Creare videogiochi con Java

Golden T Game Engine

www.goldenstudios.or.id

Scaricare e installare la libreria

- Scaricare la libreria da http://www.goldenstudios.or.id.
- Installare anche i file che costituiscono l'help in linea
- cercare il file golden_0_2_3.jar ed aggiungerlo alle libreie nel progetto creato con NetBeans

Struttura della classe principale del gioco

La classe principale di ogni nostro gioco dovra' necessariamente estendere *Game*. Poiche' *Game* e' una classe astratta, dovremo definire **tutti** i suoi metodi astratti, che sono:

- **public abstract void** *initResources()*: in questo metodo vengono caricate tutte le risorse necessarie al gioco, come immagini, file audio ecc. e vengono inizializzati i vari oggetti. Viene chiamato una sola volta, appena la classe viene istanziata.
- **public abstract void** *update(long elapsedTime)*: si occupa di aggiornare in tempo reale il gioco, aggiornando la posizione degli oggetti, cambiando il frame di un animazione ecc.. Il valore long passato indica i millisecondi che devono passare tra un aggiornamento ed il successivo.
- **public abstract void** *render(Graphics2D g)*: questo metodo si occupa di mettere tutto a schermo.

Prima esecuzione

```
import com.golden.gamedev.*;
 import java.awt.*;

public class MyGame1 extends Game {

public void initResources() { }

  public void update(long elapsedTime) { }

  public void render(Graphics2D g) { }

  public static void main(String[] args) {
    GameLoader gameLoader = new GameLoader();
    gameLoader.setup(new MyGame1(), new Dimension(640, 480), false);
    gameLoader.start();
  }
}
```

L'oggetto *gameLoader* si occupa di caricare il gioco, avviando il motore grafico e tutto il necessario.
Il metodo *setup()* accetta tre parametri: un istanza della sotto classe di *Game* che abbiamo appena definito, la risoluzione del gioco (in questo caso 640*480) e un valore booleano. Se questo booleano vale *true*, il gioco e' avviato in modalità schermo intero, altrimenti in modalità finestra.
Dopo aver fatto questo, si può far partire il gioco vero e proprio chiamando il metodo *start()*.

Sprite

Uno SPRITE è un oggetto a cui e' associata un'immagine e una serie di caratteristiche, come la posizione, la dimensione, l'eventuale velocità di spostamento ecc..
Ne esistono diversi tipi:

- **com.golden.gamedev.object.Sprite**: e' la versione più semplice di sprite, rappresenta un oggetto non animato che tuttavia può essere mosso.
- **com.golden.gamedev.object.AnimatedSprite**: rappresenta uno sprite formato da una serie di immagini che formano l'animazione. Estende *Sprite*.
- **com.golden.gamedev.object.sprite.AdvanceSprite**: rappresenta sempre uno sprite animato ma offre funzionalità avanzate per la gestione delle animazioni. Estende *AnimatedSprite*.
- **com.golden.gamedev.object.sprite.VolatileSprite**: sempre uno sprite animato, ma stavolta l'animazione viene visualizzata un'unica volta anzichè essere mandata in loop. E' particolarmente utile nel caso di esplosioni o simili. Estende *AdvanceSprite*.

```
import com.golden.gamedev.*;
import com.golden.gamedev.object.*;

import java.awt.*;
import java.awt.image.*;

public class MyGame2 extends Game
{
    private Sprite plane = null;

    public void initResources()
    {
        // carico l'immagine
        BufferedImage image = getImage("plane1.png");

        // creo lo sprite
        plane = new Sprite(image, 300, 200);
    }

    public void update(long elapsedTime)
    {
        // aggiorno lo sprite
        plane.update(elapsedTime);
    }

    public void render(Graphics2D g)
    {
        // visualizzo a schermo lo sprite aggiornato
        plane.render(g);
    }

    public static void main(String[] args)
    {
        GameLoader gameLoader = new GameLoader();
        gameLoader.setup(new MyGame2(), new Dimension(640, 480), false);
        gameLoader.start();
    }
}
```

