};
		//escolhido a cor cinza para preenchimento do polígono
		fundo.setColor(Color.GRAY);
		super.paint(fundo);
		fundo.fillPolygon(mt_x,mt_y,11);
	}

	public static void main(String[] args)
	{
		Jogo p = new Jogo();
		p.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

Resultado da tela



Parte 1 do usuário

O próximo passo é colocar a nave do usuário dentro do ambiente.

Primeiro, declara-se um objeto do tipo Image.

```
private Image Nave;

Com o uso da classe Imagem deve-se importá-la do pacote awt
import java.awt.Image;
```

Segundo, no método contrutor, faz-se o mesmo receber o nome da imagem

```
Nave = Toolkit.getDefaultToolkit().getImage("nav_0.JPG");

Com o uso do ToolKit deve-se importar a classe Toolkit do pacote AWT
import java.awt.Toolkit;
```

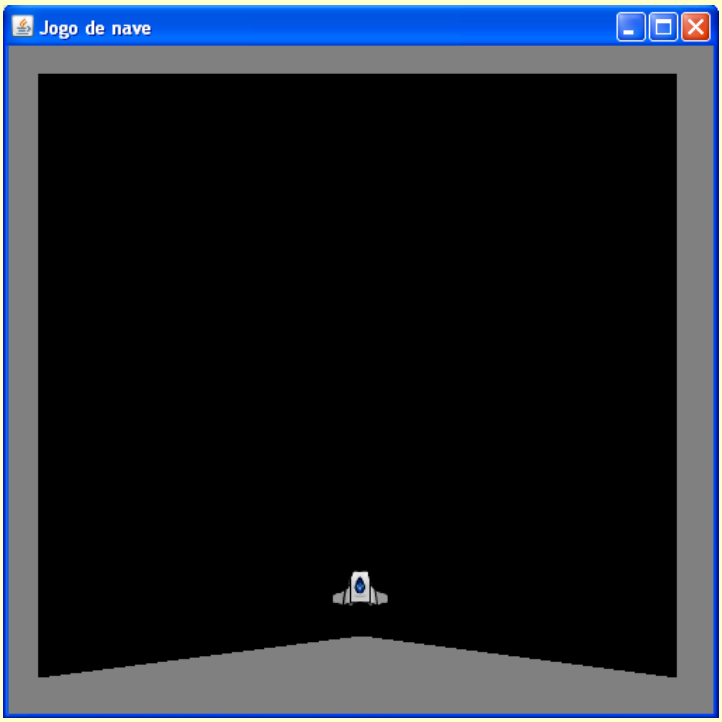
Terceiro, no método paint, apresenta a imagem

```
fundo.drawImage(Nave, 80, 100, 50, 50, this);
```

Código completo até o momento:

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;
import javax.swing.*;
import java.awt.Toolkit;

public class Jogo extends Jframe
{
	private Image Nave;

	public Jogo()
	{
		super("Jogo de nave");

		//Muda a cor de fundo para preto
		getContentPane().setBackground(Color.black);

		Nave = Toolkit.getDefaultToolkit().getImage("nav_0.JPG");

		//Muda o tamanho da janela
		setSize(510,510);

		//Deixa a janela visivel
		setVisible(true);
	}
	public void paint(Graphics fundo)
	{
		int mt_x[] = {0, 510, 510, 0, 0, 25, 480, 480, 255, 25, 25};
		int mt_y[] = {29, 29, 510, 510, 29, 50, 50, 480, 450, 480, 50};
		fundo.setColor(Color.GRAY);

		super.paint(fundo);
		fundo.fillPolygon(mt_x,mt_y,11);
		fundo.drawImage(Nave, 230, 400, 50, 50, this);
	}
	public static void main(String[] args)
	{
		Jogo p = new Jogo();
		p.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

Resultado da tela



Parte 2 do usuário

Agora que a nave foi importada para dentro do jogo, seria interessante fazer com que ela tenha uma pequena animação, como por exemplo, um fogo saindo dela. Isto é o que será abordado agora.

Primeiro deve-se entender como são feitas animações.
Animações nada mais são do que uma seqüência de imagens que são alteradas dentro de um intervalo de tempo dando uma ilusão de movimento.

Baseando-se neste conceito, será montada a animação de fogo saindo da nave.

