Lecture 1: DB Schema, Model Classes, Data Access Classes, Data Importer

Architecture Overview

See asteroids-architecture-diagram.pdf

Initial Class Identification

(Put in "model" package)

Avoid code duplication (use inheritance)

Space

BackgroundObject

Ship

Asteroid ← RegularAsteroid, OcteroidAsteroid, GrowingAsteroid

Projectile

MiniMap

All of these are visible, some have positions, some move

VisibleObject ← PositionedObject ← MovingObject

VisibleObject ← Space

PositionedObject ← BgObject, MiniMap

MovingObject ← Projectile, Asteroid, Ship

Update & Draw methods on VisibleObject

Container object for model objects (e.g., AsteroidsGame)

JSON

(See Asteroids Data document)

Background Object "types" [extendible list]

BgObject(*) → BgObjectType

Asteroid "types" [extendible list]

Asteroid(*) → AsteroidType

Level → BgObject → BgObjectType

Level → LevelAsteroid → AsteroidType

Ship Part “types”

ShipPart ← MainBody, Engine, Cannon, ExtraPart, PowerCore

Some ship parts are “attached” to the main body

AttachableShipPart ← Engine, Cannon, ExtraPart

Ship(1) → MainBody, Engine, Cannon, ExtraPart, PowerCore

Some of these objects have images associated with them (image file, imageWidth, imageHeight)

ImageObject ← BackgroundObjectType, AsteroidType, ShipPart

DB Schema

Game definition objects are in the database, many of the Visible objects are not.

When in memory, DB objects need to store their DB IDs.

Consider creating a ModelObject superclass that stores a DB ID.

Translate JSON objects into DB tables

Data Access Classes

Encapsulate all DB access code

(Put in database package)

CatalogDemo Example in XML-JSON lecture notes

DbOpenHelper

DAO

Add* methods

GetAll* methods

ClearAll method

Database

Contains DbOpenHelper and DAO(s)

Creates DbOpenHelper, Gets DB, Sends DB to DAO(s)

Has ClearAll method that calls ClearAll method on each DAO

Implements ImportData method (i.e., Database is the data importer)

Adding your code to the SuperAsteroids project

Open SuperAsteroids project in AS

Add "model" package

Add "database" package

Create classes in “model” and “database” packages (or any other package)

Design Document

Text file containing SQL "create table" statements

Javadoc

Put Javadoc comment on each class

Put Javadoc comments on fields and methods (except gettings/setters)

Generating Javadoc in Android Studio

Select packages to you want Javadocs for

Select Tools -> Generate JavaDoc...

Select output directory

Select "private" on the private/package/protected/public scale

De-select "Include test sources"

Click OK. Javadocs will be in the specified output directory.

Public Schema file and Javadocs on the web somewhere (e.g., in your Linux home directory)

Web publishing tutorial

https://docs.cs.byu.edu/wiki/Website_Setup_and_Configuration

Running Your Model/Database Code

ImportActivity has controller it calls on import

Controller must implement the IGameDataImporter interface

Write a controller class that implements IGameDataImporter

Install your controller in ImportActivity.onCreate()

Now your code will be called when user imports

We will learn about Unit Testing next time, which is another way to run and test your code

Accessing SQLite file on your device

Put the <android-sdk-dir>/platform-tools directory on your development computer's PATH so you can easily call adb and sqlite3

See last few slides in Databases lecture slides

Lecture 2: Ship Builder

Controllers Overview

IGameDataImporter – Runs your data importer

Installed in ImportActivity.onCreate(…)

IShipBuildingController – Processes Ship Builder inputs

Installed in ShipBuildingActivity.onCreate(…)

IGameDelegate – Processes Game Play inputs (see Simple Game Architecture document)

Installed in GameActivity.onCreate(…)

IMainMenuController – Processes “Quick Play” input

Installed in MainActivity.onCreate(…)

Asteroids Singleton

Use Asteroids singleton class to make program state accessible across different activities