Background

La classe *com.golden.gamedev.object.Background* rappresenta un generico "sfondo" per il nostro videogioco. Tutti gli sprite (ed altri oggetti) che fanno parte della stessa area di gioco dovrebbero sempre essere associati ad un oggetto *Background*. Alcune di queste classi sono:

- **com.golden.gamedev.object.background.ColorBackground**: semplice sfondo, riempito con un unico colore.
- **com.golden.gamedev.object.background.ImageBackground**: usa come sfondo un immagine.
- **com.golden.gamedev.object.background.TileBackground**: questo sfondo suddivide la mappa in "piastrelle" (tile), ognuna delle quali ammette più livelli di oggetti. Ogni "piastrella" e' un pezzo di immagine.
- **com.golden.gamedev.object.background.ParallaxBackground**: questo e' uno sfondo creato sovrapponendo diversi altri sfondi.

```
import com.golden.gamedev.*;
import com.golden.gamedev.object.*;
import com.golden.gamedev.object.background.*;

import java.awt.*;
public class MyGame3 extends Game
{
   private ImageBackground background = null;
   private Sprite sprite = null;
   public void initResources()
   {
      // istanzio uno sfondo con immagine
      background = new ImageBackground(getImage("background.jpg"));

      // sprite dell'aereoplanino
      sprite = new Sprite(getImage("plane1.png"));

      // imposto la posizione dell'aereoplanino
      sprite.setLocation(320, 240);

      // associo uno sfondo allo sprite
      sprite.setBackground(background);
   }
   public void update(long elapsedTime)
   {
      // aggiorno sfondo e sprite
      background.update(elapsedTime);
      sprite.update(elapsedTime);
   }
   public void render(Graphics2D g)
   {
      // metto a schermo tutto
      background.render(g);
      sprite.render(g);
   }

   public static void main(String[] args)
   {
      GameLoader game = new GameLoader();
      game.setup(new MyGame3(), new Dimension(640,480), false);
      game.start();
   }
}
```

Muovere uno sprite

La classe *Sprite* mette a disposizione diversi metodi per muovere gli oggetti sullo schermo. I piu' utilizzati sono sostanzialmente questi due:

- **public void** *setSpeed(double vx, double vy)*: imposta la velocita' di movimento dello sprite. Un valore positivo di *vx* fa muovere lo sprite verso destra, un valore positivo di *vy* lo fa muovere verso il basso. Valori negativi lo fanno andare verso sinistra o verso l'alto.
- **public void** *move(double dx, double dy)*: muove lo sprite di *dx* pixel in orizzontale e *dy* pixel in verticale. Per il segno dei parametri, valgono le considerazione di *setSpeed()*

La differenza tra i due metodi e' che *move()* muove lo sprite dei pixel indicati e poi questo si ferma mentre *setSpeed()* continua a muovere lo sprite fino a che entrambe le velocita' *vx* e *vy* non vengono azzerate (quindi il movimento continuera' anche se smettete di premere il bottone, a meno che non abbiate dato le istruzioni del caso).

