|  |
| --- |
| aplu.ch |
| GameGrid Tutorial |
| 10/08/2014 |

|  |
| --- |
| Source: http://www.aplu.ch/home/apluhomex.jsp?site=45  01.10.2012 |

Tutorial for GameGrid

An Education Oriented Java Package - For Developing Computer Games

**The JGameGrid Framework**

*JGameGrid*is a class library specially designed for programming courses with a focus on object oriented programming (OOP). The concept follows the well-known educational principles to keep the learning curve gently and motivate the student by analyzing, developing and playing computer based games. The recent research in **game based learning** (GBP) shows that using games in the classroom makes learning easier and less frustrating. Students may be so amused that they even do not realize they are learning. We want to make programming a pleasure and not a frustration.

When creating computer games you will encounter many classical problems of program design and implementation, and, what is most important for us, you realize how modern programming concepts like OOP is imperative to reduce the complexity of just the most simple game. The game figures are concrete **realizations** of software **objects** and it is natural to model their **state**and **behavior** by **instance variables (attributes, fields)** and **methods**.

Every kid knows playing video games is fun, but making them is serious business. Until now game programming was just too complicated for beginners and could badly serve in introductory programming courses. *JGameGrid* reduces the effort to create game-oriented Java applications by an order of magnitude compared to Java-API programming, without hiding essential parts of the application code or restrict the student by using a specific integrated IDE or a particular game window. With *JGameGrid* you may use your favorite programming environment, from very simple to very professional, and it us up to you how to design the user interface. *JGameGrid* gives you an important support for creating two-dimensional computer games but does not limit your imagination and liberty.

**Important Features:**

* Choice between simple predefined window or integration of the JGameGrid bean component into a standard application using any IDE/GUI builder
* Easy generation of applets. No need for a special applet thread
* Automatic creation of a game playground based on grid cells of any size down to one pixel
* Active rendering to ensure periodic game actions with little influence by other system tasks
* Integrated animation machine using an internal game thread, no need of user threads
* Game objects (actors) with one or more sprite images, automatically rotated when the  
  moving direction changes. To increase performance, rotated images are kept in memory
* Sophisticated collision detection using fast algorithms, like the Separation Axis Theorem SAT). User definable collision areas: rectangle, circle, line and point. Free positioning  
  of the collision area relative to the image, but following the sprite image when it moves or rotates
* Background image with free positioning relative to the playground
* Background off-screen buffer with Java2D graphics support for drawing calculated figures and text
* Tile map with image preload, free positioning relative to the playground and collision detection between tiles and actors
* Button actor classes for push buttons, toggle buttons, check buttons and radio buttons  
  based on button event callbacks invoke from a separate thread
* Dynamically created actor sprites from BufferedImages using Java2D
* Integration of sound clips with volume control and start/stop/resume/repeat options.  
  Support for WAV and MP3 (with additional free jar libraries)
* Detailed JavaDoc of all public classes and methods
* Typical sample applications distributed like basic versions of PacMan, Sokoban, Lunar Lander, Game of Life, etc.
* Demo version of source code available ([download](http://www.aplu.ch/home/download/JGameGridBase.zip))
* *-> Source code in a separate folder (“GameGridTutorial”)*
* Free for educational institutions and personal use.

*The concept and some method names of JGameGrid are heavily influenced by another great game framework*[*GREENFOOT*](http://www.greenfoot.org/)*, developed at the University of Kent and the Deakin University, Melbourne, but no code from the Greenfoot source distribution is included in our library. While Greenfoot is a complete graphical based learning system with its own development environment based on BlueJ, JGameGrid is a Java library that can be used with any IDE and GUI builder.*

**Design principle explained by an example**

In most cases the user application class extends the Actor class and overwrites the methodact() that is called periodically by the game thread when the application is run. As with GREENFOOT a standard navigation area may be displayed containing the buttons *Step*,*Run/Pause*, *Reset* and a slider to select the *simulation period*. This makes the skeleton of a game extremely simple. In the following example a fish moves back and forth in a reef playground.

