

Using a GameEngine with GTGE

John Wagner

Introduction

This tutorial uses the Golden T Game Engine (GTGE) for Java. This excellent and free gaming engine can be found at <http://goldenstudios.or.id/>

Source code for the tutorial should be included in the archive you downloaded. It is in the file gameenginutorial.jar.

What we're going to learn here is how to make use of the GTGE GameEngine class, which is essentially a game controller. You might be wondering why this necessary, why not just use a Game class and be done with it? It's a good question, and it's best answered by building a simple game and then taking it to the next level...

The Next Big Thing

We're writing a game called "Eraser Head" – it's about a guy with an eraser on his head who runs around cleaning up screens. Sounds like a big hit! Let's get started...

At first our vision of Eraser Head is pretty simple: a one screen game and one character with an eraser on his head. See Figure 1 - Our Hero.



Figure 1 - Our Hero

Version 0.1

Let's create this masterpiece by extending a Game object and implementing the required abstract methods. Let's also add the code to create a background and a sprite and code to move the sprite around.

All together now:

```
package eraserhead;

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

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

public class EraseLevelOne extends Game
{
    private Sprite hero;
    private Background backgr;

    public static void main(String[] args)
    {
```

```

        GameLoader game = new GameLoader();
        game.setup(new EraseLevelOne(), new Dimension(640, 480), false);
        game.start();
    }

    @Override
    public void initResources()
    {
        hero = new Sprite(getImage("resources/eraserguy.png"));
        backgr = new ColorBackground(Color.WHITE, 640, 480);
        hero.setBackground(backgr);
    }

    @Override
    public void update(long elapsedTime)
    {
        backgr.update(elapsedTime);
        hero.update(elapsedTime);

        if(keyDown(KeyEvent.VK_ESCAPE))
        {
            finish();
        }

        if(keyDown(KeyEvent.VK_LEFT))
        {
            hero.setX(hero.getX()-1);
        }

        if(keyDown(KeyEvent.VK_RIGHT))
        {
            hero.setX(hero.getX()+1);
        }

        if(keyDown(KeyEvent.VK_UP))
        {
            hero.setY(hero.getY()-1);
        }

        if(keyDown(KeyEvent.VK_DOWN))
        {
            hero.setY(hero.getY()+1);
        }
    }

    @Override
    public void render(Graphics2D g)
    {
        backgr.render(g);
        hero.render(g);
    }
}