The Asteroids singleton could do the following:

1. Store model container
2. Store database object (which could also be data importer)
3. Method to load model from database
    a. Called when program first starts up
    b. Called after data import completes (called by importer at end of import)

Singleton created/initialized from MainActivity.onCreate(…)

IGameDataImporter

Installed in ImportActivity.onCreate(…)

Implement IGameDataImporter.importData(…) method so that it invokes your data importer class

Parse JSON data and use DAO class(es) to clear existing data and load new data into the DB

After import, tells AsteroidsGame to reload model from database

Catalog Demo (shows how to import CD catalog data from JSON into database)

IShipBuildingController (extends IGameDelegate)

Installed in ShipBuildingActivity.onCreate(…)

Pass “this” reference into Controller constructor (as IShipBuildingView)

Current ship configuration stored in model (which can be accessed in the Asteroids singleton)

Each part view has its own “Start Game” button (five buttons, not one). Should be enabled only when ship has been completely defined.

Also, each part view has its own arrows (twenty arrows, not four). The arrows in each part view can be configured independently.

Draw diagram with the ShipBuildingActivity on the left, and the ShipBuildingController on the right. Show that both of these classes implement the pre-defined interfaces, and that they have pointers to each other. Show and discuss the methods in the controller interface (view passes user input to the controller). Also show and discuss the methods in the view interface, and how the controller calls these methods to change what is displayed on the screen. Demonstrate the back-and-forth interaction between view and controller while discussing the methods on each.

loadContent()

Loads ship builder images, sounds into memory

unloadContent()

Unloads ship builder images, sounds

onViewLoaded(PartSelectionView partView)

Call View.setArrows(…) to configure arrows

Call View.setStartGameButton(…) to enable/disable “Start Game” button

onSlideView(ViewDirection direction)

Call View.animateToView(…) to move to next part

onPartSelected(int index)

Installs selected ship part in the model

Call View.setStartGameButton(…) to enable/disable “Start Game” button

onStartGamePressed()

Calls View.StartGame() to start the game

onResume()

Does nothing (unless you need it for some reason)

update()

Does nothing (because the ship doesn’t move in the ship builder)

draw() draws the ship

ALL drawing should be done from the draw() method (i.e., if you try to draw using DrawingHelper outside the draw() method, it won’t work properly).

Use DrawingHelper.drawImage(imageId, x, y, rotationDegrees, scaleX, scaleY, alpha) to draw the ship parts

Define SCALE as needed to make the ship look good on your device.

Origin of view is in top-left (y axis inverted). Image center drawn at (x, y). Zero degrees is up. 0 alpha is completely transparent, 255 alpha is completely opaque.

Drawing main body

```
Point gameViewCenter = new Point(DrawingHelper.getGameViewWidth() / 2,
                                 DrawingHelper.getGameViewHeight() / 2);
```

Drawing attached ship parts

See DrawingTheShip.pptx

Use State Pattern to implement stateful behavior of onViewLoaded(), onSlideView(), onPartSelected()

Implementing “Quick Play”: IMainMenuController

Installed in MainActivity.onCreate(…)

onQuickPlayPressed()

Implements the “Quick Play” feature. Automatically builds a complete ship in the model, and starts the game by calling startGame() on its view.

Lecture 3: Game Play

See “Simple Game Architecture” document.

Implementing Game Play: IGameDelegate

Draw picture showing GameActivity and GameController. Show that GameController implements the IGameDelegate interface. (GameActivity implements no interface, because all drawing is done through DrawingHelper.) Show how GameController points to the Model, which is where all the game objects come from.

Installed in GameActivity.onCreate(…)

loadContent() is called once before game loop is entered

update(…) and draw(…) are called 60 times per second

unloadContent() is called once after the game loop terminates

All other calls to loadContent() and unloadContent() must be generated by you during level transitions

Model Details

(0, 0) is in the top-left corner (y axis inverted).

The trigonometry “quadrants” are also inverted.

