Computer Game Fundamentals
1CB104
03 – Game Dev Environments

Dr Andy Guest

a.guest@yorksj.ac.uk

Room 104 44 Lord Mayor’s Walk

01904 87**6524**

Different Approaches

- Modern games are complex and made of many different parts
- To develop a game from scratch, using the only the core components of a programming language is possible but requires a great deal of effort
- Fortunately we don't have to develop games this way, there are many tools out there to help us

Levels of tools

- The next few slides break these tools down into different levels of tools.
- Keep in mind these levels are completely arbitrary and for illustrative purposes only

Lowest Level – Programming Languages

- The lowest level covered here is that of the programming language
    - Note it is possible to go lower to assembly language and even machine code but nobody in their right minds develops games at that level!
- Developer only has the features of the language to work with
    - Depending on the language, may have to write code to create windows from scratch
    - Certainly will have to create any game features from scratch, probably has to implement most features for graphics, sound, etc.
- Doesn't happen in industry, too much re-invention of code that already exists
- Only really done be people who are learning and building up from basics.

Next Level Up – Common Libraries

- The next level consists of using common libraries
  - Window creation/management, graphics (OpenGL, DirectX), sound, etc..
  - Sit on top of programming language
  - Not designed specifically for games
  - Handle the complexity, use tried and tested code
  - Still have to write specialised game code – collision detection, etc..
- Not used in industry, though some demo games. Student games written this way sometimes slip through

Third Level – Game Libraries

- Libraries written specifically for games
- Box2D, libGDX, etc.
    - Handle game specific features – collision detection, etc..
    - Handle graphics, sound, etc.. in ways that support games specifically
    - Sits on top of common libraries
- Used in industry
    - Libraries may be licensed, open source, made in house

Fourth Level – Game Engines 1

- (I'm breaking game engines up in to three levels, again these are arbitrary distinctions for this lecture rather than real levels)
- Lowest level of game engine is simply the core of a game
  - Generally an engine focused on a specific type of game
    - FPS, point and click adventure, etc..
  - Handles the features of that game
  - Leaves specifics of implementation up to the developers to code
  - Unreal, Quake
  - Level developing, graphics, sounds
- Used in industry, not as simple, plug and play as I'm making it sound though

Fifth Level – Game Engines 2

- Fully featured game development engines – Unity, etc..
- Can be used to implement any type of game
  - though some engines are better suited to some types of games than others
- Still involves a great deal of coding
- Most commercial games are built this way
  - Or at least partially this way

Sixth Level – Game Engines 1

- Game Making Software
- Drag'n'drop game creation
- Involves very little coding
- Aimed at kids (usually!)
- Never used to make commercial games
    - At least not by anyone reputable

Game Development

- A wide range of tools
- Supporting developers in different ways and to different levels
- Choosing the right tool to develop a game with is important
- Choice is affected by
    - Price – some tools are completely free to use, some are very expensive
    - Familiarity – the experience of the developer(s) with specific tools may mean the best tool might not be the one that is best for the game but the one that the developers know best
    - Company choice – the choice will often come down to “this company uses X and Y, never A or B”

Game Development

- It is good to get to know a few of these tools well
- If you only know one you may find it hard if you have to work with another
- If you know a handful you should understand the features better and find it easier to learn a new tool
- At Uni is the perfect time to experiment and play around with different tools to get a feel for them

