Game Programming with

presented by Nathan Baur

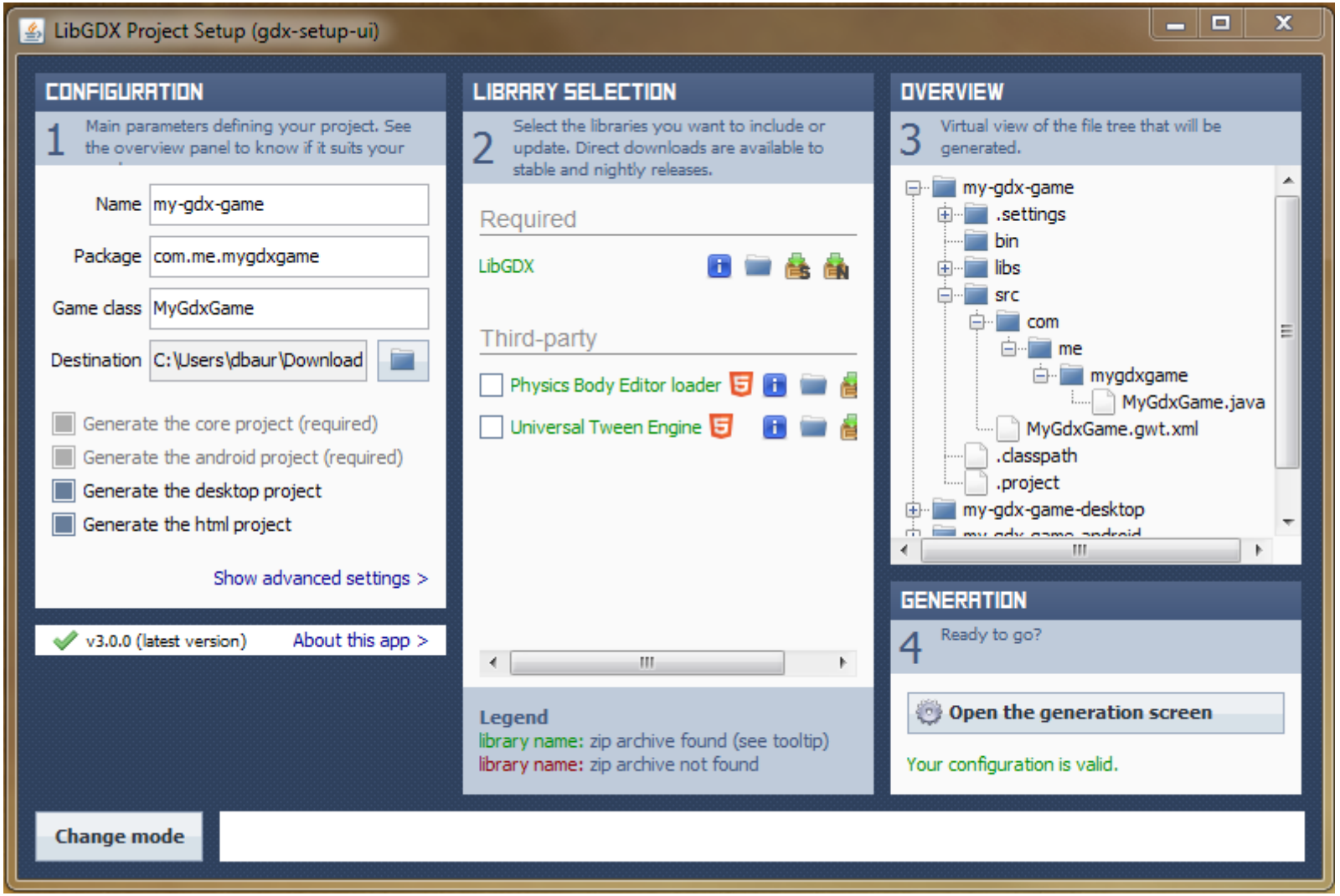
What is libGDX?

