APÊNDICE C
O Motor 2D javaPlay

O motor 2D javaPlay foi especialmente desenvolvido para este livro e tem a estrutura de um motor profissional. O completo entendimento do seu funcionamento requer algum tempo de estudo. A idéia é que você começe usando as suas funcionalidades mais gerais, sem precisar entrar em detalhes. Depois você vai esmiuçando cada uma das classes, com auxílio da Parte III do livro (site internet). Para um estudo mais detalhado, veja o Apêndice D que contém os princípios de projeto do javaPlay.

Neste apêndice você encontra o código e as instruções das versões mais básicas que servem de ponto de partida para projetos mais completos.

Todas as versões são compostas por dois pacotes:

- Demo
- javaPlay

Para criar um jogo, você deve alterar o **Demo**. As primeiras alterações podem ser inclusões de Game Objects. O **Demo** já vem com um GameObject chamado Player. No método step de SimpleController.java está a movimentação do Player através do teclado para cada tick do jogo. Existem 3 versões do Motor, da menos completa à mais completa: Versão Mini, Versão 00 e Versão 0. Apenas na versão Versão 0, a mais completa, o método step de SimpleController.java (pacote Demo) tem tratamento da colisão.

O módulo **javaPlay** é o motor propriamente dito. Na Versão Mini, o javaPlay é bastante reduzido (para fins didáticos). Nas outras versões o javaPlay é completo e não muda.

C.1 javaPlay – Versão Mini

Esta é a versão mais simples do Motor (Fig. C1), que basicamente implementa um loop. O módulo DemoMini contém apenas as classes Main.java e Player.java. No Player.java temos os seguintes métodos: Player, step e draw. No método `Player()` definimos x e y iniciais (e criamos e inicializamos outras variáveis que queremos usar). No método draw, desenhamos o que queremos que o GameObject Player seja (no caso: pintamos o canvas de branco, desenhamos o player como um quadrado preto na coordenada (x,y), desenhamos dois retângulos como se fossem paredes e desenhamos um círculo na coordenada (x2,y2). No método step, definimos o que deve mudar a cada tick do jogo (no caso, as coordenadas (x,y) e (x2,y2)). O javaPlay contém as versões mais básicas das 3 classes indispensáveis: GameCanvas.java, GameEngine.java e GameObject.java. Nesta versão também estão indicadas as modificações que poderiam ser feitas para usar sprites, dando a esta versão Mini um aspecto de um *Game Engine* de fato. Note que o exercício de desenhar o player como um quadrado preto é apenas

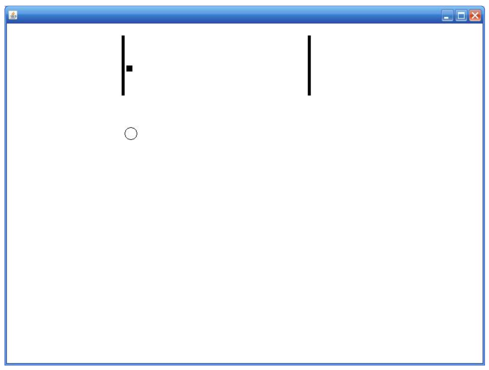
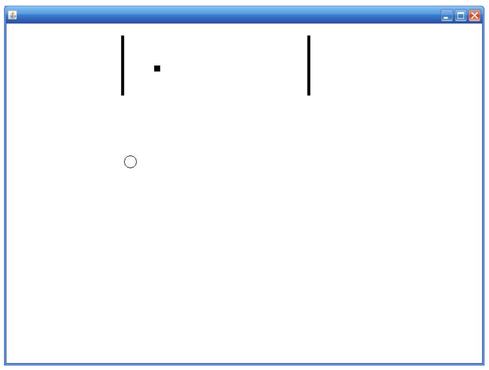

Fig. C1 Janelas do demo do motor Nível Mini.

didático, visto que os únicos desenhos em um verdadeiro motor de jogos 2D são carregamentos de imagens (background, tiles e sprites).

C.1.1 Demo da Versão Mini

Main.java

```
/*
 * Main
 */

package demo;

import javaPlay.GameEngine;

public class Main
{
    public static void main(String[] args)
    {
        Player player = new Player();

        GameEngine.getInstance().addGameObject(player);

        GameEngine.getInstance().run();
    }
}
```

Player.java

```
/*
 *  GameObject: player
 */

package demo;

import java.awt.Color;
import java.awt.Graphics;
// import java.util.logging.Level;   // if Sprite is required
// import java.util.logging.Logger;  // if Sprite is required
import javaPlay.GameEngine;
import javaPlay.GameObject;
// import javaPlay.Sprite;

/*
 * @author VisionLab/PUC-Rio
 */
public class Player extends GameObject
{
//    private Sprite sprite;
    private int delta;
    private int min;
    private int max;
    private int x2, y2; // circulo extra

    public Player()
    {
/* Descomente para sprites
        try
        {
            sprite = new Sprite("player.png", 8, 32, 32);
        }
        catch (Exception ex)
        {
            Logger.getLogger(Player.class.getName()).log(Level.SEVERE, null, ex);
        }
*/
        x = 200;
        y = 100;

        x2 = 200;  // circulo extra
        y2 = 200;  // circulo extra

        delta = 1;
        min = x;
//        max = GameEngine.getInstance().getGameCanvas().getWidth();  // = canvas width
        max = 500;
    }
    public void step(long timeElapsed)
    {
        x += delta;
        y2 += delta;
        if(x > max)
            delta *= -1;
        if(x < min)
            delta *= -1;

    }

    public void draw(Graphics g)
    {
        // canvas in white
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, GameEngine.getInstance().getGameCanvas().getWidth(),
                GameEngine.getInstance().getGameCanvas().getHeight());
        // draw player as a back square
        g.setColor(Color.BLACK);
        g.fillRect(x, y, 10, 10);
        // draw walls
        g.fillRect(195, 50, 5, 100);
        g.fillRect(505, 50, 5, 100);
        // draw a circle
        g.drawOval(x2,y2,20,20);

//        sprite.draw(g, x, y);
    }
}
```

