Revista TECCEN – Edição Especial – volume 2 – número 1 - março de 2009 – ISSN 1984-0993

Tutorial para o Desenvolvimento de Jogos 2D usando a Linguagem Java

Soraia Teixeira Barbosa[1], Carlos Vitor de Alencar Carvalho[2]

[1]Universidade Severino Sombra, Centro de Ciências Exatas e Tecnológicas e da Natureza, Curso de Sistemas de Informação, soraia_tb@yahoo.com.br

[2]Universidade Severino Sombra, Centro de Ciências Exatas e Tecnológicas e da Natureza, Curso de Sistemas de Informação e Programa de Mestrado Profissional em Educação Matemática e UniFOA – Centro Universitário de Volta Redonda – Departamento das Ciências de Tecnologia e Engenharias, cvitorc@gmail.com

Resumo. *Atualmente é grande o estudo e desenvolvimento de jogos eletrônicos tanto no meio acadêmico como fora dele. Entretanto o desenvolvimento do mesmo não é uma tarefa trivial. Este artigo mostra o desenvolvimento de um jogo computacional através de uma versão do conhecido jogo de arcade Space Invaders. O artigo proposto tem como objetivo mostrar os passos básicos para a construção de um jogo computacional em duas dimensões utilizando a linguagem de programação Java.*

1 Introdução

A área de jogos eletrônicos está se tornando cada vez mais popular, sendo uma área de grande interesse financeiro principalmente no segmento de entretenimento digital podendo ser encontrado em diversos dispositivos, como computadores pessoais, console domésticos e dispositivos móveis [Barboza 2008].

Uma das aplicações interessantes dos jogos computacionais é sua utilização no processo educacional. Muitos jogos possibilitam o desenvolvimento de habilidades como cooperação, competição, perseverança, envolvimento, organização e autonomia. Através de *softwares* educacionais, temos a oportunidade de criar novos jogos e reinventar jogos utilizados em madeira, papel, etc. A possibilidade de divertidas animações, a integração de várias mídias como a escrita, a imagem, o vídeo e o som que o computador oferece enriquecem os jogos [Porto et al. 2008]. Casos de sucesso de jogos computacionais na educação podem ser vistos em Mendes (2007), Santos (2008) e Porto (2008) através do *software* **CONSTRUFIG3D**. Este artigo visa mostrar os passos básicos do desenvolvimento de um jogo 2D inspirado no clássico de *arcade*, *Space Invaders*.

A linguagem de programação escolhida foi Java, devido á sua portabilidade e facilidade de utilização dos seus recursos de computação gráfica através do pacote Java2D. O objetivo é que este texto seja mais uma referência para alunos de cursos de computação interessados em iniciar os estudos e desenvolvimento de jogos computacionais.

O restante deste artigo está organizado da seguinte forma: na seção 2 são descritas as fases de desenvolvimento de um jogo computacional. Na seção 3 é mostrada a classe principal do sistema. Na seção 4 é descrita a classe *Stage*. A seção 5 descreve como é feito o gerenciamento dos recursos (sons e imagens, por exemplo). Na seção 6 são mostradas as informações os atores e finalmente na seção 7 os resultados e considerações finais do trabalho.

2 Planejamento de um jogo computacional

A fase inicial do ciclo de desenvolvimento de um jogo é o planejamento. Assim que se tem a idéia inicial do jogo, deve-se fazer uma reunião com membros da equipe onde à mesma deve ser exposta e discutida [PERUCIA et al. 2007] . Cabe lembrar que uma equipe para o desenvolvimento de jogos deve possuir profissionais de várias áreas (artística, produtores de textos, animadores, programadores etc.). Terminada a fase de planejamento inicia-se a fase de *game design*.

Na fase de *game design* são definidas as principais características do jogo, como sua *interface*, jogabilidade, personagens, fases, inimigos, itens e outras características gerais, que geram o *design document*, um documento onde os membros da equipe podem ter noção de como irão trabalhar para realizar o que foi planejado [PERUCIA et al. 2007].

É importante também a criação de um *level design*, que é como um mapa geral com as missões e desafios das fases em geral. Em projetos maiores é necessário que cada fase tenha seu planejamento específico, para que cada membro da equipe possa planejar como fazer a sua parte.

Depois de tudo planejado, inicia-se a fase de criação ou desenvolvimento do jogo. A Figura 1 mostra esquematicamente as etapas descritas acima.