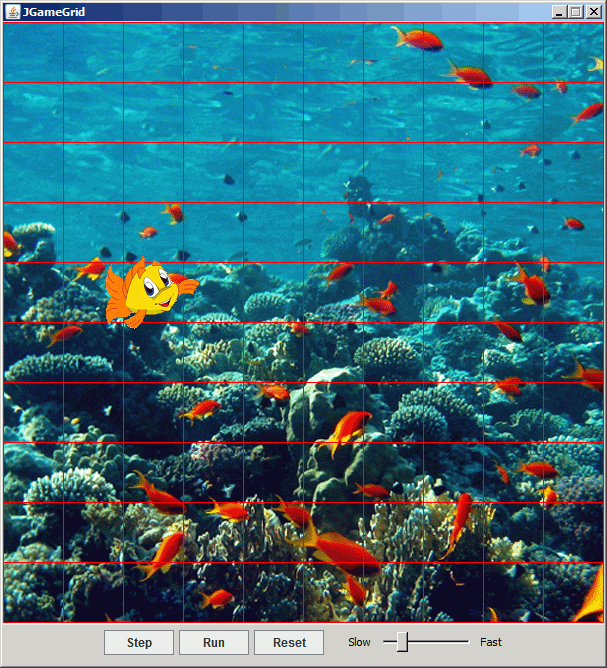
Following the basic principles of OOP, we first declare a class *Fish* that is a subclass of *Actor*. Because we override its method *act()* that is automatically called in every simulation cycle by the underlying animation machine, we move the fish to the neighbour cell just by calling *move()*. When the fish is at the border of the playground it must turn 180 degrees. All needed methods are inherited from the class Actor, so Fish becomes a rich class without great effort.

|  |
| --- |
| import ch.aplu.jgamegrid.\*;  public class Fish extends Actor {   public Fish()   {     super("sprites/nemo.gif");   }   public void act()   {     move();     if (!isMoveValid())       turn(180);   } } |

For the sake of simplicity the application class*MyGame*is derived from the class *GameGrid* and uses one of its constructors to design the visible window with 10 horizontal and 10 vertical cells of size 60 pixels. Red grid lines are displayed and a background image from the file *reef.gif*in the subdirectory *sprites* shows the rich underwater world from the Red Sea. Then a instance of Fish called *nemo* is created and added to the playground in cell location 2, 4. Finally *show()* displays the window.

|  |
| --- |
| import ch.aplu.jgamegrid.\*;  public class MyGame extends GameGrid {   public MyGame()   {     super(10, 10, 60, java.awt.Color.red, "sprites/reef.gif");     Fish nemo = new Fish();     addActor(nemo, new Location(2, 4));     show();   }    public static void main(String[] args)   {     new MyGame();   } } |

The result of these few lines of code is a full-fledged animation program. Enjoy!



[Execute](http://www.aplu.ch/webstart/mygame.jnlp) the program locally using WebStart.

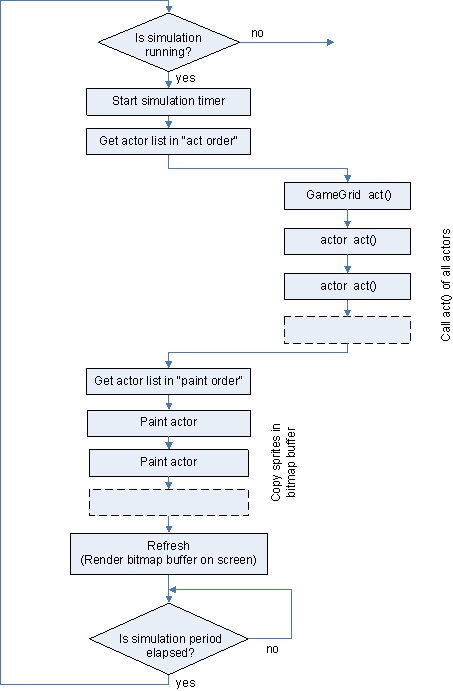
**How to use the navigation area:**

Clicking the *Run* button **starts** the animation and the button text changes to *Pause*. Clicking*Pause* **halts** the execution and the button text changes to *Run*. Clicking it **resumes** the animation. Clicking *Step* performs a **single animation step**. Clicking *Reset* restores the **initial situation**. The slider changes the time between two consecutive calls of *act()*, called**simulation period**. It works even while the animation is running.

**Important remarks about the design strategy:**

Many of the methods of *JGameGrid* can be implemented more or less easily by the user based on a few elementary library methods. This would be a good exercise for beginners, but we think that there is no need (and time) to reinvent the wheel. The student should better spend his effort in his specific application development.

JGameGrid's simulation thread "serves" actors by sequentially calling their *act()* method. The order of this sequence may be modified, but all actors must collaborate friendly and not execute long lasting code that would cause starvation of the following actor. The following flow diagram visualizes the principle.



* <http://www.aplu.ch/home/apluhomex.jsp?site=45> a lot more pages of tutorial and samples