libGDX

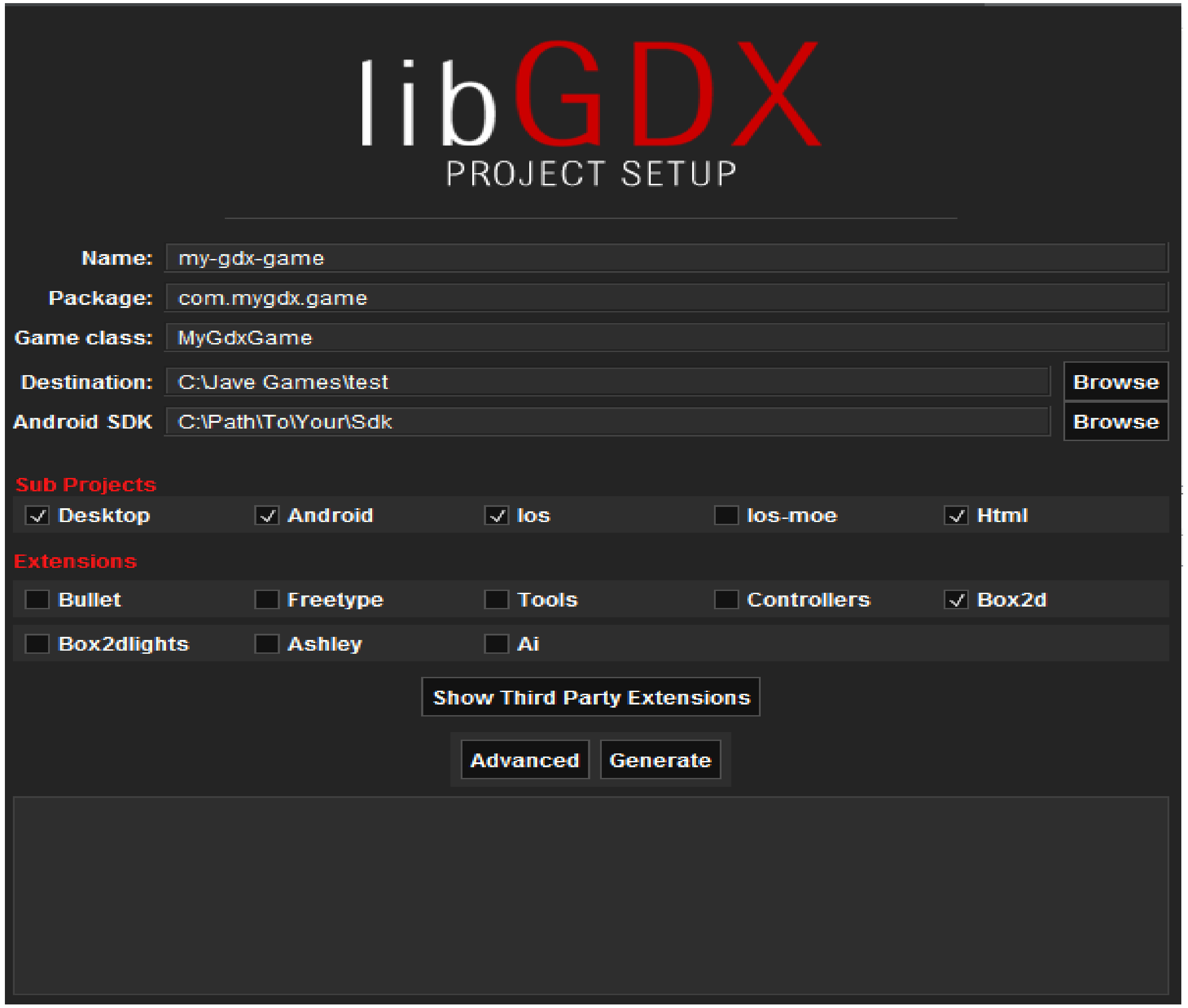
- We are going to be using libGDX in this module
- libGDX is a cross-platform library for Java
  - Desktop (Windows/Linux), Android, BlackBerry, iOS, HTML5
- libGDX sits on top of Java and handles many of the basic features common in games
- We will be writing some code using it to put together a basic 2D game
  - Learning some game fundamentals along the way

Installing libGDX

- Download libGDX from https://libgdx.badlogicgames.com/download.html
  - Click the Download Setup App
- Create a folder in your home directory called libGDX
- Copy gdx-setup.jar from the download folder to your new libGDX folder
- Change the permission of gdx-setup.jar so it is executable

Setting Up A libGDX Project - 1

- Run gdx-setup.jar
- Should see this window



Name: my-gdx-game
Package: com.mygdx.game
Game class: MyGdxGame
Destination: C:\Jave Games\test
Android SDK C:\Path\To\Your\Sdk

Sub Projects
✓ Desktop ✓ Android ✓ Ios ☐ Ios-moe ✓ Html

Extensions
☐ Bullet ☐ Freetype ☐ Tools ☐ Controllers ✓ Box2d
☐ Box2dlights ☐ Ashley ☐ Ai

Setting Up A libGDX Project - 2

- Change the **Name:** to my-first-game
- Change **Game class:** to my-first-game
- Change **Destination:** to /home/your.name/libGDX/my-first-game
- Uncheck all the tick boxes except Desktop
  - Desktop should be the only one ticked
- Click Advanced
  - Tick Eclipse, Press Save
- Click Generate
- Your should see a bunch of text appear at the bottom of the window, hopefully telling you it was successful.
- If successful, close the window
- If unsuccessful, yell for me

Setting Up A libGDX Project - 3

- Start up Eclipse
- File > Import > General > Existing Projects into Workspace
- Next
- Select root directory, Browse to libGDX/my-first-game
- Ensure that Copy projects in to workspace is ticked
- Click Finish

Setting Up A libGDX Project - 4

- You should now see two projects in the package explorer window of Eclipse
- Open the one with core in its name
- Right click on assets, select Build Path and Use as Source Folder
- Open up the package with desktop in its name
- Open src and keep opening up that chain till you get to DesktopLauncher
- Right click on DesktopLauncher and select Run As > Java Application
- With any luck the code should compile and you should see an ugly window appear on the screen!

libGDX Project Parts

- The package labelled …-desktop is used to launch the “game” on the PC
- The package labelled …-core is where we are going to write our code.
  - For the time being at least!
- The assets folder is where we will store any graphics or sound files we use in the game