Como é uma seqüência de imagens, nada melhor do que utilizar matriz. Assim, o segundo ponto a ser feito é alterar o objeto nave do tipo image para uma matriz nave do tipo image.

Antes:

```
private Image Nave;
```

Agora:

```
private Image Nave[];
```

Deve-se instanciar a matriz, acrescentando dentro do método contrutor

```
Nave = new Image[4];
```

E passar os nomes das imagens em sequência:

```
Nave[0] = Toolkit.getDefaultToolkit().getImage("nav_0.JPG");
Nave[1] = Toolkit.getDefaultToolkit().getImage("nav_1.JPG");
Nave[2] = Toolkit.getDefaultToolkit().getImage("nav_2.JPG");
Nave[3] = Toolkit.getDefaultToolkit().getImage("nav_3.JPG");
```

No método paint deve-se alterar Nave para Nave[a], sendo a o índice

```
fundo.drawImage(Nave[a], 230, 400, 50, 50, this);
```

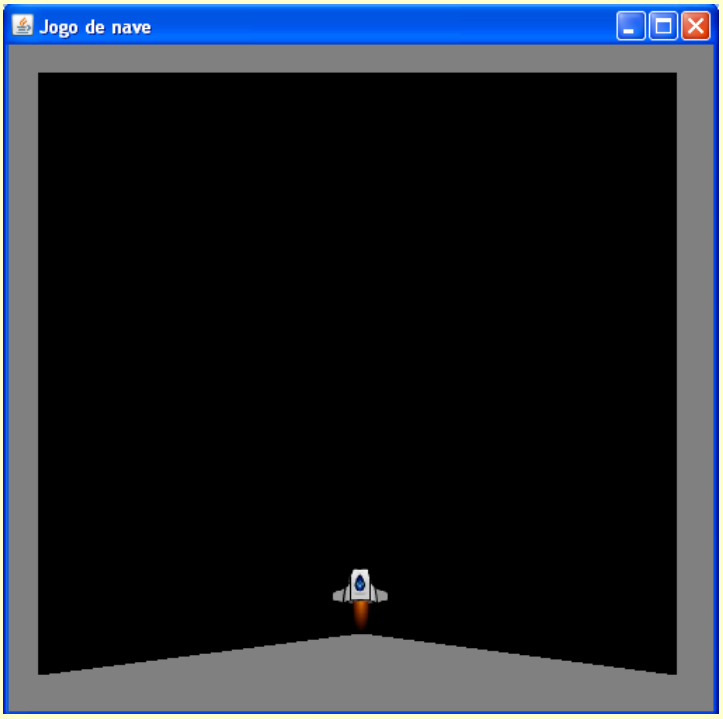
Como deve-se varia o índice para que mude as imagens, é necessário então um loop. Neste caso, será utilizado o FOR e a cada interação é necessário um pequeno atraso. Para o atraso, será apresentado uma primeira opção, o uso de Thread.sleep(x), do qual permite um atraso em termos de milissegundos, indicado por X. Este atraso irá determinar a velocidade do do jogo.

Para o uso de um Thread, é necessário o Try/Cath() que nada mais é do que uma forma de segurança caso ocorra algum erro, possibilitando o tratamento deste. Logo após, o uso do método repaint() que permitirá o recarregamento do método paint(), ficando em um loop infinito.

```
for(int a = 0; a <= 3; a++)
{
   fundo.drawImage(Nave[a], 230, 400, 50, 50, this);
   try
   {
      Thread.sleep(50);
   } catch (InterruptedException e)
   {
      e.printStackTrace();
   }
}
repaint();
```

Eis o resultado da tela:

Resultado da tela



O código até o momento é apresentado na página seguinte.

Código completo até o momento:

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;
import javax.swing.*;
import java.awt.Toolkit;

public class Jogo extends JFrame
{
	private Image Nave[];
	int i = 0;

	public Jogo()
	{
		super("Jogo de nave");

		Nave = new Image[4];

		//Muda a cor de fundo para preto
		getContentPane().setBackground(Color.black);

		Nave[0] = Toolkit.getDefaultToolkit().getImage("nav_0.JPG");
		Nave[1] = Toolkit.getDefaultToolkit().getImage("nav_1.JPG");
		Nave[2] = Toolkit.getDefaultToolkit().getImage("nav_2.JPG");
		Nave[3] = Toolkit.getDefaultToolkit().getImage("nav_3.JPG");

		//Muda o tamanho da janela
		setSize(510,510);

