Se aventure nessa

Introdução ao Desenvolvimento de Jogos com Java

Francisco A. S. Rodrigues

Clube dos Geeks

SE AVENTURE NESSA

Introdução ao Desenvolvimento de Jogos com Java

Francisco de A. S. Rodrigues

Juazeiro do Norte – CE, BRA, Julho de 2014.

Agradecimentos

Quero agradecer a Deus por tudo o que tens feito em minha vida.

Este livro é dedicado a toda a minha família e amigos que estiveram comigo e me apoiaram, e principalmente a Tamires, por ter dado todo o amor, que é uma pessoa muito especial em minha vida e sempre esteve ao meu lado me apoiando e me compreendendo em todos os momentos. Te amo.

Sobre

Francisco de Assis de Souza Rodrigues, Servo de Deus, Pós-Graduando em Docência do Ensino Superior - IDJ, Tecnólogo em Análise e Desenvolvimento de Sistemas - FALS, fundador do site Clube do Geeks, sócio e Back-End Developer na empresa GrowUp; ênfase em JavaEE, JavaSE, Java Android, Arduíno, Robótica, C/C++, entusiasta Linux e toca violão nas horas vagas.

Siga-me: FaceBook

Sumário

1 - Introdução

O clássico Super Mario Bros, foi praticamente o primeiro jogo que joguei em meados 1996 na plataforma Nintendo, lançado em 1985 foi o primeiro jogo com rolagem lateral, recurso conhecido como *slide-scroling*. O jogo é o mais vendido na história dos vídeos games e inspirou incontáveis imitações que ajudaram a fixar esse estilo. O jogador controlava o principal protagonista da serie, Mario. O objetivo do jogo é percorrer o Reino do Cogumelo, sobreviver as forças do principal vilão Bowser, e salvar a princesa Peach e seu reino do domínio dos Koop Troopas.

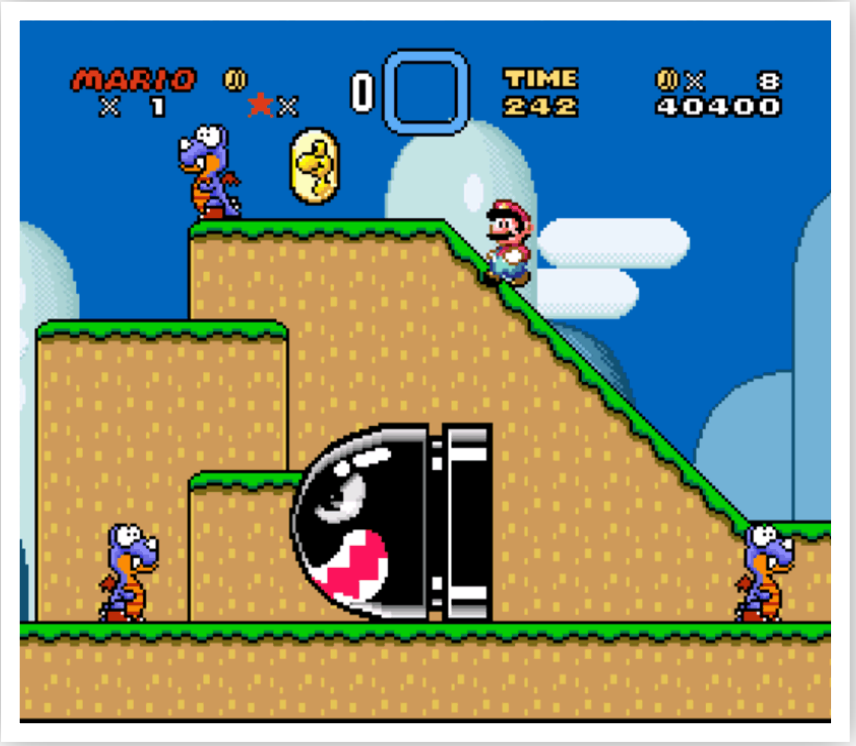

Figura 01

Um dos meus jogos favoritos logo em seguida foi o Aero Fighters conhecido como Sonic Wings no Japão, um jogo eletrônico de nave estilo Shot em up com rolagem vertical. Lançado originalmente com nome "Sonic Wings" para fliperama em 1992 que foi alterado em 1994 quando lançado para a plataforma SNES. Nesse clássico você é um piloto de avião, e com uma nave milita, tem o desafio de acabar com as bases inimigas. Detone as naves, pegue novas armas e tiros e conquiste grandes territórios.

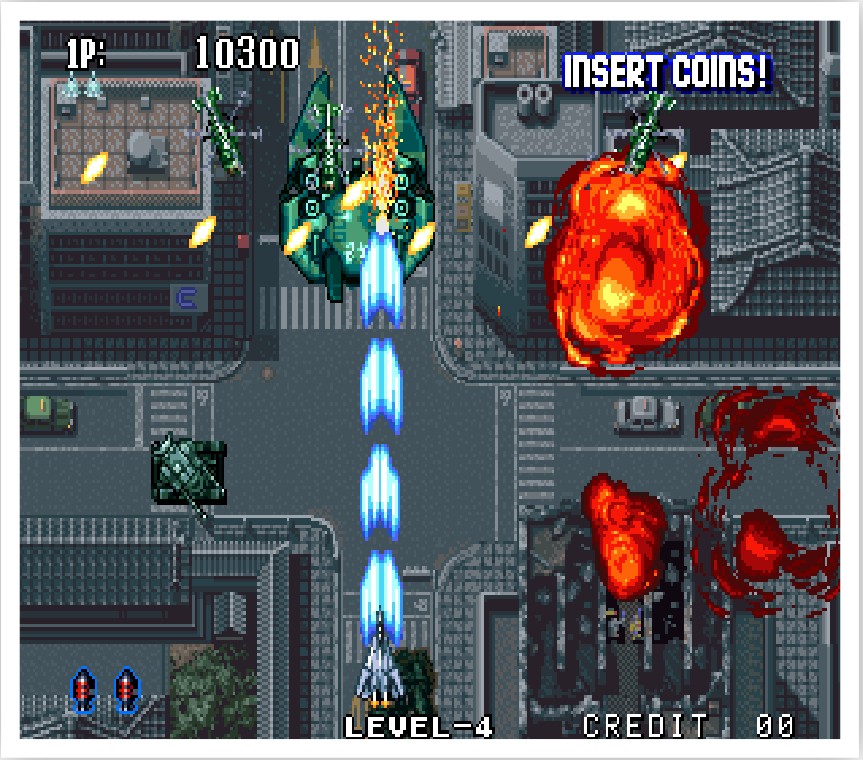

Figura 02

1.1 - Objetivo

Inspirar e motivar estudantes novatos ou até mesmo veteranos programadores, para programação voltada a desenvolvimento de jogos, ver que é possível criar bons jogos com linguagem de alto nível com Java e entender a mecânica básica de um jogo.

1.2 - Protótipo

Figura 03

Usaremos o jogo Aero Fighters como inspiração para o protótipo que iremos desenvolver jogo chamdo Champs da Galáxia, um gênero espacial, mostra uma batalha galáctica entre diversas especeis; controle sua nave, pegue o máximo de power-up "potencia" para sua arma e destrua tudo o que vier em sua frente, você foi escolhido para a missão de combater o exercito do General Grong, evitando assim que ele governe e crie um caos no hiperespaço.

A história do jogo é o marco inicial, e o documento que é utilizado para projetar o jogo é conhecido como **Game Designer,** nele é que se escreve o projeto do jogo a ser desenvolvido, historia, personagens, cenários, jogador, inimigos, desafios, logica do jogo, em fim tudo o que constara no seu jogo, porem não é o foco principal desse livro e não iremos nos aprofundar nessa documentação.

Não iremos utilizar nenhum Framework para tratamento gráfico, usaremos uma *class gráfica* 2D que o java possuí e mesmo assim chegaremos a um jogo bem interessante, pois o objetivo é se familiarizar e ter conceitos básico para desenvolvimento de games, esses conceitos sempre estarão presentes, seja em jogos simples ou avançados.