- Free, open source cross-platform game library
- Supports Desktop, Android, HTML5, and experimental iOS support available with MonoTouch license ($400)
- OpenGL support means relatively high performance despite high level of abstraction and portability

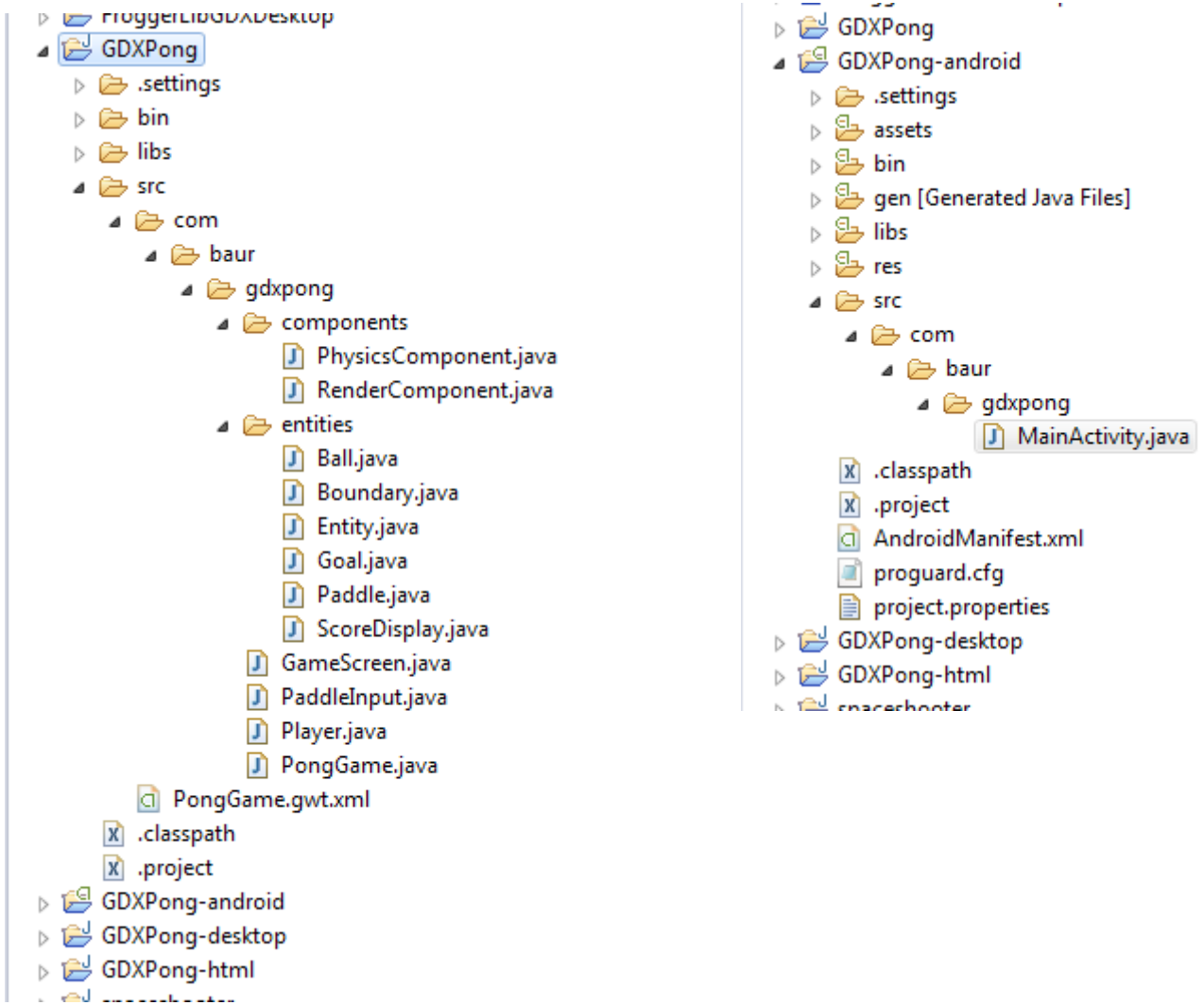
Platform Independence

- Automatic project setup GUI tool will download libraries, update existing projects, create new project layout with working “Hello World”
- One main project for core, platform independent game code
- One project each for platform specific code like Android Manifest XML file
- APIs for handling assets, persistence, graphics, sound, input, etc, minimize code needed in platform specific projects