```

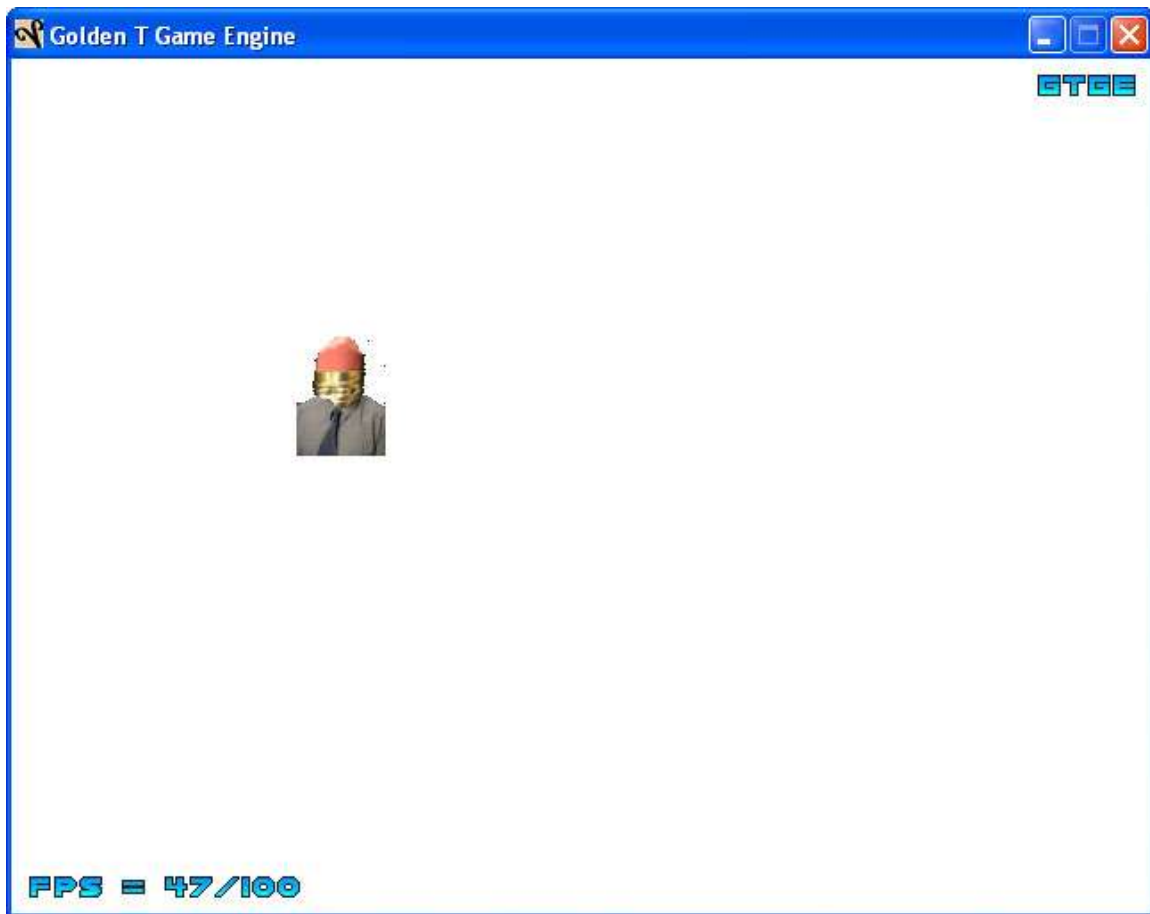


Figure 2 - Screenshot of Eraser Head in action (note to self: probably put this one on the box)

OK, now we have a COOL game! But we need to add some features to make it really and truly complete. Most people expect more from their games. It's bad enough that we aren't writing "EraserHead 3D – The Legions of Death and Destruction", we're leaving things out that people expect of their games, even in their run-of-the-mill 2D EraserHead type games.

I mean wouldn't it be sweet if we had a title screen where we could show a picture of our hero, and show our copyright and title of the game? What about instructions for playing our game?



Version 0.2

Providing for completely different modes of operation in a game like this really has the potential to mess up your code. Your Game class not only has to implement the details of the game, it has to keep track of what “mode” your game is in and act accordingly.

Think of the possibilities:

- If the user uses the arrow keys while displaying the Intro screen, there is no hero to move around. Same goes for the Help screen.
- While playing the game, you don’t want to display your Intro Screen or help screen.

This means you’ll need special cases in your update() method and your render() method. And when you start adding gameplay features to your game, you’ll need to make sure you only add them in the right place.

Here is what it would look like if we did it all in our EraseLevelOne class:

```
package eraserhead;

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

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

public class EraseLevelOne extends Game
{
    private Sprite hero;
    private Background backgr;

    enum GameMode
    {
        INTRO, HELP, PLAY
    };
}
```

```

private GameMode mode = GameMode.INTRO;

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

@Override
public void initResources()
{
    hero = new Sprite(getImage("resources/eraserguy.png"));
    backgr = new ColorBackground(Color.WHITE, 640, 480);
    hero.setBackground(backgr);
}

@Override
public void update(long elapsedTime)
{
    backgr.update(elapsedTime);

    if (mode == GameMode.PLAY)
    {
        hero.update(elapsedTime);

        if (keyDown(KeyEvent.VK_ESCAPE))
        {
            mode = GameMode.INTRO;
        }

        if (keyDown(KeyEvent.VK_LEFT))
        {
            hero.setX(hero.getX() - 1);
        }

        if (keyDown(KeyEvent.VK_RIGHT))
        {
            hero.setX(hero.getX() + 1);
        }

        if (keyDown(KeyEvent.VK_UP))
        {
            hero.setY(hero.getY() - 1);
        }

        if (keyDown(KeyEvent.VK_DOWN))
        {
            hero.setY(hero.getY() + 1);
        }
    }
    else if(mode == GameMode.HELP)
    {
        if (keyPressed(KeyEvent.VK_ESCAPE))
        {
            mode = GameMode.INTRO;
        }
    }
    else if(mode == GameMode.INTRO)
    {
        hero.setLocation(300, 100);

        if (keyPressed(KeyEvent.VK_ESCAPE))
        {
            finish();
        }

        if (keyDown(KeyEvent.VK_SPACE))
        {
            mode = GameMode.PLAY;
        }

        if (keyDown(KeyEvent.VK_F1))
        {