		//Deixa a janela visivel
		setVisible(true);
	}

	public void paint(Graphics fundo)
	{
		int mt_x[] = {0, 510, 510, 0, 0, 25, 480, 480, 255, 25, 25};
		int mt_y[] = {29, 29, 510, 510, 29, 50, 50, 480, 450, 480, 50};
		fundo.setColor(Color.GRAY);

		super.paint(fundo);
		fundo.fillPolygon(mt_x,mt_y,11);

		for(int a = 0; a <= 3; a++)
		{
			fundo.drawImage(Nave[a], 230, 400, 50, 50, this);
			try
			{
				Thread.sleep(20);
			} catch (InterruptedException e)
			{
				e.printStackTrace();
			}
		}
		repaint();
	}
	public static void main(String[] args)
	{
		Jogo p = new Jogo();
		p.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

Parte 3 do usuário

Agora é necessária a interação do usuário com a nave a partir do teclado, modificando assim, as posições da nave na tela.

Para isso, basta utilizar o método addKeyListenner() e passar como parâmetro um objeto do tipo keyListener

```
addKeyListener
(new KeyListener()
```

O objeto do tipo KeyListener possui 3 métodos:

```
1: keyPressed(KeyEvent a)
2: keyTyped(KeyEvent b)
3: keyReleased(KeyEvent e)
```

O primeiro é ativado assim que o usuário pressionar qualquer tecla do teclado.
O segundo é ativado assim que o usuário pressionar alguma tecla que não seja de ação, como: setas direcionais, home, end, page up, page down, num lock, print screen, scrool lock, caps lock, pause e teclas de função f1, f2, etc).

A lógica utilizada é a mesma do menu móvel já dita em aula.

Parte 1/3 do código até o momento:

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;

import javax.swing.*;
import java.awt.Toolkit;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

public class Jogo extends JFrame
{
   private Image Nave[];
   int i = 0;
   int x1=230,
       x2=20,
       y1=400,
       y2=20;

   public Jogo()
   {
      super("Jogo de nave");

      Nave = new Image[4];

      //Muda a cor de fundo para preto
      getContentPane().setBackground(Color.black);

      Nave[0] = Toolkit.getDefaultToolkit().getImage("nav_0.JPG");
      Nave[1] = Toolkit.getDefaultToolkit().getImage("nav_1.JPG");
      Nave[2] = Toolkit.getDefaultToolkit().getImage("nav_2.JPG");
      Nave[3] = Toolkit.getDefaultToolkit().getImage("nav_3.JPG");

      //Muda o tamanho da janela
      setSize(510,510);
      //Deixa a janela visivel
      setVisible(true);
```

Parte 2/3 do código até o momento:

```
    addKeyListener
    (
          new KeyListener()
          {
                //Método disparado assim que pressiona a tecla
                public void keyPressed(KeyEvent a)
                {
                      int code = a.getKeyCode();
                      switch(code)
                      {
                      //Move o objeto para esquerda
                      case 37:
                            if(x1!=40)
                            {
                                  x1=x1-10;
                                  repaint();
                            }
                            break;

                      //Move o objeto para cima
                      case 38:
                            if(y1!=60)
                            {
                                  y1=y1-10;
                                  repaint();
                            }
                            break;

                            //Move o objeto para direita
                      case 39:

                            if(x1!=420)
                            {
                      //muda a posição X do quadrado na tela
                                  x1=x1+10;
                                  repaint();
                            }
                            break;

                      //Move o objeto para baixo
                      case 40:
                            if(y1!=400)
                            {
                                  y1=y1+10;
                                  repaint();
                            }
                            break;

                      default:
                      System.out.println("Codigo da Tecla é:" + code);
                      break;
                      }
                }
                public void keyTyped(KeyEvent b) { }

                //Método disparado assim que solta a tecla
                public void keyReleased(KeyEvent e) { }
          }
      );
}
```

Parte 3/3 do código até o momento:

```
	public void paint(Graphics fundo)
	{
		int mt_x[] = {0, 510, 510, 0, 0, 25, 480, 480, 255, 25, 25};
		int mt_y[] = {29, 29, 510, 510, 29, 50, 50, 480, 450, 480, 50};
		fundo.setColor(Color.GRAY);

		super.paint(fundo);
		fundo.fillPolygon(mt_x,mt_y,11);

		for(int a = 0; a <= 3; a++)
		{
			fundo.drawImage(Nave[a], x1, y1, 50, 50, this);
			try
			{
				Thread.sleep(20);
			} catch (InterruptedException e)
			{
				e.printStackTrace();
			}
		}
		repaint();
	}

	public static void main(String[] args)
	{
		Jogo p = new Jogo();
		p.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