Per orientarvi sul valore da assegnare a *vx* e *vy* tenete conto di questa proporzione: se una velocita' viene impostata a *n* significa che lo sprite si muovera' di (*n* * 1000) pixel al secondo in quella direzione.

```
import com.golden.gamedev.*;
import com.golden.gamedev.object.*;
import com.golden.gamedev.object.background.*;

import java.awt.*;
import java.awt.event.*;

public class MyGame4 extends Game
{
    private Background background = null;
    private Sprite     sprite = null;
    public void initResources()
    {
      background = new ImageBackground(getImage("background.jpg"));
      sprite = new Sprite(getImage("plane1.png"));
      sprite.setLocation(100, 100);
      sprite.setBackground(background);
    }
    public void update(long elapsedTime)
    {
      background.update(elapsedTime);
      sprite.update(elapsedTime);

      // a seconda del tasto premuto muovo lo sprite dell'aereoplano
      // non usando il costrutto 'else if' permetto i movimenti in
      // diagonale
      // ad ogni iterazione lo sprite si muove di 5 pixel nella
      // direzione indicata dal tasto premuto

      if (keyDown(KeyEvent.VK_UP))
      {
        sprite.move(0, -5);
      }
      if (keyDown(KeyEvent.VK_DOWN))
      {
        sprite.move(0, 5);
      }
      if (keyDown(KeyEvent.VK_RIGHT))
      {
        sprite.move(5, 0);
      }
      if (keyDown(KeyEvent.VK_LEFT))
      {
        sprite.move(-5, 0);
      }
    }
    public void render(Graphics2D g)
    {
      background.render(g);
      sprite.render(g);
    }
    public static void main(String[] args)
    {
      GameLoader game = new GameLoader();
      game.setup(new MyGame4(), new Dimension(640,480), false);
      game.start();
    }
}
```

Sprite animati

La libreria GTGE fornisce diverse classi per la gestione di animazioni:

- **com.golden.gamedev.object.AnimatedSprite**: e' uno sprite formato da piu' immagini (che danno origine all'animazione).
- **com.golden.gamedev.object.sprite.AdvanceSprite**: rappresenta sempre uno sprite animato ma offre funzionalita' avanzate per la gestione delle animazioni.
- **com.golden.gamedev.object.sprite.VolatileSprite**: sempre uno sprite animato, ma stavolta l'animazione viene visualizzata un'unica volta anziche' essere mandata in loop. E' particolarmente utile nel caso di esplosioni o simili.

AnimatedSprite

```
...
private  AnimatedSprite  sprite; // riferimento allo sprite animato

 ...
 // crea lo sprite
sprite = new AnimatedSprite(getImages("c:/butta/plane2.png", 3, 1), 0, 0);

// imposta il ritardo tra un frame ed il successivo
sprite.getAnimationTimer().setDelay(100);

// imposto i frame dell'animazione
sprite.setAnimationFrame(0, 2);

// animo lo sprite
sprite.setAnimate(true);                    // se passo false lo sprite non si vedrà

// metto in loop l'animazione
sprite.setLoopAnim(true);
 ...
```

```
import com.golden.gamedev.*;
import com.golden.gamedev.object.background.*;
import com.golden.gamedev.object.*;
import java.awt.*;
public class AnimatedSpriteDemo extends Game
{
        private AnimatedSprite sprite;
        private Background background;

        public void initResources()
        {
                background = new ColorBackground(Color.blue);

                // crea lo sprite
                sprite = new AnimatedSprite(getImages("plane2.png", 3, 1),
100, 100);

                // imposta il ritardo tra un frame ed il successivo
                sprite.getAnimationTimer().setDelay(50);

                // imposto i frame dell'animazione
                sprite.setAnimationFrame(0, 2);

                // animo lo sprite
                sprite.setAnimate(true);

                // metto in loop l'animazione
                sprite.setLoopAnim(true);
        }

        public void update(long elapsedTime)
        {
                background.update(elapsedTime);
                sprite.update(elapsedTime);
        }

        public void render(Graphics2D g)
        {
                background.render(g);
                sprite.render(g);
        }

        public static void main(String[] args)
        {
                GameLoader gameLoader = new GameLoader();
                gameLoader.setup(new AnimatedSpriteDemo(), new
Dimension(640, 480), false);
                gameLoader.start();
        }
}
```

Effetti sonori

GTGE fornisce due metodi per inserire suoni nel nostro videogioco:

- **public int** *playMusic(String)*
- **public int** *playSound(String)*
- La stringa richiesta dai due metodi e' ovviamente il percorso ai file sonori.
- La differenza tra i due metodi e' che *playMusic* manda l'effetto sonoro in loop, mentre *playSound* esegue il suono solo una volta.
  Inoltre cambia il formato audio: *playMusic* vuole un file *.mid* mentre *playSound* vuole un *.wav*.