C.1.2 javaPlay reduzido para a Versão Mini

GameCanvas.java

```
/*
 * GameCanvas
 */

package javaPlay;

import java.awt.Canvas;
import java.awt.Container;
import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.awt.GraphicsDevice;
import java.awt.GraphicsEnvironment;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.image.BufferStrategy;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JFrame;

/**
 * @author VisionLab/PUC-Rio
 */
public class GameCanvas extends JFrame
{
    private final int defaultScreenWidth = 800;
    private final int defaultScreenHeight = 600;
    private Graphics g;
    private BufferStrategy bf;
    private int renderScreenStartX;
    private int renderScreenStartY;

    public GameCanvas(GraphicsConfiguration gc)
    {
        super(gc);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(defaultScreenWidth, defaultScreenHeight);
        setVisible(true);

        createBufferStrategy(2);

        renderScreenStartX = this.getContentPane().getLocationOnScreen().x;
        renderScreenStartY = this.getContentPane().getLocationOnScreen().y;
        bf = getBufferStrategy();
    }

    public int getRenderScreenStartX()
    {
        return renderScreenStartX;
    }

    public int getRenderScreenStartY()
    {
        return renderScreenStartY;
    }

    public Graphics getGameGraphics()
    {
        g = bf.getDrawGraphics();
        return g;
    }

    public void swapBuffers()
    {
        bf.show();
        g.dispose();
        Toolkit.getDefaultToolkit().sync();
    }
}
```

GameEngine.java

```
/*
 * GameEngine
 */

package javaPlay;

import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.awt.GraphicsDevice;
import java.awt.GraphicsEnvironment;
import java.util.*;

/**
 * @author VisionLab/PUC-Rio
 */
public class GameEngine
{
    private GameCanvas canvas;
    private long lastTime;
    private boolean engineRunning;
    private static GameEngine instance;
    private Vector gameObjects;

    private GameEngine()
    {
        GraphicsEnvironment graphEnv = GraphicsEnvironment.getLocalGraphicsEnvironment();
        GraphicsDevice graphDevice = graphEnv.getDefaultScreenDevice();
        GraphicsConfiguration graphicConf = graphDevice.getDefaultConfiguration();

        canvas = new GameCanvas(graphicConf);

        lastTime = System.currentTimeMillis();
        engineRunning = true;
        instance = null;

        gameObjects = new Vector();
    }

    public static GameEngine getInstance()
    {
        if(instance == null)
        {
            instance = new GameEngine();
        }
        return instance;
    }

    public GameCanvas getGameCanvas()
    {
        return canvas;
    }
    public void requestShutdown()
    {
        engineRunning = false;
    }

    public void addGameObject(GameObject object)
    {
        gameObjects.add(object);
    }

    public void removeGameObject(GameObject object)
    {
        gameObjects.remove(object);
    }

    public void run()
    {
        long currentTime;

        while(engineRunning == true)
        {
            currentTime = System.currentTimeMillis();

            for(int i = 0 ; i < gameObjects.size() ; i++)
            {
                GameObject obj = (GameObject)gameObjects.elementAt(i);

                obj.step(currentTime - lastTime);

                obj.draw(canvas.getGameGraphics());
            }

            canvas.swapBuffers();
        }

        canvas.dispose();
    }
}
```

GameObject.java

```
/*
 * GameObject
 */

package javaPlay;

import java.awt.Graphics;

/**
 * @author VisionLab/PUC-Rio
 */
public abstract class GameObject
{
    public int x;
    public int y;

    public abstract void step(long timeElapsed);
    public abstract void draw(Graphics g);
}
```

C.2 Demo Versão 00

A Versão 00 tem apenas um jogador (Player) e não trata colisão (Fig. C2).

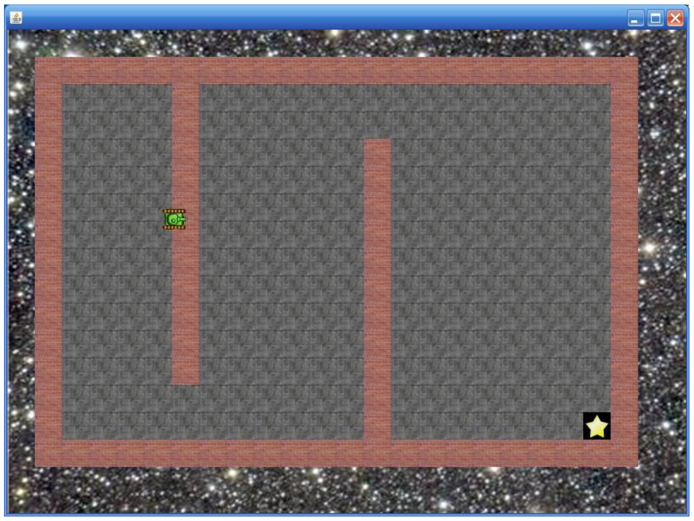

Fig. C2 Janela do demo do motor Versão 00.

O jogo se resume a passear com o jogador (*Player*) em um cenário de tiles, sem colisões. Os elementos gráficos são os seguintes:

1. imagem do jogador (player.png de tamanho 256x32)
2. imagem do fundo (backdrop.jpg de tamanho 800x600)
3. imagens de Tiles:
   muro: wall.jpg de tamanho 32x32
   chão: floor.jpg de tamanho 32x32
   alvo (em forma de estrela): end.jpg de tamanho 32x32

O cenário é definido através de um arquivo texto (com nome scene.scn) no seguinte formato:

número-de-tiles
imagem-do-tile-1
...
imagem-do-tile-n
mapa-de-localização
%
imagem-do-fundo

No exemplo da Fig.C2, o arquivo scene.scn é o seguinte:

```
3
wall.jpg
floor.jpg
end.jpg
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
0,1,2,2,2,2,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1
0,1,2,2,2,2,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1
0,1,2,2,2,2,1,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,2,1
0,1,2,2,2,2,1,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,2,1
```

0,1,2,2,2,2,1,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,1
0,1,2,2,2,2,1,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,1
0,1,2,2,2,2,1,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,1
0,1,2,2,2,2,1,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,1
0,1,2,2,2,2,1,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,1
0,1,2,2,2,2,1,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,1
0,1,2,2,2,2,1,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,1
0,1,2,2,2,2,2,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,1
0,1,2,2,2,2,2,2,2,2,2,2,2,1,2,2,2,2,2,2,2,**3**,1
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
%
backdrop.jpg

Você deve notar o *tile* número 3 (o alvo end.jpg) em negrito no canto inferior direito.