VisibleObject (and all subclasses) have update(…) and draw(…) methods.

PositionedObject class stores bounding rectangle (used for collision detection, updated as object moves)

PositionedObject also has a touch(PositionedObject) method that is called on an object when it collides with another object.

MovingObject stores object’s speed and rotation (or direction).

> Rotation can be stored as angle, or as sine and cosine of angle (which is what you actually need to do movement calculations).

World and Viewport

See “Coordinate Systems” document.

The “world” is the current level. Each object has a position in the world (or level).

At any moment, only a subset of the world is visible on the screen. The rectangular region of the world that is currently visible on screen is called the “viewport”. Create a Viewport class to represent the currently-visible rectangular sub-region of the world.

The viewport remains centered on the ship. That is, as the ship moves in the world, the viewport follows it such that the ship remains at the viewport's center (except when the viewport runs into the edge of the world, at which point the ship will no longer be centered on the ship).

Updating model in the GameDelegate.update() method

GameDelegate.update(…) is called 60 times per second by the game loop. It should call update(…) on all VisibleObjects (including the Viewport). This will cause moving objects to move, and collisions to be handled.

Updating Asteroid and Projectile positions

Call GraphicsUtils.moveObject(…)

Call GraphicsUtils.ricochetObject(…)

Projectiles are similar except they don't ricochet (they disappear after leaving the world)

Updating the Ship rotation and position

Updating ship position and rotation based on user input (InputManager.movePoint & InputManager.firePressed)

1. Update rotation

InputManager.movePoint is reported in viewport coordinates. It must first be converted from Viewport to World coordinates.

movePointWorld = InputManager.movePoint + viewportPosWorld [top-left corner of viewport in world]

(dx, dy) = movePointWorld– shipPosWorld

```
If (dx != 0 && dy != 0)
        newShipRotation = Math.atan2(dy, dx)
else if (dx != 0)
        newShipRotation = ((dx > 0) ? 0 : Math.PI);
else if (dy != 0)
        newShipRotation = ((dy > 0) ? (0.5 * Math.PI) : (1.5 * Math.PI));
```

EXTRA CREDIT: Make ship rotate smoothly through smallest angle

2. Update position

Call GraphicsUtils.moveObject(…)

Don't allow the ship to move outside the world (i.e., if it moves out, bring it back in).

Updating the Viewport position

Center the viewport on the ship's updated position. Don't allow any part of the Viewport to leave the world (i.e., if one of its edges moves out of the world, bring it back in).

Handling Collisions

Collision detection done by intersecting PositionedObject bounding rectangles. When objects A and B collide, call A.touch(B) and B.touch(A).

Drawing visible objects in the draw() method

GameDelegate.draw(…) is called 60 times per second. It should call draw(…) on all VisibleObjects that are currently visible on the screen. This will cause all objects to be drawn in their updated positions.

All drawing is done with DrawingHelper, and should be done within the draw() method. Drawing outside of draw() won't work.

When drawing objects, only those objects that are within the viewport should be drawn (drawing invisible objects is a waste of time, because you won't see them anyway). Q: How can you tell if an object is visible? A: Calculate the intersection of the object's bounding rectangle and the Viewport rectangle.

When drawing objects, we must use Viewport coordinates instead of World coordinates.

objPosViewport = objPosWorld - viewportPosWorld

Use DrawingHelper.drawImage(…), etc. to draw objects.

Drawing rotated ship

See DrawingTheShip.pptx

Draw ship halo with DrawingHelper.drawFilledCircle(…).

Drawing the Mini-Map

Draw border by drawing a filled black rectangle on top of a filled red rectangle that is somewhat wider and taller than the black one [DrawingHelper.drawFilledRectangle(…)]. (Mini-map size is about 20% of the screen size.)

Draw green point for the ship and red points for asteroids (or something like that) [DrawingHelper.drawPoint(…)].

Draw each object at its world position scaled to the size of the mini-map.