Gruppi di sprite

- Grazie ai gruppi e' possibile tenere separati i diversi elementi del nostro gioco semplificandone la gestione e alleggerendo il codice.
- I gruppi di sprite inoltre, ci permettono di gestire le collisioni tra sprite, utilissime nel caso dei videogiochi.

```
import java.awt.*;
import com.golden.gamedev.*;
import com.golden.gamedev.object.*;
import com.golden.gamedev.object.background.*;
public class MyGame extends Game {
    Background   background;
    SpriteGroup  PLAYER_GROUP;
    public void initResources() {
        background = new ColorBackground(Color.BLUE);
        PLAYER_GROUP = new SpriteGroup("Player Group");
        PLAYER_GROUP.setBackground(background);
        PLAYER_GROUP.add(new Sprite(getImage("player.png")));
        PLAYER_GROUP.add(new Sprite(getImage("player.png")));
        PLAYER_GROUP.add(new Sprite(getImage("player.png")));
        PLAYER_GROUP.add(new Sprite(getImage("player.png")));
        PLAYER_GROUP.add(new Sprite(getImage("player.png")));
        PLAYER_GROUP.add(new Sprite(getImage("player.png")));
        PLAYER_GROUP.add(new Sprite(getImage("player.png")));
        PLAYER_GROUP.add(new Sprite(getImage("player.png")));
    }
    public void update(long elapsedTime) {
        background.update(elapsedTime);
        PLAYER_GROUP.update(elapsedTime);
    }
    public void render(Graphics2D g) {
        background.render(g);
        PLAYER_GROUP.render(g);
    }
    public static void main(String[] args) {
        GameLoader game = new GameLoader();
        game.setup(new MyGame(), new Dimension(640,480), false);
        game.start();
    }
}
```

```
import java.awt.*;
import com.golden.gamedev.*;
import com.golden.gamedev.object.*;
import com.golden.gamedev.object.background.*;
public class MyGame extends Game {
    Background   background;
    SpriteGroup  PLAYER_GROUP, PLAYER_SHOT_GROUP, ENEMY_GROUP,
                 ENEMY_SHOT_GROUP, ASTEROID_GROUP, BONUS_GROUP;
    public void initResources() {
        background = new ColorBackground(Color.BLUE);
        PLAYER_GROUP       = new SpriteGroup("Player Group");
        PLAYER_SHOT_GROUP = new SpriteGroup("Player Shot Group");
        ENEMY_GROUP        = new SpriteGroup("Enemy Group");
        ENEMY_SHOT_GROUP   = new SpriteGroup("Enemy Shot Group");
        ASTEROID_GROUP     = new SpriteGroup("Asteroid Group");
        BONUS_GROUP        = new SpriteGroup("Bonus Group");
        PLAYER_GROUP.setBackground(background);
        PLAYER_SHOT_GROUP.setBackground(background);
        ENEMY_GROUP.setBackground(background);
        ENEMY_SHOT_GROUP.setBackground(background);
        ASTEROID_GROUP.setBackground(background);
        BONUS_GROUP.setBackground(background);
    }
    public void update(long elapsedTime) {
        background.update(elapsedTime);
        PLAYER_GROUP.update(elapsedTime);
        PLAYER_SHOT_GROUP.update(elapsedTime);
        ENEMY_GROUP.update(elapsedTime);
        ENEMY_SHOT_GROUP.update(elapsedTime);
        ASTEROID_GROUP.update(elapsedTime);
        BONUS_GROUP.update(elapsedTime);
    }
    public void render(Graphics2D g) {
        background.render(g);
        PLAYER_GROUP.render(g);
        PLAYER_SHOT_GROUP.render(g);
        ENEMY_GROUP.render(g);
        ENEMY_SHOT_GROUP.render(g);
        ASTEROID_GROUP.render(g);
        BONUS_GROUP.render(g);
    }
    public static void main(String[] args) {
        GameLoader game = new GameLoader();
        game.setup(new MyGame(), new Dimension(640,480), false);
        game.start();
    }
}
```