Player.java

```
/*
 * Game Object: Player
 */

package demo;

import java.awt.Graphics;
import javaPlay.GameObject;
import javaPlay.Sprite;

/**
 * @author VisionLab/PUC-Rio
 */
public class Player extends GameObject
{
    private Sprite sprite;
    private int currFrame;

    private final int UP_FRAMES = 0;
    private final int LEFT_FRAMES = 2;
    private final int RIGHT_FRAMES = 6;
    private final int DOWN_FRAMES = 4;

    public void setSprite(Sprite sprite)
    {
        this.sprite = sprite;
        currFrame = 0;
    }

    public void moveUp()
    {
        y--;
        currFrame = UP_FRAMES;
    }

    public void moveDown()
    {
        y++;
        currFrame = DOWN_FRAMES;
    }

    public void moveLeft()
    {
        x--;
        currFrame = LEFT_FRAMES;
    }

    public void moveRight()
    {
        x++;
        currFrame = RIGHT_FRAMES;
    }
    public void step(long timeElapsed)
    {

    }

    public void draw(Graphics g)
    {
        sprite.setCurrAnimFrame(currFrame);
        sprite.draw(g, x, y);
    }
}
```

SimpleController

```
/*
 * SimpleController
 */

package demo;

import java.awt.Graphics;
import java.util.logging.Level;
import java.util.logging.Logger;
import javaPlay.GameEngine;
import javaPlay.GameStateController;
import javaPlay.Keyboard;
import javaPlay.Scene;
import javaPlay.Sprite;

/**
 *
 * @author VisionLab/PUC-Rio
 */
public class SimpleController implements GameStateController
{
    private Sprite playerSprite;
    private Player player;
    private Scene scene;

    public void load()
    {
        try
        {
            playerSprite = new Sprite("tankplayer.png", 8, 32, 32);

            scene = new Scene();
            scene.loadFromFile("scene.scn");

            player = new Player();
            player.setSprite(playerSprite);

            scene.addOverlay(player);
        }
        catch (Exception ex)
        {
            Logger.getLogger(SimpleController.class.getName()).log(Level.SEVERE,  null,
ex);
        }
    }

    public void unload()
    {

    }

    public void start()
    {
        player.x = 92;
        player.y = 114;
    }
    public void step(long timeElapsed)
    {
        Keyboard k = GameEngine.getInstance().getKeyboard();

        if(k.keyDown(Keyboard.UP_KEY) == true)
        {
            player.moveUp();
        }
        if(k.keyDown(Keyboard.DOWN_KEY) == true)
        {
            player.moveDown();
        }
        if(k.keyDown(Keyboard.LEFT_KEY) == true)
        {
            player.moveLeft();
        }
        if(k.keyDown(Keyboard.RIGHT_KEY) == true)
        {
            player.moveRight();
        }

        player.step(timeElapsed);
    }

    public void draw(Graphics g)
    {
        scene.draw(g);
    }

    public void stop()
    {

    }
}
```

EndScreenControl.java

```
/*
 * EndScreenController
 */

package demo;

import java.awt.Graphics;
import java.util.logging.Level;
import java.util.logging.Logger;
import javaPlay.GameEngine;
import javaPlay.GameStateController;
import javaPlay.Keyboard;
import javaPlay.Sprite;

/**
 * @author VisionLab/PUC-Rio
 */
public class EndScreenController implements GameStateController
{
    private Sprite endScreen;

    public void load()
    {
        try
        {
            endScreen = new Sprite("finalback.jpg", 1, 800, 600);
        }
        catch (Exception ex)
        {
            Logger.getLogger(EndScreenController.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void unload()
    {

    }
    public void start()
    {

    }

    public void step(long timeElapsed)
    {
        Keyboard k = GameEngine.getInstance().getKeyboard();

        if(k.keyDown(Keyboard.ENTER_KEY) == true)
        {
            GameEngine.getInstance().setNextGameStateController(Global.START_MENU_CONTROLLER_ID);
        }
    }

    public void draw(Graphics g)
    {
        endScreen.draw(g, 0, 0);
    }


    public void stop()
    {

    }
}
```

Global.java

```
/*
 *  Global
 */

package demo;

/**
 * @author VisionLab/PUC-Rio
 */
public class Global
{
    public static final int START_MENU_CONTROLLER_ID = 0;
    public static final int SIMPLE_CONTROLLER_ID = 1;
    public static final int END_SCREEN_CONTROLLER_ID = 2;
}
```

Main.java

```
/*
 * Main
 */

package demo;

import javaPlay.GameEngine;

public class Main
{
    public static void main(String[] args)
    {
        GameEngine.getInstance().addGameStateController(Global.START_MENU_CONTROLLER_ID,       new
StartMenuController());

        GameEngine.getInstance().addGameStateController(Global.SIMPLE_CONTROLLER_ID,           new
SimpleController());

        GameEngine.getInstance().addGameStateController(Global.END_SCREEN_CONTROLLER_ID,       new
EndScreenController());

        GameEngine.getInstance().setStartingGameStateController(Global.START_MENU_CONTROLLER_ID);

        GameEngine.getInstance().run();
    }
}
```

StartMenuControl.java

```java
/*
 * StartMenuController
 */

package demo;

import java.awt.Graphics;
import java.awt.Point;
import java.util.logging.Level;
import java.util.logging.Logger;
import javaPlay.GameEngine;
import javaPlay.GameStateController;
import javaPlay.Mouse;
import javaPlay.Sprite;

/**
 *
 * @author VisionLab/PUC-Rio
 */
public class StartMenuController implements GameStateController
{
    private Sprite menu;

    public void load()
    {
        try
        {
            menu = new Sprite("startmenu.jpg", 1, 800, 600);
        }
        catch (Exception ex)
        {
            Logger.getLogger(StartMenuController.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void unload()
    {

    }

    public void start()
    {

    }

    public void step(long timeElapsed)
    {
        Mouse m = GameEngine.getInstance().getMouse();

        if(m.isLeftButtonPressed() == true)
        {
            Point p = m.getMousePos();

            if((p.x >= 363) && (p.y >= 353) && (p.x <= 485) && (p.y <= 389))
            {
                GameEngine.getInstance().setNextGameStateController(Global.SIMPLE_CONTROLLER_ID);
            }
        }
    }

    public void draw(Graphics g)
    {
        menu.draw(g, 0, 0);
    }

    public void stop()
    {

    }
}
```

C.3 Demo da Versão 0

Esta versão é a mais completa, contendo um jogador com sprite animado mais elaborado, placar de vidas (Score), dois inimigos e tratamento de colisão (Fig. C3). Nesta seção estão incluídos apenas as novas classes. O pacote javaPlay é omesmo para as versões 0 e 00.