…-core Package

Open up the package labelled …-core and find my-first-game.java, open that file up

```
package com.mygdx.game;

import com.badlogic.gdx.ApplicationAdapter;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;

public class firstGame extends ApplicationAdapter {
SpriteBatch batch;
Texture img;

@Override
public void create () {
batch = new SpriteBatch();
img = new Texture("badlogic.jpg");
}

@Override
public void render () {
Gdx.gl.glClearColor(1, 0, 0, 1);
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
batch.begin();
batch.draw(img, 0, 0);
batch.end();
}

@Override
public void dispose () {
batch.dispose();
img.dispose();
}
}
```

…-core Package

```
package com.mygdx.game;

import com.badlogic.gdx.ApplicationAdapter;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;


public class firstGame extends ApplicationAdapter {



SpriteBatch batch;
Texture img;


@Override
public void create () {
batch = new SpriteBatch();
img = new Texture("badlogic.jpg");
}
```

- import section ensures the parts of libGDX that we are going to use are loaded
- Creates the class we will use for our game
- Creates class variables that are important for our game
- The create() function initialises our game. It sets up all the things than need to be set up

…-core Package

```
@Override
public void render () {
Gdx.gl.glClearColor(1, 0, 0, 1);
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
batch.begin();
batch.draw(img, 0, 0);
batch.end();
}

@Override
public void dispose () {
batch.dispose();
img.dispose();
}
}
```

- The render() function handles all the “drawing” our game requires

- The dispose() function tidies up when we exit the game

What does it all mean?

```
package com.mygdx.game;
```

- Gives our game a package name
  - So we can use it from other packages

```
import com.badlogic.gdx.ApplicationAdapter;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
```

- Loads the basic libGDX functionality required for the game

```
import com.badlogic.gdx.graphics.Texture;
```

- Loads the texture class
  - Used to load and store image files so that they can be used by the game

```
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
```

- Loads the SpriteBatch class
  - SpriteBatch handles the rendering of 2D graphics

What does it all mean?

```
public class firstGame extends ApplicationAdapter {
```

- Creates an object of the SpriteBatch class called *batch*

```
SpriteBatch batch;
```

- We can use *batch* to access the functionality of the SpriteBatch class

```
Texture img;
```

- Creates an object of the Texture class called *img*
  - We can use img to access the functionality of the Texture class
  - Specifically we will use it to store a jpg image so that we can display it

What does it all mean?

```java
@Override
public void create () {
```

- The start of the create() function
  - Handles all the initialisation required for the game

```java
batch = new SpriteBatch();
```

- Turns the **batch** variable in to a handle for accessing the functionality of SpriteBatch

```java
img = new Texture("badlogic.jpg");
}
```

- Loads the badlogic.jpg from the assets folder in to the object variable **img**

What does it all mean?

```
@Override
public void render () {


Gdx.gl.glClearColor(1, 0, 0, 1);
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);


batch.begin();


batch.draw(img, 0, 0);


batch.end();
}
```

- The render() function handles all of the rendering of graphics
- Sets the current colour to red
- Clears the window to the current colour - red
- All the code to draw must be called between batch.begin(); and batch.end();
- Calls on batch to draw the graphics stored in img
- Tells batch that all the drawing code is complete

Let's Make Some Changes!

Add the red lines to your code

```
package com.mygdx.game;

import com.badlogic.gdx.ApplicationAdapter;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.g2d.BitmapFont;

public class firstGame extends ApplicationAdapter {
SpriteBatch batch;
Texture img;
private BitmapFont font;

@Override
public void create () {
batch = new SpriteBatch();
img = new Texture("badlogic.jpg");
font = new BitmapFont();
font.setColor(Color.WHITE);
}

@Override
public void render () {
Gdx.gl.glClearColor(1, 0, 0, 1);
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
batch.begin();
batch.draw(img, 0, 0);
font.draw(batch, "Hello World", 200, 200);
batch.end();
}

@Override
public void dispose () {
batch.dispose();
img.dispose();
font.dispose();
}
}
```

Putting Text On Screen

To put text on screen we need to do several things

```
import com.badlogic.gdx.graphics.Color;

import com.badlogic.gdx.graphics.g2d.BitmapFont;

private BitmapFont font;

font = new BitmapFont();

font.setColor(Color.WHITE);


font.draw(batch, "Hello World", 200, 200);

font.dispose();
```

- We load the colour palette
- We need to load the part of the library that handles fonts
- We have to set up a variable to be the handle for dealing with fonts
- We have to initialise that variable

- We set the font colour to white so it shows up against the red background
- We tell the game to draw, as part of the batch draw, the string “Hello World” at position (200,200)
- We tidy up after ourselves

Try Fiddling With The Code

- What happens when you change the coordinates of the text or img?
  - Not every jpg will work, why?
- Try downloading an image, adding it to the assets folder and try and show it on screen?
  - Not every jpg will work, why?

If You Make It This Far

- The libGDX wiki details “A Simple Game” at https://github.com/libgdx/libgdx/wiki/A-simple-game
- We will work through this next week and I’ll explain how it works
- Start a new project and try to get it working
- Skip the Android sections