

Game Programming with Groovy

James Williams
@ecspike

Sr. Software Engineer, BT/Ribbit

About Me

- Sr. Software Engineer at BT/Ribbit
- Co-creator of Griffon, a desktop framework for Swing using Groovy
- Contributor to several open source projects
- Blogger and aspiring writer

About Ribbit

- Based in Mountain View, California, USA
- Subsidiary of British Telecom (BT)
- Provides a programmable telephony API
- Products and services include:
 - Ribbit Mobile
 - Ribbit for Salesforce
 - Ribbit for Oracle CRM

Agenda

- What is Groovy?
- Why Groovy for Gaming?
- What games are good for Groovy?
- Game Loop
- Enhancing Java Applications/Tooling
- User Input
- Alternative Controllers
- Connecting users

What is Groovy?

- Dynamically typed language for the JVM
- Superset of Java
- Inspired by Python, Ruby, and Smalltalk
- Was designed for Java developers in mind
- Integrates seamlessly with Java objects and libraries

Why Groovy for Gaming?

- reduces the amount of code
- can use Domain Specific Languages to provide fluent APIs
- provides robust libraries for reading/writing from XML
- all your Java code just works

Types of games are good for Groovy

- Turn-based games
- Side-scrollers
- Card games
- Simple arcade classics from the old days

Gaming in Java

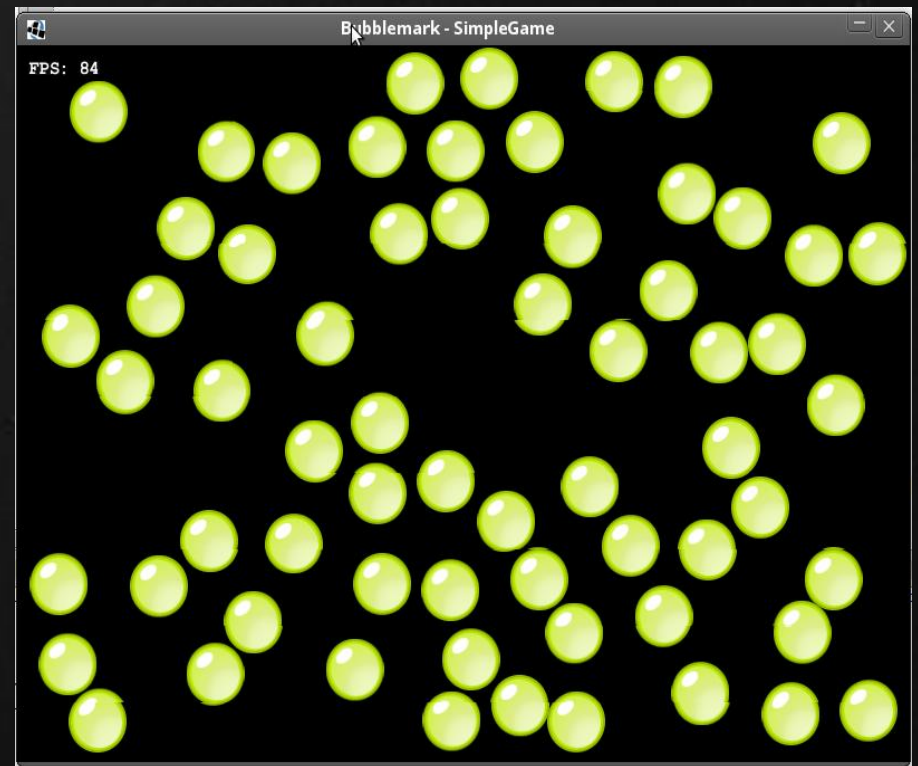
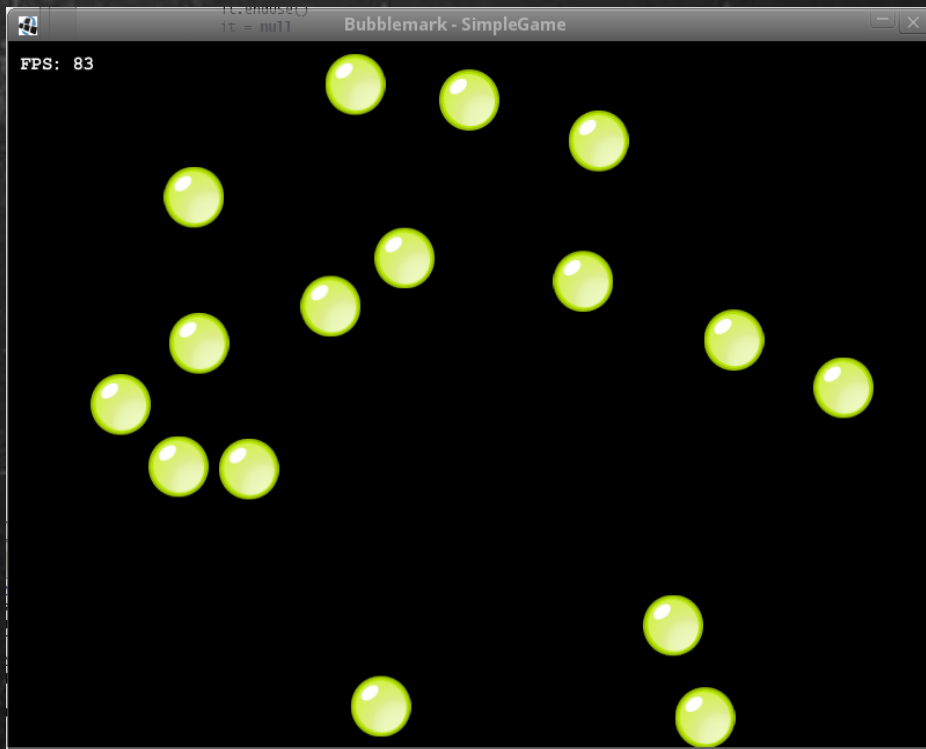
- JavaMonkeyEngine (JME)
 - Supports JOGL and LWJGL
 - Full stack providing sound, graphics, and input
- Lightweight Java Graphics Library
 - Exposes OpenGL, OpenAL, as well as input controller support
- JOGL
- JOAL
- JInput