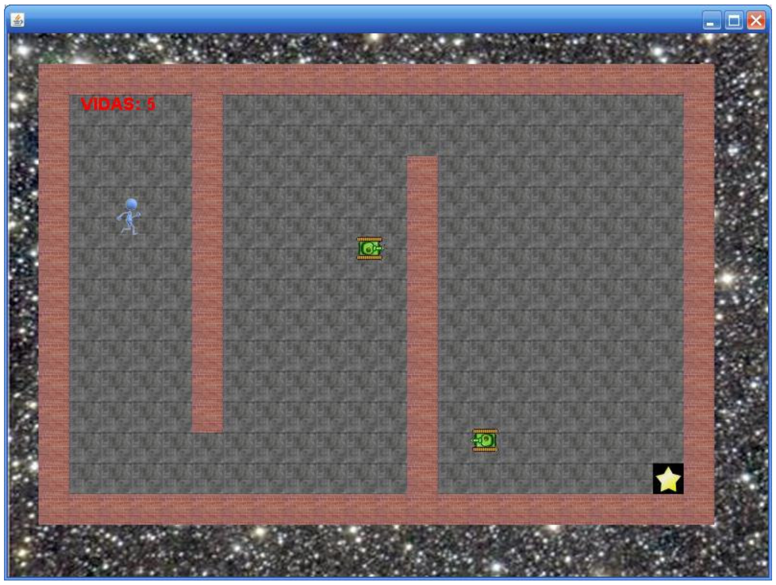

Fig. C3 Janela do demo do motor Versão 0.

Neste jogo, o objetivo é chegar ao alvo em forma de estrela. A cada colisão com um inimigo, o placar geral registra perdas de vida. Quando a vida zera, o programa é fechado (shutdown). Quando o Player alcança o alvo, uma imagem de finalização é apresentada e um toque na tecla “Enter” reinicia o jogo (estas ações são consideradas na classe EndScreenController.java).

Player.java

```
/*
 * Game Object: Player
 */

package demo;

import java.awt.Graphics;
import javaPlay.GameObject;
import javaPlay.Sprite;

/**
 * @author VisionLab/PUC-Rio
 */
public class Player extends GameObject
{
    private Sprite sprite;
    private int currFrame;

    /*  animation with 8 frames - file "tankplayer.png"
    private final int[] UP_FRAMES = {0, 1};
    private final int[] LEFT_FRAMES = {2, 3};
    private final int[] RIGHT_FRAMES = {6, 7};
    private final int[] DOWN_FRAMES = {4, 5};
    private final int SHOW_STEP = 1;
    */

    /* animation with 40 frames - file "walkingplayer.png" */
    private final int[] UP_FRAMES =    { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    private final int[] LEFT_FRAMES =  {10,11,12,13,14,15,16,17,18,19};
    private final int[] RIGHT_FRAMES = {20,21,22,23,24,25,26,27,28,29};
    private final int[] DOWN_FRAMES =  {30,31,32,33,34,35,36,37,38,39};
    private final int SHOW_STEP = 30;
    private int animIndex;
    private int currMoveAnim;
    private int stepCounter;

    private final int MOVE_ANIM_UP = 0;
    private final int MOVE_ANIM_DOWN = 1;
    private final int MOVE_ANIM_LEFT = 2;
    private final int MOVE_ANIM_RIGHT = 3;

    public void setSprite(Sprite sprite)
    {
        this.sprite = sprite;
        animIndex = 0;
        currFrame = UP_FRAMES[animIndex];
        currMoveAnim = MOVE_ANIM_UP;
        stepCounter = 0;
    }

    public void moveUp()
    {
        y--;
        currMoveAnim = MOVE_ANIM_UP;
    }

    public void moveDown()
    {
        y++;
        currMoveAnim = MOVE_ANIM_DOWN;
    }

    public void moveLeft()
    {
        x--;
        currMoveAnim = MOVE_ANIM_LEFT;
    }

    public void moveRight()
    {
        x++;
        currMoveAnim = MOVE_ANIM_RIGHT;
    }

    public void step(long timeElapsed)
    {
        switch(currMoveAnim)
        {
            case MOVE_ANIM_UP:
               if (++stepCounter >= SHOW_STEP)
               {
                  animIndex = (animIndex + 1) % UP_FRAMES.length;
                  currFrame = UP_FRAMES[animIndex];
                  stepCounter = 0;
               }
               break;
            case MOVE_ANIM_DOWN:
               if (++stepCounter >= SHOW_STEP)
               {
                  animIndex = (animIndex + 1) % DOWN_FRAMES.length;
                  currFrame = DOWN_FRAMES[animIndex];
                  stepCounter = 0;
               }
               break;
            case MOVE_ANIM_LEFT:
               if (++stepCounter >= SHOW_STEP)
               {
                  animIndex = (animIndex + 1) % LEFT_FRAMES.length;
                  currFrame = LEFT_FRAMES[animIndex];
                  stepCounter = 0;
               }
               break;
            case MOVE_ANIM_RIGHT:
               if (++stepCounter >= SHOW_STEP)
               {
                  animIndex = (animIndex + 1) % RIGHT_FRAMES.length;
                  currFrame = RIGHT_FRAMES[animIndex];
                  stepCounter = 0;
               }
               break;
        }
    }
```

```
    public void draw(Graphics g)
    {
        sprite.setCurrAnimFrame(currFrame);
        sprite.draw(g, x, y);
    }
}
```