Platform Independence



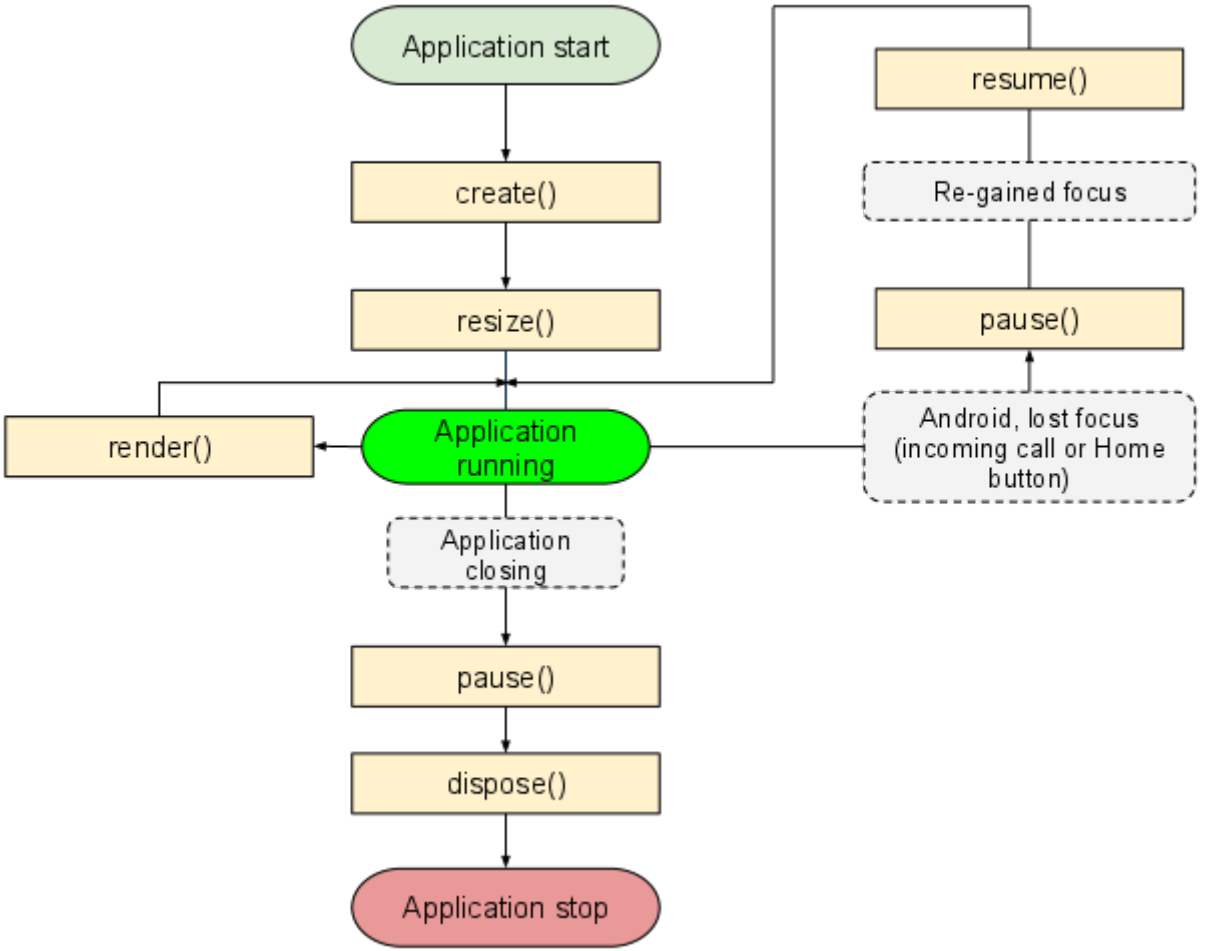
Platform Independence



Life-Cycle

- Main Game class implements ActionListener interface defining life-cycle behavior
- Methods similar to mobile app life-cycle:
  - create()
  - dispose()
  - pause()
  - render()
  - resize(int width, int height)
  - resume()