1.3 - Visão Industrial

Programação é apenas uma parte do desenvolvimento de game. Empresas focadas em desenvolvimento de jogos possuem roteiristas para criar a história do game, designer para definir o melhor visual do jogo, profissionais de som para trilhas sonoras e efeitos, designer de interface para definir como seria a experiência do jogador no game, entre outros. Jogos como "Crisys" chega ater 650 pessoas em uma equipe, todos envolvidos para se alcançar o esperado sucesso.

Figura 04

2 - Configurações do Ambiente

A configuração do ambiente de desenvolvimento utilizado é: IDE Eclipse, pode-se ser quaisquer versão do eclipse, ou até mesmo se você preferir pode usar outras IDE como Netbeans ou IntelliJ Idea, JDK Java7 Run Time ou superior e o JRE.

Caso você ainda não tenha instalado você pode esta adquirindo em:

Baixar o Eclipse aqui.
Baixar o JDK e o JRE aqui.
Baixar JLayer aqui.
Baixar o projeto "código fonte" e imagens do jogo e o arquivo .jar executável aqui:

2.1 - Criando a Base do Jogo

Com o eclipse instalado e corretamente configurado em seu computador, já podemos começar. Crie um projeto com o nome *ChampsDaGalaxia*.

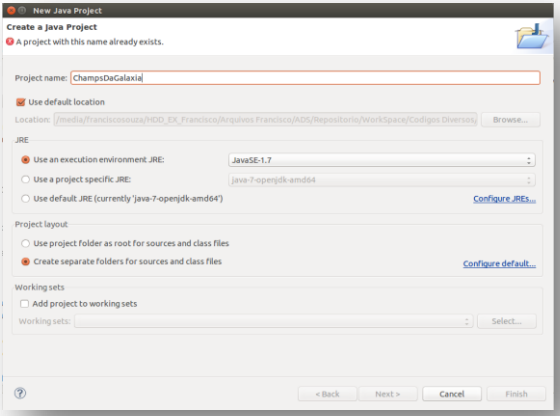

Figura 05

Na pasta *src* crie um pacote com o nome *br.com.game.app*

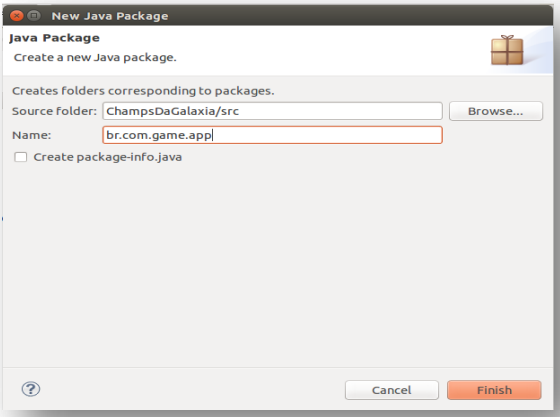

Figura 06

Dentro do pacote crie uma class com o nome *Game*.

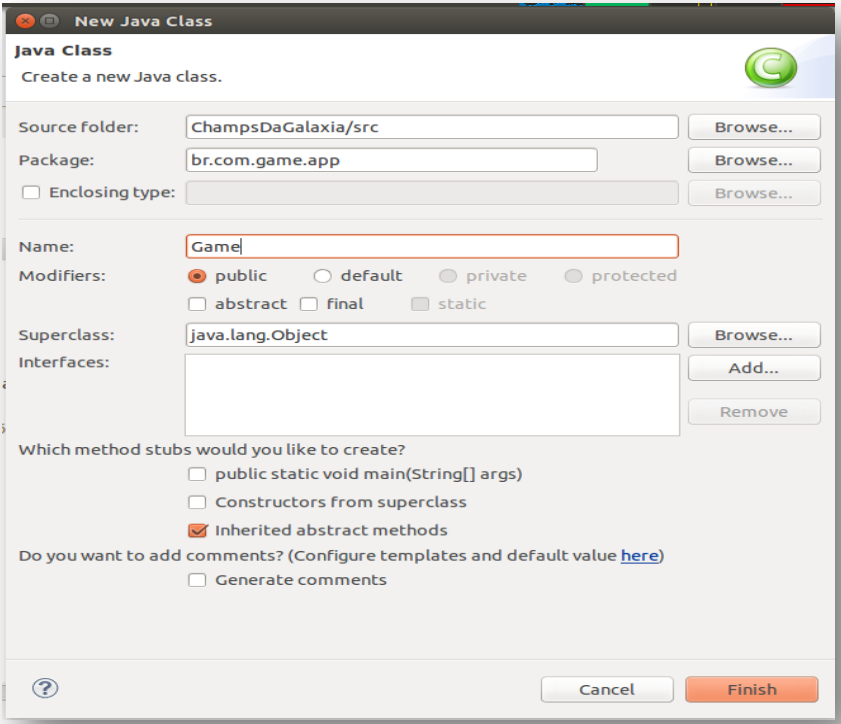

Figura 07

Agora podemos programar nessa *class* que foi criada. Ela deve herdar da *class JFrame*, componente do pacote swing que possui todos os componentes necessários para a construção da janela do jogo.

```
public class Game extends JFrame {

}
```

Agora criaremos os seguintes métodos e o construtor da class, na *class Game.java*; não citarei muitos detalhes pois o código está comentado:

O método *main* é o principal responsável pela execução da aplicação.

```
/**
 * Método principal - start a aplicação.
 * @param args
 */
public static void main(String args[]) {
        new Game();
}
```

Esse é o construtor da nossa class.

```
/**
 * Construtor.
 */
protected Game() {
        /* Chama o método componentes */
                componentes();
        }
```

Esse método que configura a nossa janela.

```
/**
 * Configura a Janela
 */
public void componentes() {
        /* Titulo da Janela */
        setTitle("Champs da Galáxia");
        /* Permite encerrar aplcação */
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        /* Define tamanho da janela */
        setSize(600, 600);
```

```
/* Instancia a janela no centro da tela */
setLocationRelativeTo(null);
/* Bloqueia redimensionamento da janela */
setResizable(false);
/* Define a janela visível */
setVisible(true);
}
```

Agora execute a aplicação e teremos uma janela como essa:

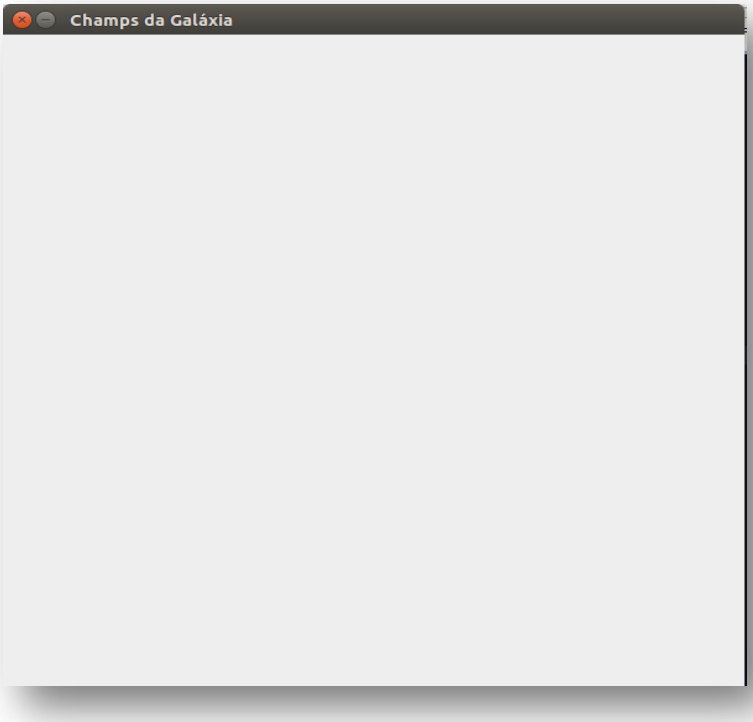

Figura 08

2.2 – Desenhando o Background

Podemos ver que ainda está muito longe de termos um jogo. Com o botão direito do mouse clique em cima do projeto **New >> Folder** crie uma pasta com o nome: *res*. *Nessa pasta ficam* todas as imagens e sons que utilizaremos no jogo.