O usuário ao pressionar as teclas direcionais, perceberá que a nave se desloca pela tela, mas não ultrapassando os limites impostos pelas condições:

```
if(x1!=40) //limita o movimento para a esquerda
if(y1!=60) //limita o movimento para cima
if(x1!=420) //limita o movimento para a direita
if(y1!=400) //limita o movimento para baixo
```

Os métodos keyTyped e Keyreleased não serão utilizados, mas dever ser declarados.
Como o resultado na tela não muda, mas apenas é mudada através da ação do usuário, não será apresentado.

Agora é o momento de desenhar a nave inimiga e os tiros

Parte 1 da nave inimiga

Para desenho da nave inimiga foi utilizado

```
private Image Inimigo[];
```

e para o tiro do usuário

```
private Image Tiro;
```

Foram declarados as variáveis pos_tiro_x e y para indicarem a posição do tiro e a variável ativa_tiro que é habilitada quando o usuário pressiona o botão de espaço com código 32.

```
private int pos_tiro_x = 0;
private int pos_tiro_y = 0;
private boolean ativa_tiro = false;
```

Foram passados os seguintes nomes das imagens tanto para o inimigo quanto para o tiro:

```
Inimigo[0] = Toolkit.getDefaultToolkit().getImage("inimiga_0.JPG");
Inimigo[1] = Toolkit.getDefaultToolkit().getImage("inimiga_1.JPG");
Inimigo[2] = Toolkit.getDefaultToolkit().getImage("inimiga_2.JPG");
Inimigo[3] = Toolkit.getDefaultToolkit().getImage("inimiga_3.JPG");

Tiro = Toolkit.getDefaultToolkit().getImage("fogo.JPG");
```

Foi acrescentado dentro do switch a condição da tecla de espaço pressionado, isto é o código 32

```
//Ativa o Tiro
case 32:
      ativa_tiro = true;
      pos_tiro_x = x1+15;
      pos_tiro_y = y1;
      repaint();
break;
```

Acrescentado a condição para desativar o tiro caso o mesmo já se encontre em uma posição fora da tela visivel

```
if (pos_tiro_y <= 10)
   ativa_tiro = false;
```

Acrescentado ainda a condição para poder mover o tiro e apresentá-lo na tela, caso ativa_tiro seja true.

```
//tiro
if (ativa_tiro == true )
{
   pos_tiro_y-=10;
   fundo.drawImage(Tiro, pos_tiro_x, pos_tiro_y, 20, 20, this);
}
```

Foi aperfeiçoado o efeito do fogo com o uso de dois FOR, sendo um para lançar o fogo e outro para recolher

```
//Lança o fogo
for(a = 0; a <= 3; a++)
{
   fundo.drawImage(Nave[a], x1, y1, 50, 50, this);
   //desenha nave inimiga
   fundo.drawImage(Inimigo[a], 100, 50, 50, 50, this);
   try {
      Thread.sleep(20);
   } catch (InterruptedException e) {
      e.printStackTrace();
   }
}
//Recolhe o fogo
for(a = 3; a >= 0; a--)
{
   //desenha nave usuário
   fundo.drawImage(Nave[a], x1, y1, 50, 50, this);
   //desenha nave inimiga
   fundo.drawImage(Inimigo[a], 100, 50, 50, 50, this);
   try{
      Thread.sleep(20);
   } catch (InterruptedException e) {
      e.printStackTrace();
   }
}
```

Eis o código completo:

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;
import javax.swing.JFrame;
import javax.swing.Timer;
import java.awt.Toolkit;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

public class Jogo extends JFrame
{
   private Image Nave[];
   private Image Inimigo[];
   private Image Tiro;

   private Timer tempo;
   private int i = 0;
   private int pos_tiro_x = 0;
   private int pos_tiro_y = 0;
   private boolean ativa_tiro = false;

   int x1=230,
       x2=20,
       y1=400,
       y2=20;

   public Jogo()
   {
      super("Jogo de nave");

      Nave = new Image[4];
      Inimigo = new Image[4];