Life-Cycle



(diagram borrowed from http://code.google.com/p/libgdx/wiki/ApplicationLifeCycle)

Life-Cycle

- Game class delegates to Screen interface which has a very similar life-cycle
- Using multiple Screens (menu, game, highscore, etc) allows for behavior much like Android Activities, although when built for Android everything is actually happening in only one Activity

Game Loop

- Event-driven life-cycle means main game loop is part of the back-end
- This is good because it contributes to platform independence
- render() method holds code for body of main loop
- render() called at 60fps max, elapsed time between frames provided
- This accommodates most approaches to game loop timing

File Handling

- All assets stored in assets directory, which is by default symlinked between projects for convenience
- Files accessed by relative path, eg Gdx.files.internal(“data/image.png”)
- Note “/” used as pathname separator even on Windows
- File module also supports storage in other places, eg Gdx.files.absolute("/some_dir/subdir/myfile.txt")
- FileHandle class provides interface for file system operations like delete and copyTo
- Best to stick to read-only internal storage when possible due to platform-specific limitations

Persistence

- Effortless key-value configuration persistence provided through Preferences class
- Preferences instance constructed by factory
  Gdx.app.getPreferences(“Name of map”)
  so all details of storage are abstracted away
- Each prefs instance can store large number of values:
  prefs.putInteger(“highscore”, 10)
- Types limited to Boolean, Float, Integer, Long, String
- Also includes utilities for JSON and XML based serialization for more complex persistence tasks, but like with assets it is best for portability to stick to Preferences whenever possible

Graphics Overview

- Everything is based on OpenGL ES
- Different back-end for each platform (lwjgl, WebGL, etc)
- Support for 2D and 3D graphics, although I have only used 2D
- Useful facades like Mesh and Sprite for basic graphical tasks
- Also provides wrappers for low-level OpenGL calls when necessary
- Built-in Camera classes for easy projection from game coordinates to screen coordinates

Texture, Sprite, SpriteBatch

- Texture class represents imported image
  - Is exempted from garbage collection, it must be disposed of manually
- TextureRegion class represents a subset of a Texture
  - Useful for sprite sheets, where multiple poses or animation frames are stored in one image
  - Useful for irregularly shaped sprites, since OpenGL 1 requires texture sizes be powers of 2
- Sprite class has a TextureRegion, concept of location, and many useful methods for scaling, rotating, tinting, etc
- SpriteBatch is basically a canvas that TextureRegions and Sprites can be drawn to
  - Uses camera projection
  - Manages alpha blending

Texture, Sprite, SpriteBatch

```
protected void loadSprite(String fileName, Vector2 size){
    texture = new Texture(Gdx.files.internal(fileName));
    texture.setFilter(TextureFilter.Linear, TextureFilter.Nearest);
    TextureRegion region = new TextureRegion(texture);

    sprite = new Sprite(region);
    sprite.setOrigin(sprite.getWidth()/2.0f, sprite.getHeight()/2.0f);

    sprite.setSize(size.x, size.y);
}
```

```
@Override
public void render(float delta) {

    float deltaT = Math.min(delta,1.0f/30.0f);
    update(deltaT);

    Gdx.gl.glViewport((int)viewport.x, (int)viewport.y, (int)viewport.width, (int)viewport.height);
    Gdx.gl.glClearColor(0, 0, 0, 0);
    Gdx.gl.glClear(GL10.GL_COLOR_BUFFER_BIT);

    batch.setProjectionMatrix(camera.combined);
    batch.begin();

    for(Entity entity : entities){
        entity.render(batch);
    }

    batch.end();

}
```