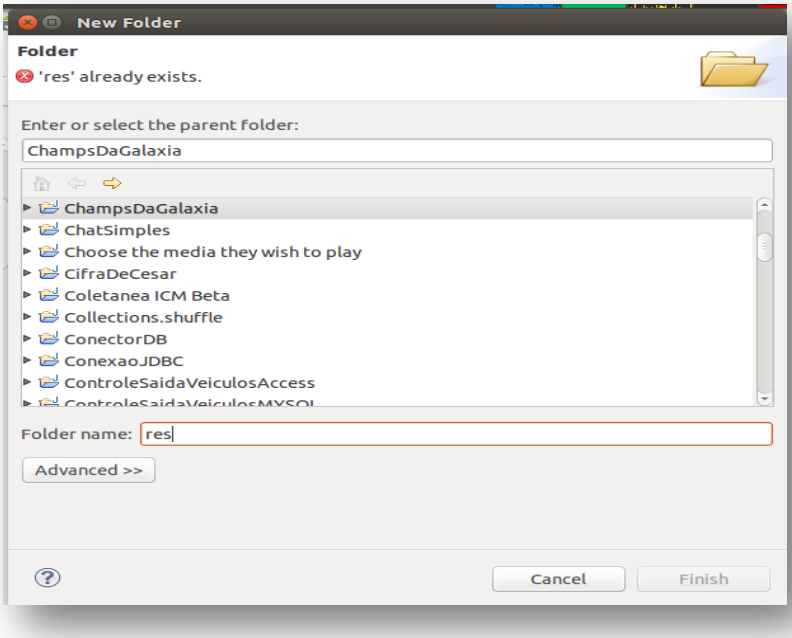

Figura 09

Pegue as imagens do jogo que você já deve ter baixado e adicione dentro dessa pasta juntamente com as subpastas. Agora no inicio da *class Game.java* criaremos os sequentes objetos:

- O objeto time da *class* Timer.java componente do pacote swing utilizaremos para manipulação de threads do jogo;

- O objeto fase que usaremos para acessar a classe Fase, uma *class* interna que iremos implementar;

- O objeto fundo da *class* Image.java componente do pacote swing utilizaremos para carregar uma imagem de fundo do cenário do jogo;

- O objeto *grafico* componente do pacote swing que usaremos para manipulas as imagens 2D no jogo.

```
private Timer timer;
private Fase fase;
private Image fundo;
private Graphics2D grafico;
```

Abaixo do método **componentes**(), criaremos uma *class* interna chamada *Listener ela serve para escutar o que acontece em um objeto e avisar a outro, ela* implementa *ActionListener, que possibilita a Listener escutar por uma ação, ou seja, essa class é um ouvinte que informa as determinadas ações dos objetos.*

.

```
private class Listener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
                }
        }
```

Abaixo da *class* interna *Listener*; criaremos mais uma *class* com o nome de *Fase* que herda de *JPanel*, que é um componente do pacote *swing* que tem como principal função servir de *conteiner* para outros componentes.

```
public class Fase extends JPanel {
```

```
private static final long serialVersionUID = 1L;

/* Define as dimensões do JPanel */
protected static final int ALTURA = 600;
protected static final int LARGURA = 600;

        protected Fase() {
                setDoubleBuffered(true);
        }

public void paint(Graphics g) {
                grafico = (Graphics2D) g;
                grafico.drawImage(fundo, 0, 0, null);
                g.dispose();
        }

        /**
         *
         * @return
         */
        public int getLar() {
                return LARGURA;
        }

        /**
         *
         * @return
         */
        public int getAlt() {
                return ALTURA;
        }
}
```

Crie um método inicializar *inicializar();* após o método *componentes*().

```
public void inicializar() {
        fase = new Fase();
        add(fase);
        ImageIcon referencia = new ImageIcon("res/fundoFase/cenario1.jpg");
        fundo = referencia.getImage();
}
```

Bem por fim no método *inicializar(),* adicione o seguinte trecho de código que é responsável por instanciar uma thread referenciando a *class* interna *Listener*.

```
/*Instância uma thread do jogo, com a class Listener*/
timer = new Timer(5, new Listener());
timer.start();/*inicia a thread*/
```

Chama o método *inicializar*() no construtor da *class Game.java*

```
/**
 * Construtor.
*/
protected Game() {
        componentes();
        inicializar();
}
```

Dentro do método *actionPerformed(ActionEvent e),* da *class Listener* adicione o seguinte código:

```
fase.repaint();
```

Agora execute novamente a aplicação e você terra uma janel

Figura 10

2.3 - Sprite

Simplificando o conceito de *sprites:* é uma imagem bidimensional, ou uma animação que é integrada em uma cena de um jogo, igual a aqueles bloquinhos de desenhos onde se desenha varias imagens estáticas e ao passar rapidamente as folhas dava a ilusão de movimento "animação do desenho", basicamente é dessa forma que também funciona no nosso jogo.

Figura 11

Para entendemos, podemos ver que a **figura 12** e uma única imagem, com 364px de largura e 62px de altura, porem iremos dividir por 7 que e a quantidade de naves que tem na imagem, sabendo que cada nave tem exatamente 52px de largura, quando implementamos no jogo passamos as coordenadas como um vetor de 7 posições, que vai de 0 até 6, porem temos que referenciar a posição inicial da imagem, que no nosso caso será a posição 0, e quando o usuário mover a nave devemos tratar as posições da imagem de acordo com o movimento, alternando as suas posições assim acontecera o movimento lateral da nave. Existe varias forma de realizar esses movimentos, mais adiante veremos uma for bem simples de realizar.

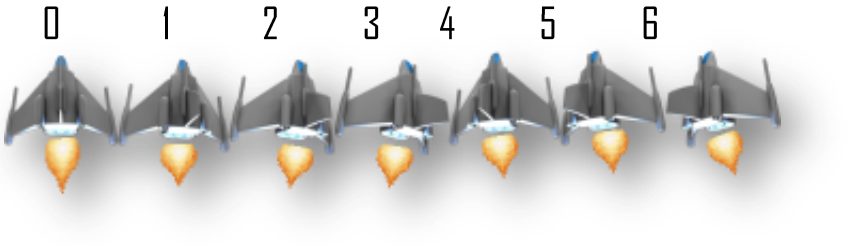

Figura 12

3 - Criando o Player

Crie um novo pacote: *br.com.game.app.player* e dentro do pacote crie uma nova *class* com o nome de Nave.

```
public class Nave {
        /* coordenadas da nave */
        private int x;
        private int y;
        private int dx;
        private int dy;

        /* posição da imagem no sprite */
        private int pos;

        /* Objeto imagem nave*/
        private Image imagem;

        /* status da nave */
        private boolean isVisivel;

        /* dimensões da imagem */
        private static final int LARGURA = 52;
        private static final int ALTURA = 62;

        public Nave() {
                /* objeto que referencia o local da imagem da nave */
                ImageIcon referencia = new
ImageIcon("res/player/nave.png");
                imagem = referencia.getImage();/* retorna a imagem */
                pos = 0;/* posição inicial da imagem */
                /* posição inicial da nave no jogo*/
                x = 300;
```

```
        y = 500;
        isVisivel = true;
}

/**
 * move a nave
 */
public void mover() {
        x += dx;
        y += dy;
}

/* ====== Métodos de acesso ====== */
public void setDx(int dx) {
        this.dx = dx;
}

public void setDy(int dy) {
        this.dy = dy;
}

public int getX() {
        return x;
}

public int getY() {
        return y;
}

public Image getImagem() {
        return imagem;
}
```

```
        public int getPos() {
                return pos;
        }

        public void setPos(int Pos) {
                pos = Pos;
        }

        public int getAlt() {
                return ALTURA;
        }

        public int getLar() {
                return LARGURA;
        }

        public boolean isVisivel() {
                return isVisivel;
        }

        public void setVisivel(boolean visivel) {
                isVisivel = visivel;
        }

}
```

