Introduction to LibLOL



Game Design Should Be Fun!

- Mobile development is often cumbersome
 - Can't test multitouch, vibration, and tilt without a physical device
 - But it takes a long time to deploy a game onto a phone (~30 seconds)
 - Hard to keep a good workflow
- Wouldn't it be nice if
 - You could simulate tilt on your computer
 - Run your game on your computer
 - Deploy it onto a phone (Android or iOS) for final 'polishing' tests
- Until last month, LibLOL was ALE
 - Used AndEngine as backend instead of LibGDX
 - Suffered from above problems



Rapid (and Fun) Game Design

- Develop your LibLOL game in Eclipse using Java
 - Set up 3 projects: game, game-desktop, and game-android
 - All media files go into Android project's assets folder
 - All core game code goes into root folder
 - Any OS-specific code (advertisements, in-app purchases) goes in desktop or Android folder, as appropriate
 - (note: can add a 4th folder for iOS)
- Test your game by running on the desktop
 - Game launches in 2-3 seconds, with no "loading" or lag
 - Simulate tilt via arrow keys, if necessary
 - Log messages show up in an Eclipse view





Games are Simulations

Key concept: the game loop

```
while (true) {
    poll_for_input()
    run_AI()
    advance_world()
    render()
    audio()
}
```

- This is different from how we usually write code
 - Something is going to happen even in the absence of user interaction



Layers of Simulation

- It's tempting to have two kinds of simulations running simultaneously
 - Use a physics engine (e.g., box2d) for the tricky stuff
 - Use ad-hoc techniques for the rest

Pros:

- Good for rapidly getting stuff running
- Can often do each task in the easiest way possible



Cons:

- Hard to maintain
- Rapidly becomes inefficient (two collision detections)
- Easy to achieve non-physical behavior



LibLOL Simulation Characteristics

- Only one simulation engine
 - Everything happens via box2d
 - This can be a pain to code up at first...
 - ...but we did most of the hard work for you
 - And the box2d tutorials online are quite good
- Note: be very careful about static vs. dynamic bodies
 - Both have physical properties
 - But static bodies shouldn't move, don't experience forces, and can't cause a collision to be handled
 - There are also kinematic bodies
- Use sensors to detect collisions without affecting momentum



- LibLOL exposes a few general classes of objects that exist in the simulation
 - Hero: the thing the player usually controls; it must achieve a goal, and must not be defeated by enemies
 - Enemy: a thing that causes harm to the hero
 - Goodie: a thing the hero needs to collect
 - Obstacle: walls, and so much more
 - Destination: a place the hero must reach to win
 - Projectile: something the hero can throw
- Other key concepts
 - There is a timer mechanism
 - Can draw arbitrary pictures
 - Can draw complex shapes as simple obstacles via SVG



Implementing "AI"

- The render() step calls each entity's render method
 - In theory, you could do per-entity AI within this code
- Preferred technique: use events to drive entity behaviors
 - Everything is obeying the laws of physics
 - Can attach a callback to almost any collision between entities
 - Can attach a callback to touches of almost any entity
 - Can attach a callback to a timer
 - Set a new timer within the callback to get periodic timers



- The most important input for most mobile games
- Every entity can have a callback when you touch it
- LibLOL also support a Heads-Up Display (HUD)
 - Allows buttons to always appear at same place, regardless of where entities are
 - Use for displaying information or receiving touch events





Bald Felines

- "There's more than one way to skin a cat"
- The best code is the code that achieves the desired behavior
 - Consider invisible buttons... they are easier than setting a touch handler on the screen
 - Likewise, consider invisible enemies or off-screen enemies... a great way to simulate a pit
 - My favorite: overlay invisible enemy on top of a visible picture/obstacle



11





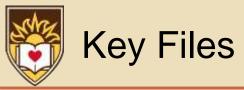
Look Ma, No Render Loop

- Every level or UI screen has its own render loop
 - It's encapsulated, you probably won't see or edit it
- For maximal simplicity, LibLOL expects everything to be in a few spaghetti functions
 - Make separate classes and forward to them if you know what you're doing

Main functions

- Describe how to draw the splash screen
- Load all graphics and sounds
- Draw initial state of each level
- Update game state in response to an event
- Draw help screens
- Configure game and level-chooser

2/7/2014 CSE202 13

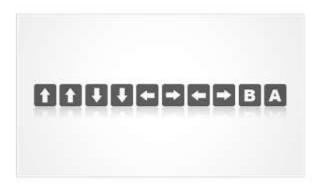


- ChooserConfig.java
 - Creates an object that describes how to draw the level chooser
- LolConfig.java
 - Describes game-wide constants (width, height, # levels, etc)
 - Use this to enable/disable debug mode
- MyLolGame.java
 - Everything else(you can forward to your own classes if you wish)



Debug Mode

- Shows the physics that accompanies each picture
- Shows the outline of every Control on the HUD
- Leads to more output in the Eclipse debug window
- Unlocks all levels





Getting Started

- Check out the code
- Use the supplied script to rename your namespaces
- Quick configuration stuff
 - Set your Android icon and screen orientation
 - Update the two configuration files
- Register media
- Replace level 1 with your code
 - Refer to the other 80+ levels for simple demos of how to use LibLOL functionality



Enough Talk, Let's Fight Code!

· Shashakablooie!



http://github.com/mfs409/liblol