Resolution Independence

- Game coordinates are transformed into screen coordinates through use of viewports and cameras
- Game coordinates are continuous by default, but can be made discrete for games with a “pixel-perfect” art design

Resolution Independence

```
@Override
public void resize(int width, int height) {

    float w = (float)width;
    float h = (float)height;

    float aspect = gameWidth/gameHeight;
    float scale = 1f;
    Vector2 crop = new Vector2(0f,0f);
    if(w/h>aspect){
        scale = h/gameHeight;
        crop.x = (w-gameWidth*scale)/2f;
    }else if(w/h<aspect){
        scale = w/gameWidth;
        crop.y = (h-gameHeight*scale)/2f;
    }else{
        scale = w/gameWidth;
    }

    viewport = new Rectangle(crop.x, crop.y, gameWidth*scale, gameHeight*scale);
}
```

scene2d and User Interfaces

- scene2d is a scene graph, which provides a different approach to drawing 2D graphics that is more convenient for creating interfaces
  - Stage and Actor concepts have children in local (relative) coordinate systems that move and rotate with their parents
  - Automatic hit detection and event-driven actions
  - API is so simple and sensible that some people choose to build their entire game in scene2d
- scene2d.ui builds convenience classes on top of scene2d for interface design
  - Provides Layouts, Tables, and a host of Widgets like Button and Slider
- Skin class packages UI assets like images and fonts for easy switching

Sound

- Sound object provides extremely simple interface for playing sound effects:

```
sound = Gdx.audio.newSound(Gdx.files.internal("data/bip.mp3"));
float pitch = MathUtils.clamp((physicsComponent.getBody().getLinearVelocity().len()-25) / 150f,0.25f,0.5f);
sound.play(1f,pitch,0f);
```

- Also supports background music and low-level PCM playback

Input

- Supports input from many sensors from keyboard and mouse to compass and accelerometer
- Event-driven input supported through InputProcessor interface
- Input polling is also available as a simpler but less reliable alternative
- Multi-touch and gesture recognition support for touch screens

Input

```
9
10 public class PaddleInput implements InputProcessor{
11
12     protected Array<Paddle> paddles;
13     protected Camera camera;
14     protected GameScreen screen;
15
16     public PaddleInput(GameScreen screen, Camera camera){
17         this.camera = camera;
18         this.screen = screen;
19         paddles = new Array<Paddle>(false,2);
20     }
21
22     public void addPaddle(Paddle paddle){
23         paddles.add(paddle);
24     }
25
26     @Override
27     public boolean touchDown(int x, int y, int pointer, int button) {
28         return touchedOrDragged(x, y);
29     }
30
31     @Override
32     public boolean touchDragged(int x, int y, int pointer) {
33         return touchedOrDragged(x, y);
34     }
35
36     private boolean touchedOrDragged(int screenX, int screenY){
37         boolean handled = false;
38         Vector3 position = new Vector3(screenX, screenY, 0);
39         camera.unproject(position, screen.viewport.x, screen.viewport.y, screen.viewport.width, screen.viewport.height);
40         float x = position.x;
41         float y = position.y;
42
43         for(Paddle paddle : paddles){
44             if(paddle.getInputRegion().contains(x, y)){
45                 paddle.setY(MathUtils.clamp(y, -GameScreen.gameHeight/2+paddle.getBounds().height/2+1, GameScreen.gameHeight/2-paddle.getBounds().height/2-1));
46                 handled = true;
47             }
48         }
49         return handled;
50     }
51
52
```

Physics

- Includes wrappers for popular C++ physics engines Box2D and Bullet3D
- Each physics engine could be a whole presentation on its own

Box2D Overview

- World
  - Manages all bodies and global properties like gravity
  - Handles passage of time, movement integration, collisions, etc
- Body
  - Represents single physical object
  - Made up of Fixtures
  - Can be Dynamic, Static, or Kinematic
    - In the Pong game example the ball is Dynamic, the paddles are Kinematic, and the boundaries are Static