3.1 – Desenhando o Player

No inicio da *class* Game.java criaremos um objeto para *class* Nave.java.

```
private Nave nave;
```

No método inicializar(); abaixo da linha de código *fundo = referencia.getImage();* instancie a *class* Nave.java com o seguinte trecho de código:

```
/* Instancia o objeto nave */
nave = new Nave();
```

Agora para desenhar a nave na tela temos que na *class* interna Fase, no método *pain(Graphics g);*, antes de *g.dispose();* adicionar código a seguinte:

```
/*Desenha Nave*/
grafico.drawImage(nave.getImagem(), nave.getX(), nave.getY(),
nave.getX() + nave.getLar(),
nave.getY() + nave.getAlt(),
1 + (nave.getPos() * nave.getLar()), 1,
1 + (nave.getPos() * nave.getLar()) + nave.getLar(),
nave.getAlt() + 1, null);
```

Agora execute e você terra uma tela igual a essa:

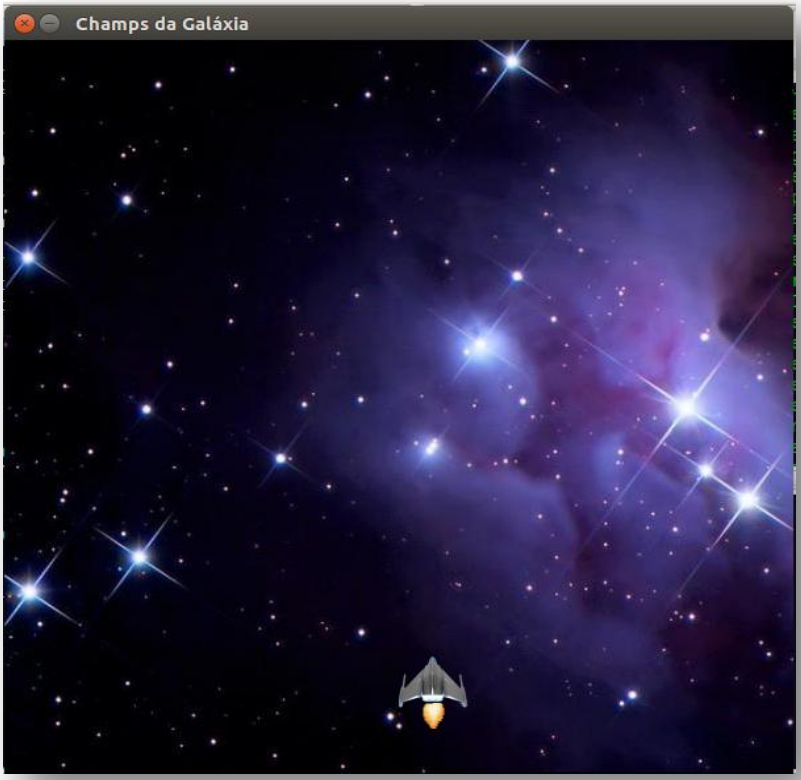

Figura 13

3.2 – Movendo o Player

Agora criaremos no final da *class* Game.java, uma *class* interna chamada de *Controle*, que herda de *KeyAdapter um* componente do pacote swing, e implementar os seus métodos de eventos para podemos escutar os comandos do usuário no teclado e realizar ações necessárias no jogo.

```
private class Controle extends KeyAdapter {
        /*Tecla pressionada*/
                public void keyPressed(KeyEvent tecla) {
                        int codigo = tecla.getKeyCode();
        /*Move nave para esquerda*/
                        if (codigo == KeyEvent.VK_LEFT) {
                                nave.setPos(4);
                                nave.setPos(5);
                                nave.setPos(6);
                                nave.setDx(-1);
                        }
        /*Move nave para direita*/
                        if (codigo == KeyEvent.VK_RIGHT) {
                                nave.setPos(1);
                                nave.setPos(2);
                                nave.setPos(3);
                                nave.setDx(1);
                        }
        /*Move nave para frente*/
                        if (codigo == KeyEvent.VK_UP) {
                                nave.setDy(-1);
                        }
        /*Move nave para traz*/
                        if (codigo == KeyEvent.VK_DOWN) {
                                nave.setDy(1);
                        }
```

```
        }
/*Tecla solta*/
        public void keyReleased(KeyEvent tecla) {
                int codigo = tecla.getKeyCode();
                if (codigo == KeyEvent.VK_LEFT) {
                        nave.setPos(6);
                        nave.setPos(5);
                        nave.setPos(4);
                        nave.setPos(0);
                        nave.setDx(0);
                }
                if (codigo == KeyEvent.VK_RIGHT) {
                        nave.setPos(3);
                        nave.setPos(2);
                        nave.setPos(1);
                        nave.setPos(0);
                        nave.setDx(0);
                }

                if (codigo == KeyEvent.VK_UP) {
                        nave.setDy(0);
                }
                if (codigo == KeyEvent.VK_DOWN) {
                        nave.setDy(0);
                }
        }
                }
```

Opa nossa nave não se move na tela, então para realizar os movimentos da nave temos que adicionar o seguinte trecho de código na *class* interna *Listener*; antes do método *fase.repaint();* que escuta as ações do usuário e informa para os outros objetos.

```
nave.mover();
```

Após adicionar o código a cima execute novamente a aplicação e terra a nave movendo na tela quando pressionar as teclas.

3.3 – Ajustes Finais do Player na Tela

Bem se você reparou, nossa nave esta saindo da janela, para que isso não ocorra temos que tratar o posicionamento da nave na janela limitando o seu espaço de deslocamento dentro da mesma, para isso adicione o seguinte código no método ***mover()*** na *class* Nave.java.

```
if (this.x < 1) {
        this.x = 1;
}
if (this.x > 560) {
        this.x = 560;
}
if (this.y < 30) {
        this.y = 30;
}
if (this.y > 480) {
        this.y = 480;
}
```

Execute novamente a aplicação e teste se a nave não sai mais da janela.

3.4 - Atirando

Agora criaremos um pacote *br.com.game.app.armas* e uma class chamada de *Laiser*:

```
public class Laiser {
        private int x, y;
        private Image imagem;
        private boolean isVisivel;
        private static final int VELOCIDADE = 2;
        private static final int ALTURA = 50;
        private static final int LARGURA = 20;

        public Tiro1(int x, int y) {
                this.x = x;
                this.y = y;
                ImageIcon referencia = new
ImageIcon("res/armas/laiser.png");
                imagem = referencia.getImage();
                isVisivel = true;
        }

        public void mover() {
                y -= VELOCIDADE;
        }

        public Image getImagem() {
                return imagem;
        }

        public int getX() {
                return x;
        }
```

```
	public int getY() {
		return y;
	}

	public int getAlt() {
		return ALTURA;
	}

	public int getLar() {
		return LARGURA;
	}

	public boolean isVisivel() {
		return isVisivel;
	}

	public void setVisivel(boolean visivel) {
		isVisivel = visivel;
	}
}
```

Agora no inicio da *class* Game.java iremos criar um objeto da *class* Laiser.java, do tipo lista para podermos ter acesso só disparo de laiser da nave.

```
private List<Laiser> laiser;
```

Bem, vamos ate o método inicializar e Instanciar nossa lista de disparo de laiser como o código a seguir:

Obs.: lembre-se de incluir após a instancia do objeto nave.

```
laiser = new ArrayList<Laiser>();
```

Agora para desenharmos os laiser na tela adicionaremos na *class* interna Fase, no método *paint(Graphics g)* antes de *g.dispose()* o seguinte código:

```
Laiser laiserTemp;
for (int i = 0; i < laiser.size(); i++) {
        laiserTemp = laiser.get(i);

grafico.drawImage(laiserTemp.getImagem(), laiserTemp.getX(),
laiserTemp.getY(),
laiserTemp.getX() + laiserTemp.getLar(),
laiserTemp.getY() + laiserTemp.getAlt(), 1, 1,
laiserTemp.getLar() + 1, laiserTemp.getAlt() + 1, null);
}
```

Para disparar o laiser devemos adicionar a função na *class* interna *Controle* no método *keyPressed(KeyEvent teclas)* o seguinte comando:

```
if (codigo == KeyEvent.VK_SPACE) {
        laiser.add(new Laiser (nave.getX() + 20, nave.getY() + -25));
}
```

E por fim devemos mover o laiser disparado na tela usando a *class* ouvinte interna *Listener* que como o implementando o código a seguir:

Obs.: implemente antes do código *nave.mover()*

```
Laiser laiserTemp;
for (int i = 0; i < laiser.size(); i++) {
        laiserTemp = laiser.get(i);

        if (laiserTemp.getY() > -10) {
                laiserTemp.mover();
        } else {
                laiser.remove(i);
        }
}
```

Agora execute a aplicação e dispare alguns laiser.