Slick

- Provides a simple 2D API over LWJGL
- Enables Webstart distribution without the drama
- Extensible framework
- Helps with rendering, sound, and collision detection but doesn't overpower
- Doesn't lock you in

Is it fast enough?

- Yes!
- Bubblemark capped at 80fps runs @ 12% CPU utilization
- Uncapped it devours 90% CPU for 840-1100 fps



BasicGame

- Self-contained instance of a game
- Implements
 - init
 - update
 - render
- Great for single frame games

BasicGame example

```
public class SimpleGame extends BasicGame{
    public SimpleGame() { super("SimpleGame"); }

    @Override
    public void init(GameContainer gc) throws SlickException { }

    @Override
    public void update(GameContainer gc, int delta) throws SlickException{ }

    public void render(GameContainer gc, Graphics g) throws SlickException{ }

    public static void main(args) throws SlickException {
        def app = new AppGameContainer(new SimpleGame())
        app.setDisplayMode 800, 600, false
        app.start()
    }
}
```

GameState

- Useful when there are multiple states like start screen, credits, levels, etc
- Sort of MVC for games
- Allows you to decouple your code
- Supports using visual transitions to move between states

Making things move

- Frame independent movement
 - Calculates based on delta time
 - Same experience on slow and fast computers
 - Built into Slick

@Override

```
public void update(GameContainer gc, int delta) throws  
SlickException { }
```

Bubblemark Demo

Tilemaps

- Think Legend of Zelda for the NES
- series of images arranged in a grid format
- can be of arbitrary size and layer multiple levels
- Slick supports TileD for import
(<http://www.mapeditor.org/>)

Images

- Support via ImageIO for:
 - PNG
 - JPG
 - GIF
- TGA support via a pure Java loader

```
Image img = new Image("res/myimage.png");
```

Sounds and Music

- **SoundFX**

- Supports WAV and OGG
- Non-spatial sounds

```
def fx = new Sound("resources/scream.wav")  
fx.play()
```