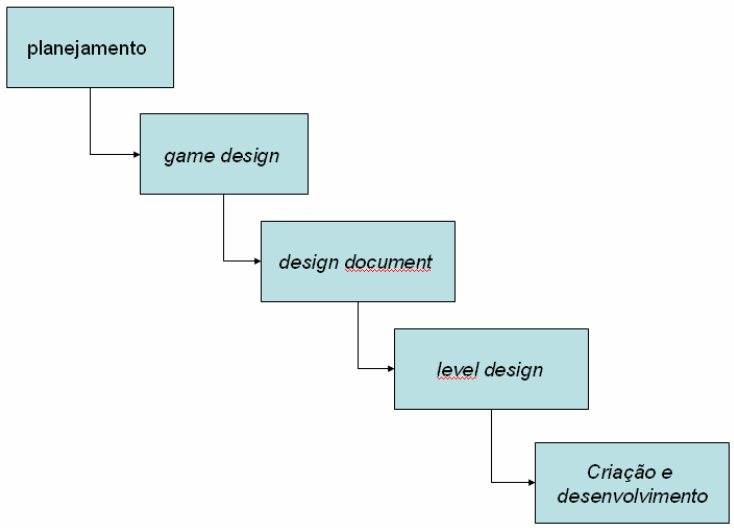

Figura 1. Esquema das etapas de desenvolvimento de um jogo computacional.

Este artigo irá visar justamente esta parte: criação e programação, utilizando a linguagem Java. Nesta etapa, é aconselhável a criação de versões intermediárias, pois estas permitem o acompanhamento da evolução do projeto, além de facilitar na detecção de *bugs* e na implementação de melhorias.

3 Criando a Classe Principal

Primeiro será criada a classe principal do jogo, a qual se chamará de *Game*. Esta classe irá conter a inicialização do jogo, que deve ser feita como mostra a Tabela 1.

Tabela 1. Game.java

```
public static void main (String args[]){
        Game inv = new Game();
```

```
    }
```

3.1 A Janela

Dentro da classe *Game* também será feito o desenho da janela. A classe *Game* deve ser uma extensão de *Canvas.* Um componente *Canvas* possibilita criar uma área na tela, na qual a aplicação poderá desenhar através da sobrescrição do seu método *paint()*, como mostra a Tabela 2.

Tabela 2. Game.java

```
public class Game extends Canvas
```

No construtor da classe, deve ser feito o desenho da janela e suas configurações. Para definir a largura (*WIDTH*) e altura (*HEIGHT*) da janela, são usadas constantes da classe *Stage*, que será criada em seguida, pois quando for preciso mudar o tamanho da janela, basta mudar os valores destas constantes no código-fonte (Tabela 3).

Tabela 3. Game.java

```
   public Game(){
         JFrame window = new JFrame("Invasores do Espaço");
         JPanel panel = (JPanel)window.getContentPane();
         setBounds(0, 0, Stage.WIDTH, Stage.HEIGHT);
         panel.setPreferredSize(new              Dimension(Stage.WIDTH,
Stage.HEIGHT));
         panel.setLayout(null);
         panel.add(this);
         window.setBounds(0, 0, Stage.WIDTH, Stage.HEIGHT);
         window.setResizable(false);
         window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Depois de criada, a janela deve se tornar visível e ser prioridade para os comandos de entrada (Tabela 4).

Tabela 4. Game.java

```
window.setVisible(true);
requestFocus();}
```

4 A Classe Stage

A classe *Stage* contém alguns dados sobre a fase. Inicialmente seu código será como mostrado na Tabela 5.

Tabela 5. Stage.java

```
public interface Stage {
    public static final int WIDTH = 800;
    public static final int HEIGHT = 600;
```

```
    public static final int PLAY_HEIGHT = 520;
    public static final int SPEED = 10;
}
```

As variáveis *WIDTH* e *HEIGHT* vão conter os valores de largura e altura da fase e das janelas, a variável *PLAY_HEIGHT* terá o valor da área em que acontecerá a ação do jogo e *SPEED* será usada para definir a velocidade do jogo.

5 Gerenciamentos dos recursos do jogo

Na tela serão desenhados todos os *sprites* (imagens que irão representar os gráficos dos personagens, armas etc.) e serão exibidas as informações do jogo. Para isso, será preciso criar classes para carregar e controlar esses *sprites*.

O mesmo método pode ser usado para carregar efeitos sonoros e músicas, que também são muito importantes em um jogo, por isso pode ser criada uma classe abstrata com os métodos comuns no carregamento de ambos os recursos. O controle desses recursos é feito através das classes *ResourceCache*, como mostra a Tabela 6. Será usado um *HashMap* para armazenar o nome do *sprite* e a imagem carregada.

Tabela 6. ResourceCache.java