4 - Criando Inimigos

Já temos o player pronto, agora podemos adicionar alguns inimigos no jogo. Para isso criaremos um novo pacote com o nome *br.com.game.app.inimigo* e dentro desse pacote uma *class* como o nome de *Inimigo*.

```
public class Inimigo {
        private double x, y;
        private Image imagem;
        private boolean isVisivel;
        private static final int LARGURA = 50, ALTURA = 52;
        private static int contador = 0;
        private ImageIcon referencia;
        private static final double VELOCIDADE = 0.5;

        public Inimigos(int x, int y) {
                this.x = x;
                this.y = y;

                if (contador++ % 2 == 0) {
                        referencia = new
ImageIcon("res/inimigos/asteroid32.png");

                } else if (contador++ % 3 == 0) {
                        referencia = new
ImageIcon("res/inimigos/DeathFighter1.png");

                } else if (contador++ % 4 == 1) {
                        referencia = new
ImageIcon("res/inimigos/spikey enemy.png");
                } else {
                referencia = new ImageIcon("res/inimigos/wings.png");
                }
```

```
            imagem = referencia.getImage();
            isVisivel = true;
    }

    public void mover() {
            y += VELOCIDADE;
    }

    public int getAlt() {
            return ALTURA;
    }

    public int getLar() {
            return LARGURA;
    }

    public Image getImagem() {
            return imagem;
    }


    public double getX() {
            return x;
    }

    public double getY() {
            return y;
    }

    public void setY(double y) {
            this.y = y;
    }
```

```
        public boolean isVisivel() {
                return isVisivel;
        }

        public void setVisivel(boolean visivel) {
                isVisivel = visivel;
        }

}
```

4.1 – Desenhando Inimigo

Criada a *class Inimigo.java* já podemos adicionar inimigos no jogo realizando algumas incorporações de código na *class Game.java*.

Conforme fizemos anteriormente iremos criar uma um objeto do tipo lista da *class* de *Inimigo.java*

```
private List<Inimigo> inimigos;
```

Criaremos uma *class* interna chamada *NovoInimigo* que implementa *ActionListener* com o metodo *actionPerformaed(Action e)* para escutar as ações dos objetos e criarmos novos inimigos na tela. Usando uma função da *class* java.leng chamada de *Math.random()*, estaremos criando os inimigos em posições aleatórias na tela do jogo.

Obs.: Adicione o trecho de código antes da *class* Controle.

```
private class NovoInimigo implements ActionListener {
        public void actionPerformed(ActionEvent e) {
                inimigos.add(new Inimigos(1 + (int) (550 * Math.random()), -
80));
                }
        }
```

No inicio da *class Game.java* iremos criar um objeto para criarmos uma *thread* de inimigos.

```
private Timer novosInimigos;
```

No método *inicializar* da *class Game.java instancie* a lista de inimigos.

```
inimigos = new ArrayList<Inimigo>();
```

E start a *thread* da *class* interna *NovoInimigo*.

```
novosInimigos = new Timer(900, new NovoInimigo());
        novosInimigos.start();
```

Obs.: o parâmetro inteiro 900 e destinado ao tempo com que temos que espera para criar inimigo.

Agora iremos desenhar os inimigos na tela, adicionando o código seguinte na *class interna Fase* no método *paint(Graphics g)*, antes do código *g.dispose()*.

```
Inimigo iniTemp;
    for (int i = 0; i < inimigos.size(); i++) {
        iniTemp = inimigos.get(i);

        grafico.drawImage(iniTemp.getImagem(), (int) iniTemp.getX(),
            (int) iniTemp.getY(),
            (int) iniTemp.getX() + iniTemp.getLar(),
            (int) iniTemp.getY() + iniTemp.getAlt(), 1, 1,
            iniTemp.getLar() + 1, iniTemp.getAlt() + 1, null);
    }
```

Na *class Listener* após o código mover laise implemente o seguinte:

```
Inimigo iniTemp;
    for (int i = 0; i < inimigos.size(); i++) {
        iniTemp = inimigos.get(i);

        if (iniTemp.getY() < fase.getAlt()) {
            iniTemp.mover();
```

```
            } else {
                    iniTemp.setY(-iniTemp.getAlt());
            }
    }
```

Bem já podemos executar novamente nossas aplicação e ver como esta funcionando.

5 - Detectando Colisões

Podemos ver que quando disparamos o laiser da nave, não acontece nada, eles passam direto e rão fazem nada como os inimigos e a nave não colide com os inimigos,, então vamos implementar identificação de colisão entre objetos e ações que devem realizar quando ocorrer colisão.

Inicialmente criaremos algumas variáveis para manipular as dimensões dos objetos nave e inimigo para identificarmos as colisões.

```
private int sx1, sx2, sy1, sy2;
private int dx1, dx2, dy1, dy2;
```

Criaremos um objeto inimigo tempo da *class* Inimigo.java.

```
private Inimigo iniTemp;
```

Agora criaremos um método chamado *ColisaoNave(),* esse método verifica se a nave "player" colidi-o com qualquer inimigo ou seja, sera responsável por capturar as dimensões da imagem nave, percorrer toda a lista de inimigos e capturar as dimensões da imagem inimigo, realizar os calculo necessários que verifica se os objetos colidi-o ou não, se houver colisão retira o inimigo interceptado do jogo.

```
private void ColisaoNave() {
        sx1 = nave.getX();
        sx2 = nave.getX()  + nave.getLar();
        sy1 = nave.getY();
        sy2 = nave.getY() + nave.getAlt();
```

```
for (int i = 0; i < inimigos.size(); i++) {
        iniTemp = inimigos.get(i);
        dx1 = (int) iniTemp.getX();
        dx2 = (int) iniTemp.getX() + iniTemp.getLar();
        dy1 = (int) iniTemp.getY();
        dy2 = (int) iniTemp.getY() + iniTemp.getAlt();

        if (sx1 < dx2 && sx2 > dx1 && sy1 < dy2 && sy2 > dy1) {
                for (int n = 0; n < inimigos.size(); n++) {
                        iniTemp = inimigos.get(n);
                        dx1 = (int) iniTemp.getX();
                        dx2 = (int) iniTemp.getX() + iniTemp.getLar();
                        dy1 = (int) iniTemp.getY();
                        dy2 = (int) iniTemp.getY() + iniTemp.getAlt();

                        if (sx1 < dx2 && sx2 > dx1 && sy1 < dy2 && sy2 > dy1) {
                                inimigos.remove(i);
                        }
                }
        }
}
}
```

Da mesma foram que implementaremos o método anterior *ColisaoNave*() iremos criar agora um novo método chamado ColisaoLaiser(), esse método irar verificar se ouve colisão de algum laiser disparado com algum inimigo, quando houver colisão entre esses dois objetos era remove-los da tela.

```
private void ColisaoLaiser() {
        for (int i = 0; i < inimigos.size(); i++) {
                iniTemp = inimigos.get(i);
                dx1 = (int) iniTemp.getX();
                dx2 = (int) iniTemp.getX() + iniTemp.getLar();
```

```
                dy1 = (int) iniTemp.getY();
                dy2 = (int) iniTemp.getY() + iniTemp.getAlt();

                Laiser tiroTemp;
                for (int j = 0; j < laiser.size(); j++) {
                        tiroTemp = laiser.get(j);

                        sx1 = tiroTemp.getX();
                        sx2 = tiroTemp.getX() + tiroTemp.getLar();
                        sy1 = tiroTemp.getY();
                        sy2 = tiroTemp.getY() + tiroTemp.getAlt();

                        if (sx1 < dx2 && sx2 > dx1 && sy1 < dy2 && sy2 > dy1) {
                                laiser.remove(j);
                                inimigos.remove(i);
                        }
                }
        }
```

Bem para que ocorra a verificação das colisões devemos chamar os métodos na *class* interna *Listener* os métodos a seguir, que estarão em constante analise.

```
ColisaoNave();
ColisaoLaiser();
```

Agora Execute mais uma vez aplicação e verifique se quando o player nave colide com o inimigo, o inimigo e retirado da tela e se quando o laiser disparado colide com o inimigo o laise e o inimigo e retirados da tela.