      //Muda a cor de fundo para preto
      getContentPane().setBackground(Color.black);

      Nave[0] = Toolkit.getDefaultToolkit().getImage("nav_0.JPG");
      Nave[1] = Toolkit.getDefaultToolkit().getImage("nav_1.JPG");
      Nave[2] = Toolkit.getDefaultToolkit().getImage("nav_2.JPG");
      Nave[3] = Toolkit.getDefaultToolkit().getImage("nav_3.JPG");

      Inimigo[0] = Toolkit.getDefaultToolkit().getImage("inimiga_0.JPG");
      Inimigo[1] = Toolkit.getDefaultToolkit().getImage("inimiga_1.JPG");
      Inimigo[2] = Toolkit.getDefaultToolkit().getImage("inimiga_2.JPG");
      Inimigo[3] = Toolkit.getDefaultToolkit().getImage("inimiga_3.JPG");

      Tiro = Toolkit.getDefaultToolkit().getImage("fogo.JPG");

      //Muda o tamanho da janela
      setSize(510,510);

      //Deixa a janela visivel
      setVisible(true);
```

```
addKeyListener
(
   new KeyListener()
   {
      //Método disparado assim que pressiona a tecla
      public void keyPressed(KeyEvent a)
      {
         int code = a.getKeyCode();
         switch(code)
         {
         //Ativa o Tiro
            case 32:
               ativa_tiro = true;
               pos_tiro_x = x1+15;
               pos_tiro_y = y1;
               repaint();
            break;
         //Move o objeto para esquerda
            case 37:
               if(x1!=40)
               {
                  x1=x1-10;
                  repaint();
               }
               else
               repaint();
            break;
         //Move o objeto para cima
           case 38:
               if(y1!=60)
               {
                  y1=y1-10;
                  repaint();
               }
               else
               repaint();
             break;
         //Move o objeto para direita
           case 39:
               if(x1!=420)
               {
         //muda a posição X do quadrado na tela
                  x1=x1+10;
                  repaint();
               }
               else
               repaint();
            break;
         //Move o objeto para baixo
           case 40:
               if(y1!=400)
               {
                  y1=y1+10;
                  repaint();
               }
               else
               repaint();
            break;
           default:
              System.out.println("Codigo da Tecla é:" + code);
           break;
         }
   }
   public void keyTyped(KeyEvent b) { }
   //Método disparado assim que solta a tecla
   public void keyReleased(KeyEvent e) { }
   }
);
}
```

```
	public void paint(Graphics fundo)
	{
		int mt_x[] = {0, 510, 510, 0, 0, 25, 480, 480, 255, 25, 25};
		int mt_y[] = {29, 29, 510, 510, 29, 50, 50, 480, 450, 480, 50};
		int a;

		fundo.setColor(Color.GRAY);

		super.paint(fundo);

		if (pos_tiro_y <= 10)
		ativa_tiro = false;

		//tiro
		if (ativa_tiro == true )
		{
			pos_tiro_y-=10;
		   fundo.drawImage(Tiro, pos_tiro_x, pos_tiro_y, 20, 20, this);
		}

		fundo.fillPolygon(mt_x,mt_y,11);

	  //Lança o fogo
		for(a = 0; a <= 3; a++)
		{

			fundo.drawImage(Nave[a], x1, y1, 50, 50, this);
		//desenha nave inimiga
			fundo.drawImage(Inimigo[a], 100, 50, 50, 50, this);

			try{
				Thread.sleep(20);
			} catch (InterruptedException e){
				e.printStackTrace();
			}
		}

		//Recolhe o fogo
		for(a = 3; a >= 0; a--)
		{
		//desenha nave usuário
			fundo.drawImage(Nave[a], x1, y1, 50, 50, this);
		//desenha nave inimiga
			fundo.drawImage(Inimigo[a], 100, 50, 50, 50, this);

			try{
				Thread.sleep(20);
			} catch (InterruptedException e){
				e.printStackTrace();
			}
		}
		repaint();
	}

	public static void main(String[] args)
	{
		Jogo p = new Jogo();
		p.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

Até aqui observa-se que o jogo precisa ser otimizado.

Um opção para otimizar o jogo é repintar a tela apenas quando for extremamente necessário, como por exemplo quando o usuário pressionar as teclas direcionais para mover a nave. Mas existem muitas outras que ficam a critério do programador.