```
public abstract class ResourceCache {
    protected HashMap resources;
    public ResourceCache(){
        resources = new HashMap();
}
}
```

Para indicar o caminho dos recursos, será utilizado *getClass().getClassLoader().getResource()*, pois desta forma poderá ser usada uma *URL* indicando o caminho direto (diretórios onde estão os recursos) de onde as classes estão sendo carregadas, o que facilita na hora de passar o jogo para um *applet*, por exemplo, sem precisar fazer mudanças em relação ao carregamento de imagens (Tabela 7).

Tabela 7. ResourceCache.java

```
protected Object loadResource(String name){
        URL url = null;
        url = getClass().getClassLoader().getResource(name);
        return loadResource(url);
    }
```

Todos os recursos estarão na mesma pasta das classes, em um diretório chamado *resources*, então, toda vez que uma imagem for carregada, o aplicativo deve buscar nesta pasta. Depois disso, o recurso será armazenado no *HashMap* criado anteriormente (Tabela 8).

Tabela 8. ResourceCache.java

```
    protected Object getResource(String name){
```

```
        Object res = resources.get(name);
        if(res == null){
            res = loadResource("resources/"+name);
            resources.put(name, res);
        }
        return res;
    }
    protected abstract Object loadResource(URL url);
}
```

5.1 Imagens

A classe usada para carregar e gerenciar as imagens será a *SpriteManager*, que será uma extensão de *ResourceCache* (Tabela 9).

Tabela 9. SpriteManager.java

```
public class SpriteManager extends ResourceCache{
```

O método *loadResource(URL url)* tenta ler as imagens, se não conseguir, cria um *JOptionPane* com uma mensagem de erro e fecha o aplicativo (Tabela 10).

Tabela 10. SpriteManager.java

```
   protected Object loadResource (URL url){
        try{
            return ImageIO.read(url);
        }catch(Exception e){
                JOptionPane.showMessageDialog(null, "Erro ao carregar
    o jogo." +
    "Por favor, verifique a instalação.", "Erro!",
    JOptionPane.ERROR_MESSAGE);
                System.exit(0);
                return null;
        }}
```

6 Atores

Os atores são todos os objetos que criam a interação no jogo. Neste caso existem o herói, os inimigos e o tiro. A classe *Actor* irá conter os métodos comuns a estes atores (Tabela 11).

Tabela 11. Actor.java

```
public class Actor {
    protected int x,y;
    protected int width, height;
    protected String[] spriteNames;
    protected Stage stage;
```

```
    protected SpriteManager spriteManager;
    protected int currentFrame;
    protected int frameSpeed, t;
    protected boolean remove;

    public Actor (Stage stage){
        this.stage = stage;
        spriteManager = stage.getSpriteManager();
        currentFrame = 0;
        frameSpeed = 1;
        t=0;
    }

    public void paint(Graphics2D g){

g.drawImage(spriteManager.getSprite(spriteNames[currentFrame]), x, y,
stage); }
```

Se um *sprite* for removido da tela durante a execução do *loop* principal do jogo, falhas podem ser geradas na execução do aplicativo. O método *setRemove* serve para contornar este problema, pois ele "marca" o *sprite* para ser removido durante a próxima execução do loop, evitando problemas na leitura dos *sprites* (Tabela 12).

Tabela 12. Actor.java

```
 public void setRemove(){ remove = true; }
 public boolean isSetToRemove(){ return remove; }
```

O método *getBounds()* cria um retângulo em torno do *sprite* (Tabela 13).

Tabela 13. Actor.java

```
public Rectangle getBounds(){
        return new Rectangle(x,y,width,height);
    }
```

Estes retângulos são usados nos cálculos de detecção de colisão dos personagens. A Figura 2 exemplifica como esses retângulos são visualizados pelo aplicativo.

Figura 2. Como funciona a detecção de colisão.

O método *act()* faz a troca de *sprites* para realizar a animação dos atores conforme pode ser visto na Tabela 14.

Tabela 14. Actor.java