Enemy.java

```
/*
 * ObjectGame: Enemy
 */

package demo;

import java.awt.Graphics;
import java.awt.Point;
import javaPlay.GameObject;
import javaPlay.Sprite;

/**
 * @author VisionLab/PUC-Rio
 */
public class Enemy extends GameObject
{
    private Sprite sprite;
    private Point pathEndPoint1;
    private Point pathEndPoint2;
    private boolean movingToEndPoint2;
    private final int UP_FRAME = 0;
    private final int LEFT_FRAME = 2;
    private final int RIGHT_FRAME = 6;
    private final int DOWN_FRAME = 4;
    private int currFrame;

    public void setSprite(Sprite sprite)
    {
        this.sprite = sprite;
    }

    public void setPath(Point endPoint1, Point endPoint2)
    {
        pathEndPoint1 = endPoint1;
        pathEndPoint2 = endPoint2;
        x = pathEndPoint1.x;
        y = pathEndPoint2.y;
        movingToEndPoint2 = true;
    }

    public void step(long timeElapsed)
    {
        Point endPoint;

        if(movingToEndPoint2 == true)
        {
            endPoint = pathEndPoint2;
        }
        else
        {
            endPoint = pathEndPoint1;
        }

        if(x < endPoint.x)
        {
            x++;
            currFrame = RIGHT_FRAME;
        }
        else if(x > endPoint.x)
        {
             x--;
             currFrame = LEFT_FRAME;
        }
        if(y < endPoint.y)
        {
            y++;
            currFrame = UP_FRAME;
        }
        else if(y > endPoint.y)
        {
            y--;
            currFrame = DOWN_FRAME;
        }

        if((x == endPoint.x) && (y == endPoint.y))
        {
            movingToEndPoint2 = !movingToEndPoint2;
        }
    }

    public void draw(Graphics g)
    {
        sprite.setCurrAnimFrame(currFrame);
        sprite.draw(g, x, y);
    }
}
```

Score.java

```
/*
 * Game Object: Score
 */

package demo;

import java.awt.Graphics;
import javaPlay.GameObject;
import java.awt.Color;
import java.awt.Font;
import java.lang.String;

/**
 *
 * @author VisionLab/PUC-Rio
 */
public class Score extends GameObject
{
    private int scoreCounter = 5;

    public void setScore(int n)
    {
        scoreCounter = scoreCounter + n;
    }

    public int getScore()
    {
        return scoreCounter;
    }

    public void step(long timeElapsed)
    {

    }
    public void draw(Graphics g)
    {
        g.setColor(Color.RED);
        g.setFont(new Font("Arial", Font.BOLD, 18));
        g.drawString("VIDAS: "+String.valueOf(scoreCounter), 80, 110);
    }
}
```

SimpleController.java

```
/*
 * SimpleController
 */

package demo;
```

```
import java.awt.Graphics;
import java.awt.Point;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;
import javaPlay.GameEngine;
import javaPlay.GameStateController;
import javaPlay.Keyboard;
import javaPlay.Scene;
import javaPlay.Sprite;
import javaPlay.TileInfo;

/**
 *
 * @author VisionLab/PUC-Rio
 */
public class SimpleController implements GameStateController
{
    private Sprite playerSprite;
    private Sprite enemySprite;
    private Player player;
    private Enemy[] enemies;
    private Scene scene;
    private final int NUM_OF_ENEMIES = 2;
    private final int TILE_WALL = 1;
    private final int TILE_EXIT = 3;

    private int[] midx;
    private int[] midy;
    private final int UP_MID = 0;
    private final int DOWN_MID = 1;
    private final int LEFT_MID = 2;
    private final int RIGHT_MID = 3;

    private Score score;

    public void load()
    {
        try
        {
//            playerSprite = new Sprite("tankplayer.png", 8, 32, 32);
           playerSprite = new Sprite("walkingplayer.png", 40, 40, 40);

            enemySprite = new Sprite("enemy.png", 8, 32, 32);

            scene = new Scene();
            scene.loadFromFile("scene.scn");

            player = new Player();
            player.setSprite(playerSprite);

            scene.addOverlay(player);
            enemies = new Enemy[NUM_OF_ENEMIES];

            for (int i = 0; i < NUM_OF_ENEMIES; i++)
            {
                enemies[i] = new Enemy();
                enemies[i].setSprite(enemySprite);
                scene.addOverlay(enemies[i]);
            }

            score = new Score();
            scene.addOverlay(score);

            midx = new int[4];
            midy = new int[4];
        }
        catch (Exception ex)
        {
            Logger.getLogger(SimpleController.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void unload()
    {

    }

    public void start()
    {
        player.x = 92;
        player.y = 114;

        Point startPosEnemy1 = new Point();
        Point endPosEnemy1 = new Point();

        startPosEnemy1.x = 251;
        startPosEnemy1.y = 208;
        endPosEnemy1.x = 379;
        endPosEnemy1.y = 208;

        enemies[0].setPath(startPosEnemy1, endPosEnemy1);

        Point startPosEnemy2 = new Point();
        Point endPosEnemy2 = new Point();

        startPosEnemy2.x = 477;
        startPosEnemy2.y = 408;
        endPosEnemy2.x = 666;
        endPosEnemy2.y = 408;

        enemies[1].setPath(startPosEnemy2, endPosEnemy2);
    }

    public void step(long timeElapsed)
    {
        Keyboard k = GameEngine.getInstance().getKeyboard();

        if(k.keyDown(Keyboard.UP_KEY) == true)
        {
            player.moveUp();
        }
        if(k.keyDown(Keyboard.DOWN_KEY) == true)
        {
            player.moveDown();
        }
        if(k.keyDown(Keyboard.LEFT_KEY) == true)
        {
            player.moveLeft();
        }
        if(k.keyDown(Keyboard.RIGHT_KEY) == true)
        {
            player.moveRight();
        }

        player.step(timeElapsed);

        Point playerMin = new Point(player.x, player.y);
        Point playerMax = new Point(player.x + 31, player.y + 31);

        for(int i = 0 ; i < NUM_OF_ENEMIES ; i++)
        {
            enemies[i].step(timeElapsed);

            Point enemyMin = new Point(enemies[i].x, enemies[i].y);
            Point enemyMax = new Point(enemies[i].x + 31, enemies[i].y + 31);

            if(isColliding(playerMin, playerMax, enemyMin, enemyMax) == true)
            {
                // update Score
                score.setScore(-1);
                // back to start point
                player.x = 92;
                player.y = 114;
                // Shutdown if Score = 0
                if (score.getScore() == 0)
                    GameEngine.getInstance().requestShutdown();

                return;
            }
        }

        Vector tiles = scene.getTilesFromRect(playerMin, playerMax);

        for(int i = 0 ; i < tiles.size() ; i++)
        {
            TileInfo tile = (TileInfo)tiles.elementAt(i);

            if((tile.id == TILE_WALL) && (isColliding(playerMin, playerMax,
                    tile.min, tile.max) == true))
            {
                int centerx = playerMin.x/2 + playerMax.x/2;
                int centery = playerMin.y/2 + playerMax.y/2;

                midx[UP_MID] = tile.min.x/2 + tile.max.x/2;
                midy[UP_MID] = tile.min.y;

                midx[DOWN_MID] = tile.min.x/2 + tile.max.x/2;
                midy[DOWN_MID] = tile.max.y;

                midx[LEFT_MID] = tile.min.x;
                midy[LEFT_MID] = tile.min.y/2 + tile.max.y/2;

                midx[RIGHT_MID] = tile.max.x;
                midy[RIGHT_MID] = tile.min.y/2 + tile.max.y/2;

                int dx = midx[UP_MID] - centerx;
                int dy = midy[UP_MID] - centery;
                int nearestMid = 0;
                double nearestDist = Math.sqrt(dx*dx + dy*dy);

                for(int j = 1 ; j < 4 ; j++)
                {
                    dx = midx[j] - centerx;
                    dy = midy[j] - centery;
                    double dist = Math.sqrt(dx*dx + dy*dy);

                    if(dist < nearestDist)
                    {
                        nearestDist = dist;
                        nearestMid = j;
                    }
                }

                switch(nearestMid)
                {
                    case UP_MID:
                        player.y = tile.min.y - 32;
                        break;
                    case DOWN_MID:
                        player.y = tile.max.y;
                        break;
                    case LEFT_MID:
                        player.x = tile.min.x - 32;
                        break;
                    case RIGHT_MID:
                        player.x = tile.max.x;
                        break;
                }
            }
            else if(tile.id == TILE_EXIT)
            {
                GameEngine.getInstance().setNextGameStateController
                        (Global.END_SCREEN_CONTROLLER_ID);
            }
        }
    }

    public void draw(Graphics g)
    {
        scene.draw(g);
    }

    public void stop()
    {

    }

    private boolean isColliding(Point min1, Point max1, Point min2, Point max2)
    {
        if(min1.x > max2.x || max1.x < min2.x)
        {
            return false;
        }
        if(min1.y > max2.y || max1.y < min2.y)
        {
            return false;
        }
        return true;
    }
}
```