5.1 - Mostrando Resultados e Criando Efeitos

Bem podemos ver que esta tudo ocorrendo corretamente, porem queremos que quando o laiser interceptar o inimigo ocorra um explosão e o mesmo se a nave interceptar o inimigo, então para isso Inicialmente iremos criar um novo pacote como o nome de *br.com.game.app.explosao* e dentro desse pacote um *class* chamada *Explosao* e implemente o código a seguir:

```
public class Explosao implements ActionListener {
        private int cont;
        private int x, y;
        private Image imagem;
        private static final int LARGURA = 52, ALTURA = 64;

        public Explosao(int x, int y) {
                ImageIcon referencia = new
ImageIcon("res/explosao/explosao.png");
                imagem = referencia.getImage();
                this.x = x;
                this.y = y;
                cont = 0;
        }

        public void actionPerformed(ActionEvent e) {
                cont++;
        }

        public int getCont() {
                return cont;
        }
```

```
        public int getX() {
                return x;
        }

        public int getY() {
                return y;
        }

        public int getAlt() {
                return ALTURA;
        }

        public int getLar() {
                return LARGURA;
        }

        public Image getImagem() {
                return imagem;
        }
}
```

Com a class Explosao.java já implementada iremos criar uma lista Timer e uma Lista de Explosões no *class Game.java, para manipular as explosões.*

```
private List<Timer> tempos;
private List<Explosao> explosoes;
```

Navegue ate o método *inicializar*() e instancie a lista de tempos e explosões.

```
tempos = new ArrayList<Timer>();
explosoes = new ArrayList<Explosao>();
```

Va ate o método *ColisaoNave*(), agora na condição em que a colisão e verdadeira adicione o código que cria a explosão.

```
explosoes.add(new Explosao(dx1 - 13, dy1 + 10));
tempos.add(new Timer(50, explosoes.get(explosoes
.size() - 1)));

Timer tempoTemp = tempos.get(tempos.size() - 1);
tempoTemp.start();
```

O mesmo você fara no método *ColisaoLaiser*(), e na condição em que a colisão e verdadeira adicione o código que cria a explosão.

```
explosoes.add(new Explosao(dx1 - 13, dy1 + 10));
tempos.add(new Timer(50,
explosoes.get(explosoes.size() - 1)));

Timer tempoTemp = tempos.get(tempos.size() - 1);
tempoTemp.start();
```

Bem já podemos criamos as explosões porem temo que desenhar na tela para que ela aconteça. Na class interna *Fase*, no método *paint(Graphics g)* implemente o condigo a seguir para podermos desenhar as explosões na tela.

```
Timer tempoTemp;
Explosao expTemp;

for (int i = 0; i < tempos.size(); i++) {
        tempoTemp = tempos.get(i);
        expTemp = explosoes.get(i);
```

```
if (expTemp.getCont() == 3) {
        tempoTemp.stop();
        tempos.remove(i);
        explosoes.remove(i);
} else {
        grafico.drawImage(expTemp.getImagem(), expTemp.getX(),
        expTemp.getY(),
        expTemp.getX() + expTemp.getLar(),
        expTemp.getY() + expTemp.getAlt(),
        2 + (expTemp.getCont() * expTemp.getLar()), 2,
        (expTemp.getCont() * expTemp.getLar())
        + expTemp.getLar() + 2,
        expTemp.getAlt() + 2, null);
        }
}
```

Agora execute a aplicação e veja se ocorre explosão quando o laiser intercepta o inimigo ou quando a nave colide com inimigo.

6 - Tela Game Over

Bem pela ordem natural do jogos, quando o player "nave" colide algum inimigo ele perde uma vida ou acontece o game over, e isso que iremos implementar em nosso game uma tela de game over, vamos la.

Na *class* interna *Fase* crie um objeto do tipo Imagem com o nome *gameover*, esse objeto recebera a imagem de game over.

```
private Image gameover;
```

Agora no construtor da *class* interna *Fase* adicione iremos referenciar onde esta a nossa imagem game over.

```
ImageIcon referencia = new ImageIcon(
        "res/gameOver/fimJogo.jpg");
        gameover = referencia.getImage();
```

Agora temos que realizar algumas alterações em nosso método *paint(Graphics g)*, que e adicionar uma condição que ocorrera em quanto o player "nave" estiver no jogo e outra condição que ocorre quando a nave for destruída ou seja esta fora do jogo. O método ficara da seguinte forma:

```
public void paint(Graphics g) {
        grafico = (Graphics2D) g;

        /*Em jogo*/
        if (nave.isVisivel()) {
        /* Desenha Fundo */
        grafico.drawImage(fundo, 0, 0, null);

        /* Desenha Nave */
        grafico.drawImage(nave.getImagem(), nave.getX(), nave.getY(),
```

```
nave.getX() + nave.getLar(), nave.getY() + nave.getAlt(), 1 + (nave.getPos() *
nave.getLar()), 1, 1 + (nave.getPos() * nave.getLar()) + nave.getLar(),
nave.getAlt() + 1, null);

        /* Desenha Laiser */
        Laiser laiserTemp;
        for (int i = 0; i < laiser.size(); i++) {
                laiserTemp = laiser.get(i);
                grafico.drawImage(laiserTemp.getImagem(),
                laiserTemp.getX(), laiserTemp.getY(),
                laiserTemp.getX() + laiserTemp.getLar(),
                laiserTemp.getY() + laiserTemp.getAlt(), 1,
                laiserTemp.getLar() + 1, laiserTemp.getAlt() + 1,
                null);
        }

        /* Desenha Inimigo */
        Inimigo iniTemp;
        for (int i = 0; i < inimigos.size(); i++) {
                iniTemp = inimigos.get(i);
                grafico.drawImage(iniTemp.getImagem(),
                (int) iniTemp.getX(), (int) iniTemp.getY(),
                (int) iniTemp.getX() + iniTemp.getLar(),
                (int) iniTemp.getY() + iniTemp.getAlt(), 1, 1,
                iniTemp.getLar() + 1, iniTemp.getAlt() + 1, null);
        }

        /* Desenha Explosão */
        Timer tempoTemp;
        Explosao expTemp;
        for (int i = 0; i < tempos.size(); i++) {
        tempoTemp = tempos.get(i);
        expTemp = explosoes.get(i);
```

```
            if (expTemp.getCont() == 3) {
                        tempoTemp.stop();
                        tempos.remove(i);
                        explosoes.remove(i);
            } else {
            grafico.drawImage(expTemp.getImagem(), expTemp.getX(),
            expTemp.getY(), expTemp.getX() + expTemp.getLar(),
            expTemp.getY() + expTemp.getAlt(), 2 + (expTemp.getCont() *
            expTemp.getLar()), 2, (expTemp.getCont() * expTemp.getLar()) +
            expTemp.getLar() + 2, expTemp.getAlt() + 2, null);
            }
}
}

            /* GameOver */
            if (!nave.isVisivel()) {
                        grafico.drawImage(gameover, 150, 150, null);
                        grafico.setColor(Color.WHITE);
                        grafico.drawString("ENTER PARA NOVO JOGO.", 220, 430);
                        timer.stop();
            }
            g.dispose();
}
```