```

```

        mode = GameMode.HELP;
    }
}

@Override
public void render(Graphics2D g)
{
    backgr.render(g);

    if(mode == GameMode.PLAY)
    {
        hero.render(g);
    }
    else if(mode == GameMode.HELP)
    {
        g.setColor(Color.BLUE);
        g.drawString("Use your arrow keys to move EraseHead around", 20, 20);
        g.drawString("Press ESC to return to main screen", 20, 50);
    }
    else if(mode == GameMode.INTRO)
    {
        g.setColor(Color.BLUE);
        g.drawString("Eraser Head - Copyright (c) Wicked Cool Games", 20, 20);
        g.drawString("Press F1 for Help", 20, 50);
        g.drawString("Press SPACE to Play", 20, 70);
        hero.render(g);
    }
}
}

```

Wow, is that ugly or what? Look at that update() method! Not good and on its way to getting worse...

Can it get worse?

Oh yes sir it can! Just think, we haven't implemented much of the game logic yet, bonus rounds, challenge levels, on and on and on it goes. We've got two methods that implement 3 different modes of play already and that thing is oinking!

GameEngine to the rescue

Lucky for us GTGE has a class to cure all that ails us and that class is the GameEngine.

GameEngine extends Game and implements the initResources(), update() and render() methods of the abstract Game class. The methods it provides do nothing but fulfill the requirement of extending an abstract class.

Of those three abstract methods that GameEngine does implement, we might want to override initResources() to initialize game global resources, like a score counter, or something that needs to maintain state between all of the various screens of a game, open a network connection, that type of thing. Such variables would be members of your GameEngine class.

GameEngine itself is an abstract class, because it contains an abstract method called getGame(), which accepts an int as its only argument and returns a reference to a GameObject. So we'll need to implement the getGame() method in our class that extends GameEngine.

If all of the above sounds confusing don't feel bad. Heck, I just wrote it and I'm confused.

Maybe some code might help:

```
package eraserhead;

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

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

    @Override
    public GameObject getGame(int gameID)
    {
        return null;
    }
}
```

Notice that the `getGame()` method simply returns null right now – we're going to fix that in a minute. First we need to set up some things so our `EraseGameEngine` class can keep track of its three modes of operation.

We'll add the following constants to our `EraseGameEngine` class to keep track of our modes. Since `getGame()` deals with int's that's what we'll have to use:

```
public static final int INTRO = 0;
public static final int HELP = 1;
public static final int PLAY = 2;
```

Let's add a score variable. To keep things simple, we'll make it package visible. Ideally we would use a getter/setter.

```
long score = 0;
```

The reason for package visibility will become apparent in a moment.

Now we need to modify our `getGame()` method, but before we do, we have to change our `EraseLevelOne` class and we also need to create classes for our Intro screen and Help screen.

Let's fix the `EraseLevelOne` class first. There are a number of changes to make. They aren't difficult but they need to be done in order to support the `GameEngine`.

We need to extend `GameObject` instead of `Game`:

```
public class EraseLevelOne extends GameObject
```

This is because the `getGame()` method in `GameEngine` returns a `GameObject`, not a `Game`.

Unlike `Game`, `GameObject` doesn't have a default constructor, so we need to call the only constructor `GameObject` does have. To do this, we need a default constructor ourselves... but what would make more sense is a constructor that accepts a `GameEngine` reference as its argument and passes it off to the `GameObject` constructor that does the same:

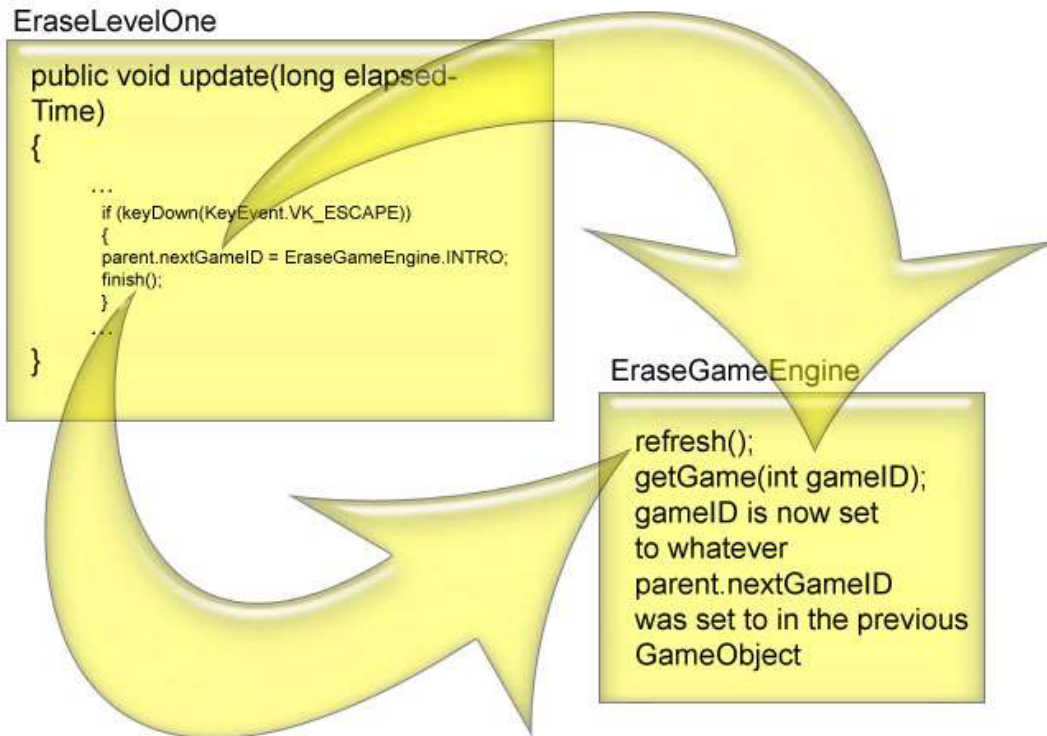
```
public EraseLevelOne(GameEngine parent)
{
    super(parent);
}
```

The parent that we're passing in is a reference to the `GameEngine` object that is running the whole game. This becomes the `GameObject.parent` variable of the `GameObject` that we extended. It's what we can use to get at game global variables, like the score! Consider this change to the `render()` method:

```
g.drawString("Score: " + ((EraseGameEngine) parent).score, 20, 400);
```

That is accessing the score variable from the parent, which is an `EraseGameEngine` (point being the cast is OK). The score variable is package visible, we're in the same package, life is good.

The key to making this work is how you leave your `GameObject` and return control to the `GameEngine`. When the user presses the key to quit (or something happens in the game, like all the lives are lost, or level is solved, etc...) and you need to move to another screen (think `GameObject`), you set the variable `parent.nextGameID` to the ID of the next game. You then call the `finish()` method, which returns control back to the parent `GameEngine` which then calls its `refresh()` method and then calls the `getGame()` method to get the next `GameObject`. I would draw a picture of this, but I'm really scared it might look funny. OK, I lied, I drew a picture and it's not so bad...



Here is a piece of code from the `EraseLevelOne.update()` method. If the user presses the Escape key, the `parent.nextGameID` variable is set to the Intro screen:

```
if (keyDown(KeyEvent.VK_ESCAPE))
{
    parent.nextGameID = EraseGameEngine.INTRO;
    finish();
}
```

Here is what the `GameObject` is saying to its parent: The next thing your going to do is run the Intro screen because I'm FINISHED!