C.4 javaPlay para as Versões 00 e 0

GameObject.java

```
/*
 * GameObject
 */

package javaPlay;

import java.awt.Graphics;

/**
 * @author VisionLab/PUC-Rio
 */
public abstract class GameObject
{
    public int x;
    public int y;

    public abstract void step(long timeElapsed);
    public abstract void draw(Graphics g);
}
```

GameEngine.java

```
/*
 * GameEngine
 */

package javaPlay;

import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.awt.GraphicsDevice;
import java.awt.GraphicsEnvironment;
import java.util.*;

/**
 * @author VisionLab/PUC-Rio
 */
public class GameEngine
{
    private GameCanvas canvas;
    private Mouse mouse;
    private Keyboard keyboard;
    private long lastTime;
    private boolean engineRunning;
    private Hashtable gameStateControllers;
    private GameStateController currGameState;
    private GameStateController nextGameState;
    private static GameEngine instance;

    private GameEngine()
    {
        GraphicsEnvironment graphEnv = GraphicsEnvironment.getLocalGraphicsEnvironment();
        GraphicsDevice graphDevice = graphEnv.getDefaultScreenDevice();
        GraphicsConfiguration graphicConf = graphDevice.getDefaultConfiguration();

        canvas = new GameCanvas(graphicConf);

        mouse = new Mouse();
        keyboard = new Keyboard();

        canvas.addMouseListener(mouse);
        canvas.addMouseMotionListener(mouse);
        canvas.addKeyListener(keyboard);

        lastTime = System.currentTimeMillis();
        engineRunning = true;
        gameStateControllers = new Hashtable();
        currGameState = null;
        nextGameState = null;
        instance = null;
    }

    public static GameEngine getInstance()
    {
        if(instance == null)
        {
            instance = new GameEngine();
        }
        return instance;
    }

    public GameCanvas getGameCanvas()
    {
        return canvas;
    }

    public Mouse getMouse()
    {
        return mouse;
    }

    public Keyboard getKeyboard()
    {
        return keyboard;
    }

    public void addGameStateController(int id, GameStateController gs)
    {
        gameStateControllers.put(id, gs);

        gs.load();
    }

    public void removeGameStateController(int id)
    {
        GameStateController gs = (GameStateController)gameStateControllers.get(id);

        gs.unload();
    }

    public void setStartingGameStateController(int id)
    {
        currGameState = (GameStateController)gameStateControllers.get(id);
    }

    public void setNextGameStateController(int id)
    {
        nextGameState = (GameStateController)gameStateControllers.get(id);
    }

    public void requestShutdown()
    {
        engineRunning = false;
    }

    public void run()
    {
        if(currGameState == null)
        {
            return;
        }

        currGameState.start();

        long currentTime;

        while(engineRunning == true)
        {
            currentTime = System.currentTimeMillis();

            currGameState.step(currentTime - lastTime);

            currGameState.draw(canvas.getGameGraphics());

            canvas.swapBuffers();

            if(nextGameState != null)
            {
                currGameState.stop();
                nextGameState.start();

                currGameState = nextGameState;
                nextGameState = null;
            }
        }

        canvas.dispose();
    }
}
```

GameCanvas.java

```
/*
 * GameCanvas
 */

package javaPlay;

import java.awt.Container;
import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.image.BufferStrategy;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JFrame;

/**
 * @author VisionLab/PUC-Rio
 */
public class GameCanvas extends JFrame
{
    private final int defaultScreenWidth = 800;
    private final int defaultScreenHeight = 600;
    private Graphics g;
    private BufferStrategy bf;
    private int renderScreenStartX;
    private int renderScreenStartY;

    public GameCanvas(GraphicsConfiguration gc)
    {
        super(gc);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(defaultScreenWidth, defaultScreenHeight);
        setVisible(true);

        createBufferStrategy(2);

        renderScreenStartX = this.getContentPane().getLocationOnScreen().x;
        renderScreenStartY = this.getContentPane().getLocationOnScreen().y;

        bf = getBufferStrategy();
    }

    public int getRenderScreenStartX()
    {
        return renderScreenStartX;
    }

    public int getRenderScreenStartY()
    {
        return renderScreenStartY;
    }

    public Graphics getGameGraphics()
    {
        g = bf.getDrawGraphics();
        return g;
    }
    public void swapBuffers()
    {
        bf.show();
        g.dispose();
        Toolkit.getDefaultToolkit().sync();
    }
}
```