No método *colisaoNave*() adicione a instrução que define que a nave esta fora do jogo ou seja nave destruída.

```
nave.setVisivel(false);
```

Bem, abaixo da class interna *NovoInimigo* devemos criar um método chamado *ReiniciarInimigo*() que ira remover todos os inimigo da tela e outro método chamado *ReiniciarLaiser*() para remover todos os laiser da tela.

```
        private void ReiniciarInimigos() {
                for (int i = 0; i < inimigos.size(); i++) {
                        inimigos.remove(i);
                        i--;
                }
                        }

        private void ReiniciarLaiser() {
                for (int i = 0; i < laiser.size(); i++) {
                        laiser.remove(i);
                        i--;
                }
                        }
```

Na *class* interna *Controle* no método *pressed*, devemos adicionar uma tecla pra recomeçar o jogo e redefinir algumas configurações iniciais.

```
if (codigo == KeyEvent.VK_ENTER) {
        nave.setVisivel(true);
        ReiniciarInimigos();
        ReiniciarLaiser();
        timer.start();
}
```

7 - Movendo o Background

Uma melhoria que iremos implementar que dara um aspecto visual aprimorado do nosso game e mover o fundo do cenário, tornando um efeito mais agradável ao usuário.

Primeiramente na *class* interna *Fase*, iremos criar uma variável do tipo inteiro que estará recebendo os pixels da imagem em movimento.

```
private int fundoEsp = 0;
```

Dentro da class interna *Fase*, criaremos uma outra *class* interna chamada *Velocidade*, que implementa a interface *ActionListaner* com o método actionPerformed(), ou seja um método ouvinte dos objetos. Nesse método teremos uma condição que esta constantemente verificando o tamanho da nossa imagem na tela, que estará realizando uma operação condicional, enquanto fundoEsp for menor que um determinado valor "tamanho da imagem" incrementa mais um, deslocando a imagem para baixo, quando a condição não mais satisfazer, ela reinicia imagem dando a ilusão de repetição e deslocamento do fundo.

```
private class Velocidade implements ActionListener {
public void actionPerformed(ActionEvent e) {
        if (fundoEsp < 600)
                fundoEsp++;
        else
                fundoEsp = 0;
                }
        }
```

No método *paint(Graphics g)* na condiçao nave em jogo, iremos modifica a forma em que estamos desenhando o fundo, para desenhamos dois fundo onde um desse e outro vem sendo desenhado em seguida:

```
grafico.drawImage(fundo, 0, fundoEsp, null);
grafico.drawImage(fundo, 0, fundoEsp - 600, null);
```

No construtor da *class* Fase, iremos instanciar uma thread com a *class* interna *Velociade* que estará movendo o fundo em multi-processamento. So lembrado que o primeiro parâmetro dentro da nossa instancia da thread e o tempo de espera para execução ou seja a velocidade que queremos que o fundo se desloque "mova".

```
Timer velocidade = new Timer(18, new Velocidade());
velocidade.start();
```

8 - Conhecendo a lib Jlayer

O nosso game já esta bem legal, porem ainda incrementaremos efeitos sonoros, musicas e sons, para torná-lo mais interativo e interessante, dando aquele toque especial.

Java possui *class* prontas para reproduzir arquivos do tipo *.midi*, *.wav*, porem utilizaremos uma biblioteca *JZoom* que possui varias lib para reproduzir arquivos de sons, entre vários formato o .mp3 que e o que estaremos usando no nosso game.

Importe o arquivo *Jlayer.jar* que baixamos no inicio para uma pasta *lib* que deve ser criada e configurada no projeto.

8.1 - Adicionando Música

Inicie criando um novo pacote com o nome *br.com.game.app.efeitos.sonoro* dentro dessa *class* estaremos criando uma *class* com o nome Musica, no irei comentar como funciona essa *class* pois a mesma já se encontra comentada como a seguir:

```
public class Musica {
	private static boolean loop;

	public boolean getloop() {
		return loop;
	}

	public void setloop(boolean l) {
		loop = l;
	}

	public void main(String[] args) {

		// String com o caminho do arquivo MP3 a ser tocado
		String path = "res/sons/cenario/musica.mp3";

		// Instanciação de um objeto File com o arquivo MP3
		File mp3File = new File(path);

		// Instanciação do Objeto MP3, a qual criamos a classe.
		MP3MusicaCenario musica = new MP3MusicaCenario();
		musica.tocarMusicaCenario(mp3File);

		// Finalmente a chamada do método que toca a música
		musica.start();
	}
```

```
/**
 * ============================================
*==========CLASS INTERNA MP3 MUSICA CENARIO
*============================================
*/
public static class MP3MusicaCenario extends Thread {
        // Objeto para nosso arquivo MP3 a ser tocado
        private File mp3;

        // Objeto Player da biblioteca jLayer. Ele tocará o arquivo MP3
        private Player player;

/*
 * Construtor que recebe o objeto File referenciando o arquivo MP3 a ser
 * tocado e atribui ao atributo MP3 da classe.
 *
 * @param mp3
 */
public void tocarMusicaCenario(File mp3) {
        this.mp3 = mp3;
}

/*
*=========================================================
*================ METODO QUE TOCA O MP3
*=========================================================
 */
public void run() {
        try {
                do {
                FileInputStream fis = new FileInputStream(mp3);
                BufferedInputStream bis = new BufferedInputStream(fis);
```

```
                this.player = new Player(bis);
                // System.out.println("Tocando Musica Cenario!");
                this.player.play();
                // System.out.println("Terminado Musica Cenario!");
                                } while (loop);
                        } catch (Exception e) {
                                JOptionPane.showMessageDialog(null,
                                                "Problema ao tocar
Musica do Cenário!" + mp3);
                                e.printStackTrace();
                        }
                }

                public void close() {
                        loop = false;
                        player.close();
                        this.interrupt();
                }

        }

}
```

Bem, com a *class Musica.java* implementada agora e so chama-la "executá-la" quando executar o jogo, para isso iremos apara a *class Game.java* criaremos um objeto para manipular para acessar os métodos da *class Musica.java*.

```
private Musica musica;
```

Agora no método *inicializar()*, instanciaremos o objeto musica e *start* a *thread* que executara a musica do game.

```
/* TOCAR MUSICA DO CENARIO */
mc = new MusicaCenario();
mc.main(null);
mc.setloop(true);
```

Agora execute a aplicação e verifique se esta reproduzindo o som musical do game.

8.2 – Adicionando Sons de Efeito

Ótimo o nosso game já possui um fundo musical, e iremos agora implementar efeito sonoros como som de disparo, explosão, game over entre outros. Criaremos então, as *class EfDisparoLaiser.java*, *EfGameOve.java* e *EfExplosao.java*, da mesma forma que criamos a *class Musica.java*, alterando apenas a referencia do som que sera executado.

Referencias dos sons de cada *class*:

- *EfDisparoLaiser.java*
  String path = "res/sons/disparo/shoot.mp3";

- *EfGameOve.java*
  String path = "res/sons/missao/falida/fimJogo.mp3";

- *EfExplosao.java*
  String path = "res/sons/explosao/asteroid_explosion.mp3";

Agora e só executar da mesma forma que anteriormente, porem onde deve ocorre a execução do som.

Para ocorrer o som de explosão do inimigo incrementaremos o código seguinte no método ColisaoLaiser(), antes do efeito explosão.

```
/* Toca efeito explosao */
EfExplosao somExplosao = new EfExplosao();
somExplosao.main(null);
```

Agora e só repetir o mesmo passo acima no método *ColisaoNave()*, para som o som de fim de jogo.

```
EfGameOver somGameOver = new EfGameOver();
```

```
somGameOver.main(null);
```

Para ocorrer o som de disparo de *laiser* da nave, incrementaremos o código seguinte na *class* interna *Controle* na condição onde e disparado o *laiser*:

```
/* Toca efeito Laiser */
EfDisparoLaiser somDisparoLaiser = new EfDisparoLaiser();
somDisparoLaiser.main(null);
```

9 - Criando o Executável do Jogo

Se você já chegou aqui, já deve esta querendo saber como pode compartilhar o jogo que você construiu como seus amigos, e com a IDE eclipse podemos fazer isso facilmente.