//or fx.play(1.0f, 0.5f) to set volume and pitch

- Spatial sounds

```
fx.playAt(-1, 0, 0)
```

- **Music**

- Supports WAV, OGG, and MOD/XM tracks

```
def music = new Music("resources/coolThemeSong.ogg")  
music.loop()
```

Input Handlers

- uses Input class and builds on the existing LWJGL support
- Retrieving input:
 - can be polled just as in LWJGL
 - can register InputListeners to handle notifications
- BasicGame has convenience methods for keyboard, mouse, and controller input

Bluetooth Support on Java (Bluecove)

- Apache 2 Licensed
- Implementation of JSR 82
- Linux version requires GPL library
- <http://code.google.com/p/bluecove/>

Nintendo Wiimote

- Controller for Nintendo's Wii console
- Uses Bluetooth for communication
- IR Camera
- X, Y, Z accelerometer
- Can use IR triangulation to determine distance
- Can use IR to derive roll, pitch, and yaw

Using the Wiimote with Groovy(Java)

- Motej (<http://motej.sourceforge.net/>)
 - pure Java / no native C libs
 - Supports
 - Wiimote
 - Nunchuk
 - Balance Board
 - Classic Controller
 - wiiusej (<http://code.google.com/p/wiiusej/>)
 - wiimote-simple (<http://code.google.com/p/wiimote-simple/>)

Motej setup

- Install Bluetooth native libs
- Install Bluecove JSR 82 libs
 - Linux: requires an extra bluecove-gpl jar
 - OSX SL: needs to compile from src unless using 32-bit JVM
- On the application classpath, include:
 - motej-0.9
 - bluecove jars
 - slf4j-api-1.5.8
 - slf4j-simple-1.5.8

Motej Sample

```
def listener = [moteFound:{Mote mote->
    System.out.println("Found mote: " + mote.getBluetoothAddress())
    mote.setPlayerLeds([false, true, false, true] as boolean[])
    mote.rumble(2000l)
    motes.add(mote)
}]

MoteFinder finder = MoteFinder.getMoteFinder()
finder.addMoteFinderListener(listener)

finder.startDiscovery()
Thread.sleep(30000l)
finder.stopDiscovery()
```

Wiimote Demo

How the Pitcher Demo works

- On button press, the app starts tracking the X motion with slight filtering
- On button release, the app stops tracking X motion
- The change in X over time is used to calculate acceleration
- The X acceleration is used with the distance from the plate to calculate the speed*

Deployment

- Applets
 - Can be done but can be flaky
 - Operates in a sandbox
- Webstart
 - Can have full rights for the system
 - Simple deployment updates
 - Slick/lwjgl jnlp targets to auto-include native files

Scripting (JSR 223)

- Exposes third-party script engines to be embedded in Java apps
- Script Engines available for:
 - Java/BeanShell
 - Groovy
 - Javascript
 - JRuby
 - Jython
 - et al

Scripting API Example

```
def factory = new ScriptEngineManager();  
def engine = factory.getEngineByName("groovy");
```

```
// basic example
```

```
println(engine.eval("(1..10).sum()"));
```

```
// example showing scripting variables
```

```
engine.put("first", "HELLO");
```

```
engine.put("second", "world"); println(engine.eval("first.toLowerCase() +  
second.toUpperCase()"));
```

JSR 223 Demo

Connecting with users (Red Dwarf)

- open source fork of Project Darkstar
- built to support all the features needed for a MMOG
- Provides:
 - persistent data storage
 - transactions
 - channel communications
 - cpu load balancing
- Client drivers exist in Java, C#, and Python

Questions?

Links

LWJGL: <http://lwjgl.org>

JME: <http://www.jmonkeyengine.com>

JOGL: <https://jogl.dev.java.net/>

JOAL: <https://joal.dev.java.net/>

JInput: <https://jinput.dev.java.net/>

Slick: <http://slick.cokeandcode.com>

Red Dwarf: <http://www.reddwarfserver.org/>

Source code for all apps:

<http://github.com/jwill/geecon-demos>