GameStateController.java

```java
/*
 * GameStateController
 */

package javaPlay;

import java.awt.*;

/**
 * @author VisionLab/PUC-Rio
 */
public interface GameStateController
{
    public void load();
    public void unload();
    public void start();
    public void step(long timeElapsed);
    public void draw(Graphics g);
    public void stop();
}
```

Keyboard.java

```java
/*
 * Keyboard
 */

package javaPlay;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.ArrayList;
import java.util.Hashtable;

/**
 * @author VisionLab/PUC-Rio
 */
public class Keyboard implements KeyListener
{
    private Hashtable keysPressed;

    public static final int UP_KEY = 38;
    public static final int LEFT_KEY = 37;
    public static final int RIGHT_KEY = 39;
    public static final int DOWN_KEY = 40;
    public static final int ESCAPE_KEY = 27;
    public static final int SPACE_KEY = 32;
    public static final int ENTER_KEY = 10;

    public Keyboard()
    {
        keysPressed = new Hashtable();
    }

    public synchronized boolean keyDown(int keyCode)
    {
        return keysPressed.contains(keyCode);
    }

    public void keyTyped(KeyEvent e)
    {

    }

    public synchronized void keyPressed(KeyEvent e)
    {
        if(keysPressed.contains(e.getKeyCode()) == false)
        {
            keysPressed.put(e.getKeyCode(), e.getKeyCode());
        }
    }

    public synchronized void keyReleased(KeyEvent e)
    {
        keysPressed.remove(e.getKeyCode());
    }
}
```

Mouse.java

```
/*
 * Mouse
 */

package javaPlay;

import java.awt.Point;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;

/**
 * @author VisionLab/PUC-Rio
 */
public class Mouse implements MouseMotionListener, MouseListener
{
    private Point mousePos;
    private boolean leftButtonPressed;
    private boolean middleButtonPressed;
    private boolean rightButtonPressed;

    public Mouse()
    {
        mousePos = new Point(0, 0);
        leftButtonPressed = false;
        middleButtonPressed = false;
        rightButtonPressed = false;
    }

    public Point getMousePos()
    {
        return mousePos;
    }

    public boolean isLeftButtonPressed()
    {
        return leftButtonPressed;
    }

    public boolean isMiddleButtonPressed()
    {
        return middleButtonPressed;
    }

    public boolean isRightButtonPressed()
    {
        return rightButtonPressed;
    }

    public void mouseClicked(MouseEvent e)
    {

    }

    public void mousePressed(MouseEvent e)
    {
        switch(e.getButton())
        {
            case MouseEvent.BUTTON1:
                leftButtonPressed = ((e.getModifiersEx() & MouseEvent.BUTTON1_DOWN_MASK) != 0);
                break;
            case MouseEvent.BUTTON2:
                middleButtonPressed = ((e.getModifiersEx() & MouseEvent.BUTTON2_DOWN_MASK) != 0);
                break;
            case MouseEvent.BUTTON3:
                rightButtonPressed = ((e.getModifiersEx() & MouseEvent.BUTTON3_DOWN_MASK) != 0);
                break;
        }
    }

    public void mouseReleased(MouseEvent e)
    {
        switch(e.getButton())
        {
            case MouseEvent.BUTTON1:
                leftButtonPressed = ((e.getModifiersEx() & MouseEvent.BUTTON1_DOWN_MASK) != 0);
                break;
            case MouseEvent.BUTTON2:
                middleButtonPressed = ((e.getModifiersEx() & MouseEvent.BUTTON2_DOWN_MASK) != 0);
                break;
            case MouseEvent.BUTTON3:
                rightButtonPressed = ((e.getModifiersEx() & MouseEvent.BUTTON3_DOWN_MASK) != 0);
                break;
        }
    }

    public void mouseEntered(MouseEvent e)
    {

    }

    public void mouseExited(MouseEvent e)
    {

    }

    public void mouseDragged(MouseEvent e)
    {

    }

    public void mouseMoved(MouseEvent e)
    {
        mousePos = e.getPoint();
    }

}
```