O que você tem que fazer e clicar com o botão direito do mouse em cima do seu projeto, navegar ate a opção **exportar**, na janela que aparecer selecione o diretório **Java >> Runnble JAR file**, como mostra a figura a baixo.

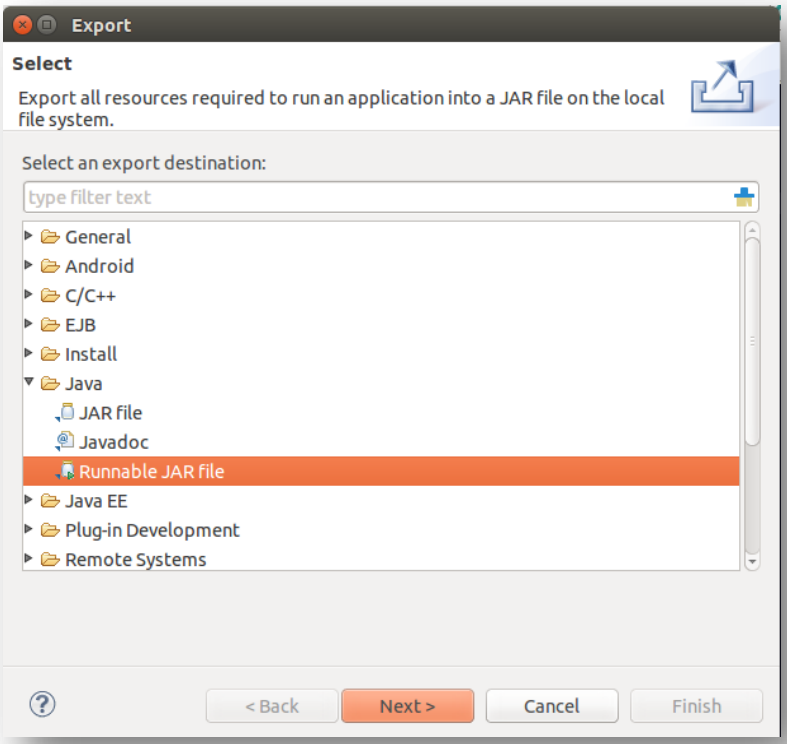

Figura 14

Clique em avançar e na próxima janela, na opção **Launch configuration**, você deve selecionar a *class Game.java*, que contem o método *main* do jogo responsável pela inicialização da aplicação. Na opção **Export destination** você ira apontar o local onde deseja salvar o arquivo executável que sera gerado. Na opção **Library handling**, você seleciona conforme mostra a imagem abaixo, pois ela define que as biblioteca e os recurso do jogo será salvos dentro do arquivo.jar que será criado.

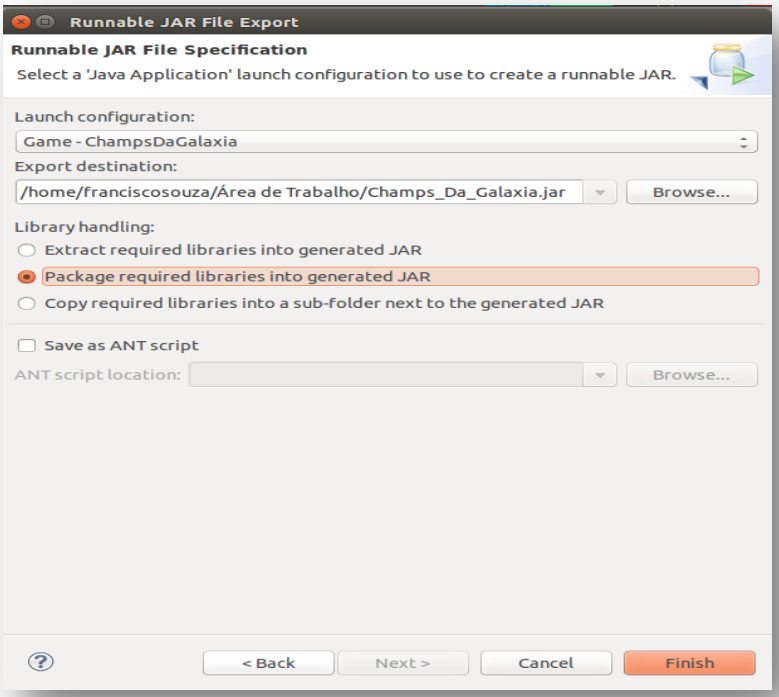

Figura 15

Pronto, se ocorre tudo certinho seu arquivo já deve ter sido criado no local que você escolheu para salvar, e so dar duplo clique e aproveitar.

10 - Desafios

1. Adicione um placar de pontuação no jogo.
2. Adicione vida extra ao player nave.
3. Adicione captura de nova vida.
4. Adicione novas armas.
5. Adicione captura de nova arma.
6. Adicione a opção de o usuário ativar e deseivar o som no jogo.
7. Adicione Tiro Inimigo.
8. Adicione a função multiplayer no jogo.

11 – Conclusão

Vimos e entendemos algumas noções básicas na construção de um jogo digital, porem existem padrões de projetos e frameworks usados por Indústrias de jogos e muitos desenvolvedores, que facilita na construção, desempenho, entre outros aspectos. Percebemos e praticamos o quanto e trabalhoso se construir jogos, e o quanto e prazeroso quando se conclui um projeto de entretenimento desse tipo, desde já, se você chegou ate aqui, espero que tenha gostado do que foi abordado e desfrutado ao máximo.

12 - Referências Bibliográficas

Super Mario Bros, Disponível em: <http://pt.wikipedia.org/wiki/Super_Mario_Bros>; Acessado em: 24 de abril de 2014.

Aero figtters, Disponível em: <http://pt.wikipedia.org/wiki/Aero_fighters>; Acessado em 24 de abril de 2014.

Developing Games in Java David Brackeen, Disponível em: <http://www.brackeen.com/javagamebook/>, acessado em 03 de dezembro de 2013.

PontoV Programação de Jogos profissionais, Disponível em: <http://pontov.com.br/site/java/48-java2d> Acessado em 06 de dezembro de 2013.

Killer Game Programming in Java - Andrew Davison, Disponível em <http://fivedots.coe.psu.ac.th/~ad/jg/>; Acessado em 02 de dezembro de 2013.

Abrindo o Jogo, Disponível em: <http://abrindoojogo.com.br/djj-index>; Acessado em 26 de novembro de 2013.

Sprite Data Base <http://spritedatabase.net/>

The Spriters Ressouce <http://www.spriters-resource.com/>

Game Marker Brasil <http://gmbr.forumeiros.com/t12-lista-de-sites-com-sprites>

Clubes dos Geeks Disponível em: <http://clubedosgeeks.com.br/programacao/java/desenvolvimento-de-jogo-em-java>; Acessado em 12 de janeiro de 2014

Livro: Killer Game Programming in Jav - Andrew Davison.

Sinopse

Esse livro é destinado a inspirar e motivar estudantes novatos ou até mesmo veteranos programadores, para programação voltada a desenvolvimento de jogos, ver que é possível criar bons jogos com linguagem de alto nível com Java e entender a mecânica básica de um jogo. Abordo de forma simples passo a passo a construção de um jogo de nave Champs da Galáxia, onde implementaremos o player jogador, disparos de armas, inimigos, música de fundo, efeitos sonoro, cenário móvel, e muito mais; no fim você ainda terá alguns desafios proposto para implementar no seu jogo, se aventure e venha nessa introdução ao desenvolvimento de jogos com Java.

CLUBE DOS GEEKS