EraseLevelOne.java – all better now

Since we've moved all the "mode" type logic out of the `EraseLevelOne` class, I'll present the new simplified version here in its entirety. I've also removed the `main()` method since we only want to start our game from the `EraseGameEngine` class. Also note that I added a way to get points, pressing the Control key (this game is gonna rock!!!!!!):

EraseLevelOne.java

```
package eraserhead;

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

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

```

public class EraseLevelOne extends GameObject
{
    private Sprite hero;
    private Background backgr;

    public EraseLevelOne(GameEngine parent)
    {
        super(parent);
    }

    @Override
    public void initResources()
    {
        hero = new Sprite(getImage("resources/eraserguy.png"));
        backgr = new ColorBackground(Color.WHITE, 640, 480);
        hero.setBackground(backgr);
    }

    @Override
    public void update(long elapsedTime)
    {
        backgr.update(elapsedTime);
        hero.update(elapsedTime);

        if (keyDown(KeyEvent.VK_ESCAPE))
        {
            parent.nextGameID = EraseGameEngine.INTRO;
            finish();
        }

        if (keyDown(KeyEvent.VK_LEFT))
        {
            hero.setX(hero.getX() - 1);
        }

        if (keyDown(KeyEvent.VK_RIGHT))
        {
            hero.setX(hero.getX() + 1);
        }

        if (keyDown(KeyEvent.VK_UP))
        {
            hero.setY(hero.getY() - 1);
        }

        if (keyDown(KeyEvent.VK_DOWN))
        {
            hero.setY(hero.getY() + 1);
        }

        if (keyDown(KeyEvent.VK_CONTROL))
        {
            ((EraseGameEngine)parent).score++;
        }
    }

    @Override
    public void render(Graphics2D g)
    {
        backgr.render(g);
        hero.render(g);
        g.setColor(Color.BLUE);
        g.drawString("Score: " + ((EraseGameEngine)parent).score, 20, 400);
    }
}

```

It's worth pointing out that the new and improved EraseLevelOne class is devoid of all Intro screen and Help screen code. All it needs to worry about now is playing the game.

We need more objects!

The Intro screen and Help screens have now been moved to their own classes that extend GameObject. Here is each of them.

EraseIntro.java

```
package eraserhead;

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

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

public class EraseIntro extends GameObject
{
    private Sprite hero;
    private Background backgr;

    public EraseIntro(GameEngine parent)
    {
        super(parent);
    }

    @Override
    public void initResources()
    {
        hero = new Sprite(getImage("resources/eraserguy.png"));
        backgr = new ColorBackground(Color.WHITE, 640, 480);
        hero.setBackground(backgr);
    }

    @Override
    public void update(long elapsedTime)
    {
        hero.setLocation(300, 100);

        if (keyPressed(KeyEvent.VK_ESCAPE))
        {
            finish();
        }

        if (keyDown(KeyEvent.VK_SPACE))
        {
            parent.nextGameID = EraseGameEngine.PLAY;
            finish();
        }

        if (keyDown(KeyEvent.VK_F1))
        {
            parent.nextGameID = EraseGameEngine.HELP;
            finish();
        }
    }

    @Override
    public void render(Graphics2D g)
    {
        backgr.render(g);

        g.setColor(Color.BLUE);
        g.drawString("Eraser Head - Copyright (c) Wicked Cool Games", 20, 20);
        g.drawString("Press F1 for Help", 20, 50);
        g.drawString("Press SPACE to Play", 20, 70);
        g.drawString("Last Score: " + ((EraseGameEngine) parent).score, 20, 400);

        hero.render(g);
    }
}
```

EraseHelp.java

```

package eraserhead;

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

import com.golden.gamedev.*;

public class EraseHelp extends GameObject
{
    public EraseHelp(GameEngine parent)
    {
        super(parent);
    }

    @Override
    public void initResources()
    {
    }

    @Override
    public void update(long elapsedTime)
    {
        if (keyPressed(KeyEvent.VK_ESCAPE))
        {
            parent.nextGameID = EraseGameEngine.INTRO;
            finish();
        }
    }

    @Override
    public void render(Graphics2D g)
    {
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, 640, 480);

        g.setColor(Color.BLUE);
        g.drawString("Use your arrow keys to move EraseHead around", 20, 20);
        g.drawString("Press your CTRL key for points!", 20, 50);
        g.drawString("Press ESC to return to main screen", 20, 80);
    }
}

```

Our complete EraseGameEngine

The last piece of the puzzle we need to complete the picture is our complete EraseGameEngine class. The interesting part is the `getGame()` method, which now has three possible types of `GameObject` classes to choose from.

EraseGameEngine.java