Box2D Overview

- Fixture
  - Exists in local coordinate system of parent Body
  - Holds actual physical properties like shape, density, friction, restitution
- Collision handling
  - Collisions are called Contacts and occur between Fixtures
  - Can be event-driven with ContactListener interface or polled with World.getContactList()
  - Contact object stores pair of Fixtures and other useful information like the angle of the collision
  - Fixtures can store references to their parent Sprite or game entity using setUserData and getUserData, which is important if the collision is to have some effect on the entity

Box2D Overview

```
world = new World(new Vector2(0f,0f), false);
world.setContactListener(new ContactListener(){
    @Override
    public void beginContact(Contact contact) {
        Object entityA = contact.getFixtureA().getUserData();
        Object entityB = contact.getFixtureB().getUserData();
        if(contact.isTouching() && entityA != null && entityB!=null){
            ((Entity)entityA).handleCollision(contact);
            ((Entity)entityB).handleCollision(contact);
        }
    }
    @Override
    public void endContact(Contact contact) {}
    @Override
    public void preSolve(Contact contact, Manifold oldManifold) {}
    @Override
    public void postSolve(Contact contact, ContactImpulse impulse) {}
});

public void update(float deltaT){
    physicsTime += deltaT;

    while(physicsTime>=physicsTimeStep){
        physicsTime -= physicsTimeStep;
        world.step(physicsTimeStep, 8, 3);
    }

    camera.update();

    for(Entity entity : entities){
        entity.update(deltaT);
    }
}
```

Box2D Overview

```
public PhysicsComponent(Entity owner, World world, Rectangle bounds){
    this.owner = owner;
    this.world = world;

    BodyDef bodyDef = new BodyDef();
    bodyDef.position.set(bounds.getX()+bounds.getWidth()/2, bounds.getY()+bounds.getHeight()/2);
    bodyDef.linearDamping = 0.0f;

    body = world.createBody(bodyDef);

    FixtureDef fixDef = new FixtureDef();
    PolygonShape shape = new PolygonShape();
    shape.setAsBox(bounds.getWidth()/2, bounds.getHeight()/2);
    fixDef.shape = shape;

    fixture = body.createFixture(fixDef);

    shape.dispose();

    fixture.setUserData(owner);
}

public Ball(GameScreen screen, Rectangle bounds){
    this.screen = screen;
    renderComponent = new RenderComponent("data/2by2.png",new Vector2(bounds.getWidth(),bounds.getHeight()));

    physicsComponent = new PhysicsComponent(this, screen.getWorld(), bounds);
    physicsComponent.getBody().setBullet(true);
    physicsComponent.getBody().setType(BodyType.DynamicBody);
    physicsComponent.getFixture().setRestitution(1.1f);
    physicsComponent.getFixture().setFriction(0.0f);
    physicsComponent.getFixture().setDensity(1.0f);

    sound = Gdx.audio.newSound(Gdx.files.internal("data/bip.mp3"));

    mustReset = false;
    reset();

}
```

Box2D Overview

```
@Override
public void handleCollision(Contact contact) {

    String typeA = ((Entity)contact.getFixtureA().getUserData()).getType();
    String typeB = ((Entity)contact.getFixtureB().getUserData()).getType();
    if(typeA.equals("Goal") || typeB.equals("Goal")){
        mustReset = true;
    }
    if(typeA.equals("Paddle") || typeB.equals("Paddle")){
        float pitch = MathUtils.clamp((physicsComponent.getBody().getLinearVelocity().len()-25) / 150f,0.25f,0.5f);
        sound.play(1f,pitch,0f);
    }
}
```

Other Utilities

- MathUtils package
  - Performance-oriented float versions of useful math functions to avoid double<->float conversion
  - Classes for Tween interpolation, Splines, Vectors, basic Geometry
- Basic collection classes like Pool and Array with garbage minimization in mind
- Particle engine with GUI editor
- BitmapFont engine with GUI editor
- Importer for files exported by popular Tiled map editor

Example Game

- Multitouch Pong game intended for touchscreen
- Code used in examples throughout presentation
- https://github.com/nathanbaur/GDXPong