Scene.java

```
/*
 * Scene
 */

package javaPlay;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * @author VisionLab/PUC-Rio
 */
public class Scene
{
    private Image backDrop;
    private Image[] tiles;
    private ArrayList tileLayer;
    private ArrayList overlays;
    private int drawStartX = 0;
    private int drawStartY = 0;
    private final int MAX_SLEEP_COUNT = 30;

    public     void     loadFromFile(String     sceneFile)     throws     InterruptedException,
FileNotFoundException, IOException, Exception
    {
        tileLayer = new ArrayList();
        overlays = new ArrayList();

        BufferedReader input = new BufferedReader(new FileReader(new File(sceneFile)));

        //first read the number of tile images
        String line = input.readLine();

        int numOfTileImages = Integer.parseInt(line, 10);
        tiles = new Image[numOfTileImages];

        int count;

        for(int i = 0 ; i < numOfTileImages ; i++)
        {
            //read each tile image name
            line = input.readLine();

            tiles[i] = Toolkit.getDefaultToolkit().getImage(line);

            count = 0;
            while(tiles[i].getWidth(null) == -1)
            {
                Thread.sleep(1);

                count++;

                if(count == MAX_SLEEP_COUNT)
                {
                    throw new Exception("image could not be loaded");
                }
            }
        }

        //now read the tile set map until the final
        //character is found "%"
        String endTileSet = "%";

        line = input.readLine();

        while(line.equals(endTileSet) != true)
        {
            ArrayList tileLine = new ArrayList();

            String[] tileIndices = line.split(",");

            for(int i = 0 ; i < tileIndices.length ; i++)
            {
                tileLine.add(Integer.parseInt(tileIndices[i]));
            }

            tileLayer.add(tileLine);

            line = input.readLine();
        }

        //now read the backdrop file
        line = input.readLine();

        backDrop = Toolkit.getDefaultToolkit().getImage(line);

        count = 0;
        while(backDrop.getWidth(null) == -1)
        {
            Thread.sleep(1);

            count++;

            if(count == MAX_SLEEP_COUNT)
            {
                throw new Exception("image could not be loaded");
            }
        }
    }

    public void addOverlay(GameObject overlay)
    {
        overlays.add(overlay);
    }

    public void removeOverlay(GameObject overlay)
    {
        overlays.remove(overlay);
    }

    public void setDrawStartPos(int drawStartX, int drawStartY)
    {
        this.drawStartX = drawStartX;
        this.drawStartY = drawStartY;
    }

    public void draw(Graphics g)
    {
        //first clear the scene
        GameCanvas canvas = GameEngine.getInstance().getGameCanvas();

        g.setColor(Color.BLACK);

        g.clearRect(0, 0, canvas.getWidth(), canvas.getHeight());

        //first draw the backdrop
        int startDrawX = canvas.getRenderScreenStartX() - drawStartX;
        int startDrawY = canvas.getRenderScreenStartY() - drawStartY;

        g.drawImage(backDrop, startDrawX, startDrawY, null);

        //now draw the tile set
        int tileWidth = tiles[0].getWidth(null);
        int tileHeight = tiles[0].getHeight(null);

        int line = 0;
        int drawY = startDrawY;

        do
        {
            ArrayList tileLine = (ArrayList)tileLayer.get(line);

            int drawX = startDrawX;

            for(int c = 0 ; c < tileLine.size() ; c++, drawX += tileWidth)
            {
                int idx = (Integer)tileLine.get(c);

                if(idx == 0)
                {
                    continue;
                }

                g.drawImage(tiles[idx-1], drawX, drawY, null);
            }

            drawY += tileHeight;
            line++;

        }while(line < tileLayer.size());

        //finally draw the overlays
        for(int i = 0 ; i < overlays.size() ; i++)
        {
            GameObject element = (GameObject)overlays.get(i);

            element.draw(g);
        }
    }

    public Vector getTilesFromRect(Point min, Point max)
    {
        Vector v = new Vector();

        GameCanvas canvas = GameEngine.getInstance().getGameCanvas();

        int startDrawX = canvas.getRenderScreenStartX() - drawStartX;
        int startDrawY = canvas.getRenderScreenStartY() - drawStartY;

        int tileWidth = tiles[0].getWidth(null);
        int tileHeight = tiles[0].getHeight(null);

        int line = 0;
        int drawY = startDrawY;

        do
        {
            ArrayList tileLine = (ArrayList)tileLayer.get(line);

            int drawX = startDrawX;

            for(int c = 0 ; c < tileLine.size() ; c++, drawX += tileWidth)
            {
                TileInfo tile = new TileInfo();

                tile.id = (Integer)tileLine.get(c);
                tile.min.x = drawX - canvas.getRenderScreenStartX();
                tile.min.y = drawY - canvas.getRenderScreenStartY();
                tile.max.x = drawX - canvas.getRenderScreenStartX() + tileWidth - 1;
                tile.max.y = drawY - canvas.getRenderScreenStartY() + tileHeight - 1;

                if((min.x > tile.max.x) || (max.x < tile.min.x))
                {
                    continue;
                }
                if((min.y > tile.max.y) || (max.y < tile.min.y))
                {
                    continue;
                }

                v.add(tile);
            }

            drawY += tileHeight;
            line++;

        }while(line < tileLayer.size());

        return v;
    }

    public void step(int timeElapsed)
    {
        for(int i = 0 ; i < overlays.size() ; i++)
        {
            GameObject element = (GameObject)overlays.get(i);

            element.step(timeElapsed);
        }
    }
}
```

Sprite.java

```
/*
 * Sprite
 */

package javaPlay;

import java.awt.Graphics;
import java.awt.Image;
import java.awt.Toolkit;

/**
 * @author VisionLab/PUC-Rio
 */
public class Sprite
{
    private Image image;
    private int animFrameCount;
    private int currAnimFrame;
    private int animFrameWidth;
    private int animFrameHeight;
    private int MAX_COUNT = 50;

    public Sprite(String filename, int animFrameCount, int animFrameWidth,
            int animFrameHeight) throws Exception
    {
        image = Toolkit.getDefaultToolkit().getImage(filename);

        int count = 0;

        while(image.getWidth(null) == -1)
        {
            Thread.sleep(1);
            count++;

            if(count == MAX_COUNT)
            {
                throw new Exception("image could not be loaded");
            }
        }
        this.animFrameCount = animFrameCount;
        this.animFrameWidth = animFrameWidth;
        this.animFrameHeight = animFrameHeight;

        this.currAnimFrame = 0;
    }

    public void setCurrAnimFrame(int frame)
    {
        currAnimFrame = frame;
    }

    public void draw(Graphics g, int x, int y)
    {
        GameCanvas canvas = GameEngine.getInstance().getGameCanvas();

        int xpos = canvas.getRenderScreenStartX() + x;
        int ypos = canvas.getRenderScreenStartY() + y;

        g.drawImage(image, xpos, ypos, xpos + animFrameWidth, ypos + animFrameHeight,
                currAnimFrame  *  animFrameWidth,  0,  (currAnimFrame  +  1)  *  animFrameWidth,
animFrameHeight, null);
    }

    private Sprite(Image image, int animFrameCount,
            int currAnimFrame, int animFrameWidth, int animFrameHeight)
    {
        this.image = image;
        this.animFrameCount = animFrameCount;
        this.currAnimFrame = currAnimFrame;
        this.animFrameWidth = animFrameWidth;
        this.animFrameHeight = animFrameHeight;
    }

    public Sprite clone()
    {
        return new Sprite(image, animFrameCount, currAnimFrame,
                animFrameWidth, animFrameHeight);
    }
}
```

TileInfo.java

```
/*
 * TileInfo
 */

package javaPlay;

import java.awt.Point;

/**
 * @author VisionLab/PUC-Rio
 */
public class TileInfo
{
    public int id;
    public Point min;
    public Point max;

    public TileInfo()
    {
       min = new Point();
       max = new Point();
    }
}
```