```

package eraserhead;

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

public class EraseGameEngine extends GameEngine
{
    public static final int INTRO = 0;
    public static final int HELP = 1;
    public static final int PLAY = 2;

    long score = 0;

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

```

```

@Override
public void initResources()
{
    nextGameID = INTRO;
}

@Override
public GameObject getGame(int gameId)
{
    GameObject nextGame = null;

    switch(gameID)
    {
        case PLAY:
            score = 0;
            nextGame = new EraseLevelOne(this);
            break;

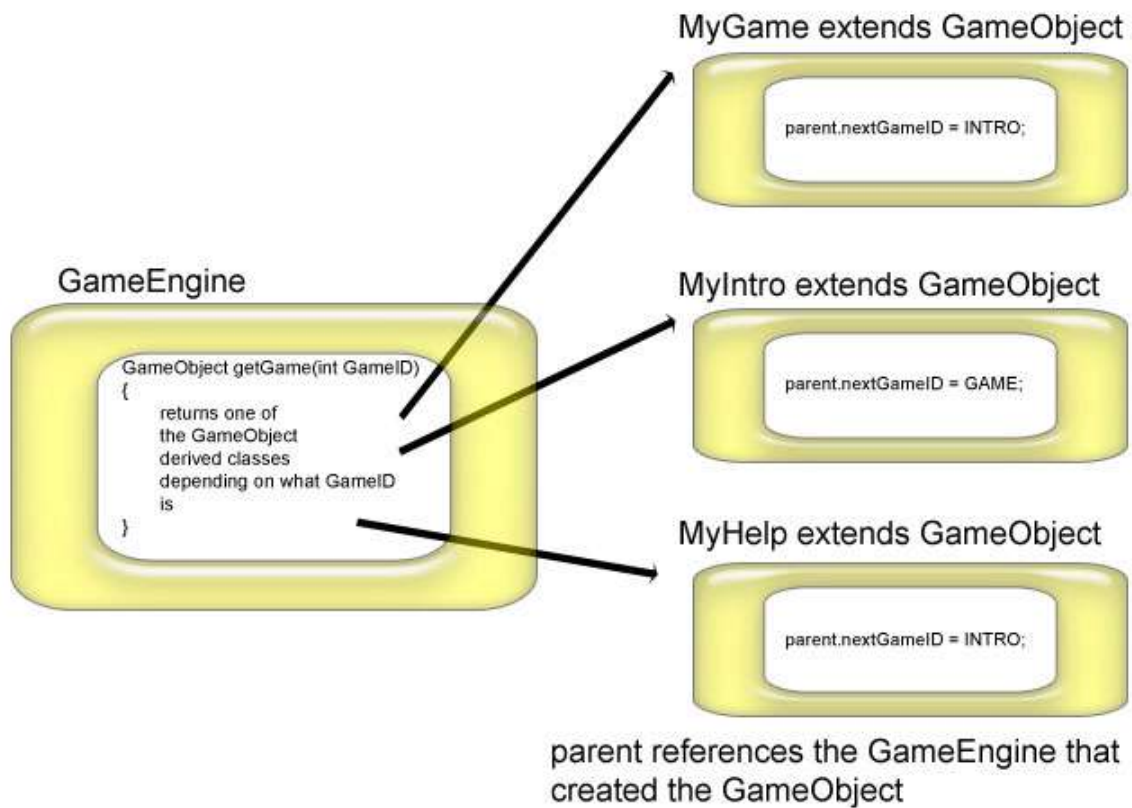
        case INTRO:
            nextGame = new EraseIntro(this);
            break;

        case HELP:
            nextGame = new EraseHelp(this);
            break;
    }

    return nextGame;
}
}

```

A picture worth...



Thank you for playing along

I hope this tutorial has cleared up any questions you might have had about the GameEngine class. It's really a necessary class to learn if you're going to write something with any kind of rooms or screen switching. Even if you have only one game play screen, you're still going to need at least Intro screen and probably a Help screen.

Good luck with your game programming!

Please send any corrections or comments to – john@wagner-usa.net.

The latest version of this document (and my games) can be found at <http://wagner-usa.net/lander.html>

Revision history:

Version 1.0 – 8/5/2005