```
public void act(){
        t++;
        if(t % frameSpeed == 0){
            t=0;
            currentFrame = (currentFrame + 1) % spriteNames.length;
        }}}
```

6.1 O Personagem

A classe *Player* irá representar o personagem que o jogador irá controlar que nesse caso é uma nave. Esta classe possui diversas variáveis e métodos exclusivos do jogador (Tabela 15).

Tabela 15. Player.java

```
public class Player extends Actor {
    protected int vx;
    protected static final int PLAYER_SPEED = 4;
    private boolean left, right;
```

O método act() desta classe evita que a nave ultrapasse os limites da tela (Tabela 16).

Tabela 16. Player.java

```
    public void act(){
        super.act();
```

```
        x+=vx;
        if (x < 0 )
          x = 0;
        if (x > Stage.WIDTH - getWidth())
          x = Stage.WIDTH - getWidth();
      }
```

O método *updateSpeed()* faz com que a nave possa se mover para os lados, de acordo com a tecla pressionada (Tabela 17).

Tabela 17. Player.java

```
    protected void updateSpeed(){
        vx = 0;
        if(left)
            vx = -PLAYER_SPEED;
        if(right)
            vx = PLAYER_SPEED;
    }
```

O método *fire()* faz com que a nave possa atirar. Primeiro é feito o posicionamento do tiro e depois criado um objeto para representá-lo na tela (Tabela 18).

Tabela 18. Player.java

```
    public void fire(){
        Shot shot = new Shot(stage);
        shot.setX(x+25);
        shot.setY(y  - shot.getHeight());
        stage.addActor(shot);
        stage.getSoundManager().playSong("sound/shot.wav");
    }
```

7 Resultados e Considerações Finais

Ao término deste artigo, no qual foram abordados métodos para a criação de um jogo computacional 2D, espera-se que os estudantes da área de computação possam ter uma idéia inicial de como esses jogos são feitos, servindo como base para pesquisas ou estudos. A Figura 3 mostra a tela do *software* em execução.

Através deste tutorial foram explicados conceitos importantes no desenvolvimento de um jogo, como por exemplo, o tratamento de imagens, a detecção de colisão entre os personagens e a movimentação do jogador na tela. Em uma próxima etapa será desenvolvido e pesquisa uma estratégia para o enfoco educativo do sistema possivelmente com alteração para a temática na área de matemática.

Figura 3. Tela do jogo.

Agradecimentos

Os autores agradecem à **FAPERJ** (Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro) pelo auxílio financeiro recebido. O segundo autor agradece à **FUNDADESP** (Fundação Nacional de Desenvolvimento do Ensino Superior Particular) pela bolsa de produtividade de pesquisa de doutorado.

Referências Bibliográficas

Barboza, D. C. (2008) *Ambiente Visual para desenvolvimento de jogos eletrônicos*, Monografia – Curso de Bacharelado em Ciência da Computação do Centro Universitário Serra dos Órgãos.

Deitel H. M., Deitel P. J. (2002) *Java Como Programar*: 4ª edição. Bookman 2002.

Harbour, J. S. (2007). *Beginning Java Game Programming,* 2th edition.

Mendes, J. L. de S., Carvalho, C. V. A., Carvalho, J. V. (2007) *CONSTRUFIG3D: Uma Ferramenta Computacional para apoio ao ensino da Geometria Plana e Espacial.* RENOTE. Revista Novas Tecnologias na Educação, v. 5, n 1, p. 1/10-10, 2007.

Perucia A., Berthêm A.., Bertschinger G., Castro R. R. (2005) *Desenvolvimento de Jogos Eletrônicos – Teoria e Prática:* 2ª edição. Novatec, 2005.

Porto, I. da P. G. Carvalho, C. V. A., Oliveira, R. (2008) *O jogo Computacional TANGRAM: um objeto de Aprendizagem sobre Geometria.* IV Colóquio de História e Tecnologia no ensino da Matemática - HTEM, UFRJ, 2008.

Santos, S. C., Carvalho, Janaina Veiga, Carvalho, C. V. A. (2008) *Utilização e Avaliação do sistema computacional CONSTRUFIG3D para apoio ao ensino da geometria*. RENOTE. Revista Novas Tecnologias na Educação, v. 6, número 1, p 1-9, 2008.