Collisioni

Ci sono diverse classi di collisioni, ognuna adatta a rilevare un certo tipo di collisione:

- **BasicCollisionGroup** controlla solo se la collisione c'e' stata o meno
- **CollisionGroup** controlla se la collisione c'e' stata o meno e fornisce qualche informazione supplementare
- **PreciseCollisionGroup** da utilizzare quando occorre sapere con precisione la posizione degli sprite e altre informazioni
- **AdvanceCollisionGroup** sistema avanzato di rilevamento, registra collisioni verso diversi sprite contemporaneamente e con molta precisione
- **CollisionBounds** registra le collisioni coi i bordi dello schermo

Se utilizziamo molti (troppi!) gruppi il codice potrebbe diventare presto illeggibile. Si puo' risolvere questo problema creando gruppi di gruppi. La classe *PlayField* serve proprio a questo.

```
import java.awt.*;
import com.golden.gamedev.*;
import com.golden.gamedev.object.*;
import com.golden.gamedev.object.background.*;
import com.golden.gamedev.object.collision.*;
public class MyGame extends Game {
    PlayField    playfield;
    Background   background;
    SpriteGroup  PLAYER_GROUP, ENEMY_GROUP;
    public void initResources() {
        background = new ColorBackground(Color.BLUE, 1024, 768);
        playfield = new PlayField(background);
        PLAYER_GROUP = playfield.add(new SpriteGroup("Player Group"));
        ENEMY_GROUP  = playfield.add(new SpriteGroup("Enemy Group"));
        PLAYER_GROUP.add(new Sprite(getImage("player.png")));
        PLAYER_GROUP.add(new Sprite(getImage("player.png")));
        playfield.addCollisionGroup(PLAYER_GROUP, ENEMY_GROUP,
                                    new MyCollision());
    }
    public void update(long elapsedTime) {
        playfield.update(elapsedTime);
    }
    public void render(Graphics2D g) {
        playfield.render(g);
    }
    public static void main(String[] args) {
        GameLoader game = new GameLoader();
        game.setup(new MyGame(), new Dimension(640,480), false);
        game.start();
    }
}
class MyCollision extends BasicCollisionGroup {
    public void collided(Sprite s1, Sprite s2) {
        s1.setActive(false);
        s2.setActive(false);
    }
}
```

```
import java.awt.*;
import com.golden.gamedev.*;
import com.golden.gamedev.object.*;
import com.golden.gamedev.object.background.*;
import com.golden.gamedev.object.collision.*;
public class MyGame extends Game {
    Background   background;
    SpriteGroup  PLAYER_SHOT_GROUP,
    SpriteGroup  ENEMY_GROUP;
    CollisionManager collisionType;
    public void initResources() {
        background = new ColorBackground(Color.BLUE);
        PLAYER_SHOT_GROUP = new SpriteGroup("Player Shot Group");
        ENEMY_GROUP = new SpriteGroup("Enemy Group");
        collisionType = new MyCollision();
        collisionType.setCollisionGroup(PLAYER_SHOT_GROUP, ENEMY_GROUP);
    }
    public void update(long elapsedTime) {
        background.update(elapsedTime);
        PLAYER_SHOT_GROUP.update(elapsedTime);
        ENEMY_GROUP.update(elapsedTime);
        collisionType.checkCollision();
    }
    public void render(Graphics2D g) {
        background.render(g);
        PLAYER_SHOT_GROUP.render(g);
        ENEMY_GROUP.render(g);
    }
    public static void main(String[] args) {
        GameLoader game = new GameLoader();
        game.setup(new MyGame(), new Dimension(640,480), false);
        game.start();
    }
}
class PMyCollision extends BasicCollisionGroup {
    public void collided(Sprite s1, Sprite s2) {
        // fa scomparire i 2 sprite che si sono scontrati
        s1.setActive(false);
        s2.setActive(false);
    }
}
```